ΤΕΙ ΠΑΤΡΑΣ
Σχολή Οικονομίας και Διοίκησης
Τμήμα: Επιχειρηματικού Σχεδιασμού & Πληροφοριακών Συστημάτων

Πτυχιακή εργασία:

Ανάπτυξη Εκπαιδευτικού Υλικού για τις Δομές Δεδομένων & Οργάνωση Αρχείων Εφαρμογές στη C++

Πάτρα 2006

Τριανταφυλλία Σταθαρά Α.Μ:404

Εποπτέων καθηγητής: **Α. Μπακάλης**

Περιεχόμενα

4 «ΕΝΤΟΛΕΣ»

5 «ΔΕΙΚΤΕΣ»

6 «ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ»

7 «ΣΥΝΑΡΤΗΣΕΙΣ»

8 «ΡΟΗ ΠΡΟΓΡΑΜΜΑΤΩΝ»

9 «ΠΙΝΑΚΕΣ»

10 «ΛΙΣΤΕΣ»

11 «ΣΤΟΙΒΑ»

12 «ΟΥΡΑ»

Πρόλογος

Η παρακάτω εργασία παρουσιάζει μια ανάπτυξη εκπαιδευτικού υλικού για τις δομές δεδομένων και οργάνωση αρχείων με εφαρμογή C++. Τα θέματα που αναλύονται είναι η ανατομία προγράμματος C++, οι τελεστές, οι εντολές, οι δείκτες, ο αντικειμενοστραφής προγραμματισμός, οι συναρτήσεις, η ροή προγραμμάτων, ο πίνακας, οι λίστες, οι στοίβες και οι ουρές. Στα παραπάνω γίνεται με προσέγγιση της γλώσσας προγραμματισμού C++. Καθένα από τα κεφάλαια ασχολείται με τους τρόπους υλοποίησης μιας συγκεκριμένης δομής δεδομένων και περιλαμβάνει ολοκληρωμένα προγράμματα σε C++. Το παρακάτω εκπαιδευτικό υλικό έχει σκοπό, να κατανοήσει ο ενδιαφερόμενος τις πρωτογενείς δομές δεδομένων που χρησιμοποιούνται για την παράσταση ή περιγραφή γεγονότων, για το πως οι δομές αυτές παρίστανται και αποθηκεύονται σε ένα σύστημα Η/Υ, αλλά και πως τις χρησιμοποιούμε. Περιλαμβάνει ορολογίες, παραδείγματα κατανόησης και ασκήσεις αξιολόγησης, με σκοπό τη διδασκαλία στις δομές δεδομένων σε εφαρμογή C++.

Αυτό το εκπαιδευτικό υλικό δεν αποτελεί μια λεπτομερή περιγραφή της C++ στις δομές δεδομένων, αλλά είναι μια εισαγωγή στα κυριότερα χαρακτηριστικά αυτής της γλώσσας και σας παρέχει ορισμένα παραδείγματα για να σας δείξει πως χρησιμοποιείται τη C++ για να λύσετε ορισμένα προβλήματα.

Εισαγωγή

Το **εκπαιδευτικό υλικό** σκοπό έχει την αυτοδιδασκαλία ή την ομαδική διδασκαλία σαν συμπληρωματική δυνατότητα, με ανοικτή και ευέλικτη διαδρομή στη μάθηση.

Στόχος του εκπαιδευτικού υλικού δεν είναι η αντικατάσταση ή ο περιορισμός του ρόλου του εκπαιδευτικού στην εκπαιδευτική διαδικασία, αλλά επιδιώκεται να καταστεί το δίκτυο ένα βοηθητικό εργαλείο με συμπληρωματικό στόχο στην ολοκλήρωση της διδακτικής πράξης.

Η αποστολή του είναι η εξ' αποστάσεως παροχή προπτυχιακής και μεταπτυχιακής εκπαίδευσης και επιμόρφωσης το οποίο να βασίζεται σε κατάλληλο εκπαιδευτικό υλικό και κατάλληλων μεθόδων για την από απόσταση μετάδοση της γνώσης.

Η ανάπτυξη εκπαιδευτικού υλικού πρέπει να είναι σε μια μορφή ώστε να είναι έτοιμο-μετασχηματίσιμο σε ηλεκτρονική μορφή κατάλληλη για τοπική χρήση στο διαδίκτυο.

Η δημιουργία ηλεκτρονικού υλικού ενισχύει τις πηγές πληροφόρησης, την ανάπτυξη και τη χρήση βελτιωμένων μεθόδων διδασκαλίας.

Το Ε.Α.Π ίδρυσε το Εργαστήριο Εκπαιδευτικού Υλικού και Εκπαιδευτικής Μεθοδολογίας. Σήμερα πολλά παραδοσιακά πανεπιστήμια έχουν αρχίσει τα τελευταία χρόνια την έρευνα και στον τομέα της μετάδοσης της γνώσης από απόσταση και της εφαρμογής των νέων τεχνολογιών διαχείρισης και μετάδοσης της πληροφορίας.

Η ελληνική ορολογία «**δεδομένα**», σύμφωνα με την Τεχνική Επιτροπή 48 του ΕΛΟΤ προτείνει την ακόλουθη ερμηνεία: Μια παράσταση γεγονότων, εννοιών ή εντολών σε τυποποιημένη μορφή που είναι κατάλληλη για επικοινωνία, ερμηνεία ή επεξεργασία από

τον άνθρωπο ή από αυτόματα μέσα. Πιο συγκεκριμένα η επίλυση οποιουδήποτε προβλήματος εμπλέκει τη χρησιμοποίηση πληροφοριών (δεδομένα). Η οργάνωση αυτών των δεδομένων είναι ένα πολύ σημαντικό κομμάτι της επίλυσης των προβλημάτων. Απαιτείται να προσδιοριστεί η συλλογή των δεδομένων, οι πιθανές σχέσεις που τα συνδέουν και οι βασικές λειτουργίες που πρέπει να εκτελούνται πάνω στα δεδομένα. Μια τέτοια συλλογή μαζί με τις σχέσεις και τις λειτουργίες καλείται **δομή δεδομένων**.

Η **C++** μπορεί να θεωρηθεί μια διαδικαστική γλώσσα με κάποιες επιπλέον δομές, μερικές από τις οποίες προστέθηκαν για αντικειμενοστραφή προγραμματισμό, ενώ άλλες για την βελτίωση του συντακτικού της γλώσσας. Η C++ είναι ουσιαστικά μια επεκτάσιμη γλώσσα αφού μπορούμε να ορίσουμε νέους τύπους με τέτοιο τρόπο ώστε να λειτουργούν σαν τους προκαθορισμένους τύπους, που είναι τμήμα της γλώσσας. Άλλες γλώσσες στην ίδια οικογένεια με τη C++ (1983) είναι η Simula67 (1967), η Smalltalk (1979) και η Java (1995).

Η C++ σχεδιάστηκε για την ανάπτυξη μεγάλων προγραμμάτων. Η C++ αναπτύχθηκε στα εργαστήρια Bell από τον Bjarne Stroustrup στις αρχές της δεκαετίας του 80. Η γλώσσα είναι κάτι πολύ περισσότερο από μια απλή επέκταση της C, διότι έχει σα βάση της τα αντικείμενα, ενώ η C θεωρείται διαδικαστική γλώσσα.

Πλεονεκτήματά της C++ σε σχέση με τη C είναι:

- η οργάνωση δεδομένων σε μια κλάση (*class*),
- η κληρονομικότητα (*inheritance*) στη δομή των κλάσεων,
- η δυνατότητα αρχικοποίησης δεδομένων,
- ο ορισμός μεταβλητών ως αναφορά (*reference*) σε κάποια άλλη,
- ο πολυμορφικός (*polymorphic*) καθορισμός συναρτήσεων με βάση τον τύπο του ορίσματός τους,
- η υπερφόρτιση (*overloading*) τελεστών,
- η πλούσια βιβλιοθήκη βασικών δομών δεδομένων και
- η δυνατότητα ορισμού πολυμορφικών βασικών τύπων με βάση ένα πρότυπο (*template*).

Σύμφωνα με τα παραπάνω η γλώσσα προγραμματισμού C++ υποστηρίζει τον αντικειμενοστραφή προγραμματισμό (ο αντικειμενοστραφής προγραμματισμός είναι

μια μεθοδολογία προγραμματισμού, όπου παρέχει υποστήριξη αυτής της μεθοδολογίας σε επίπεδο μεταγλωττιστή).

Υπάρχουν βέβαια και εφαρμογές που υποστηρίζονται καλύτερα σε C, όπως προγράμματα που έχουν άμεση σχέση με το υλικό όπως ο πυρήνας του λειτουργικού συστήματος και οι οδηγοί συσκευών, ενσωματωμένες εφαρμογές που εκτελούνται σε περιβάλλον που δεν μπορεί να ανταποκριθεί στις απαιτήσεις της C++.

Οι πιο διαδεδομένες γλώσσες προγραμματισμού στις εφαρμοσμένες επιστήμες είναι η Java, η Fortran, η C και η C++, όπου χρησιμοποιούν θέσεις στη μνήμη του υπολογιστή, για την αποθήκευση των ποσοτήτων (δεδομένων και αποτελεσμάτων) του προγράμματος. Ο υπολογιστής επιδρά στις τιμές αυτών των ποσοτήτων ακολουθώντας διαδοχικά τις εντολές που περιλαμβάνονται στο πρόγραμμα. Υπάρχει η δυνατότητα ανάθεσης τιμής στις μεταβλητές, επιλογής της εντολής που θα εκτελεστεί στο επόμενο βήμα, ανάλογα με κάποια συνθήκη, καθώς και η δυνατότητα επανάληψης μιας ή περισσότερων εντολών.

Ένας ηλεκτρονικός υπολογιστής έχει τη δυνατότητα να προγραμματιστεί ώστε να εκτελέσει μια συγκεκριμένη διαδικασία. **Προγραμματισμός** είναι η λεπτομερής περιγραφή, σε κάποια γλώσσα προγραμματισμού, των βημάτων που πρέπει να ακολουθήσει ώστε να ολοκληρώσει την επιθυμητή διεργασία. Το σύνολο των βημάτων, το πρόγραμμα δηλαδή, συνήθως απαιτεί δεδομένα που πρόκειται να επεξεργαστεί ώστε να παράγει κάποιο αποτέλεσμα, σχεδιάζεται, όμως, ανεξάρτητα από συγκεκριμένες τιμές των δεδομένων αυτών.

Κεφ.1 ΑΝΑΤΟΜΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ C++

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο, μαθαίνεται πως να μεταγλωττίζεται και να τρέχετε ένα πρόγραμμα, ώστε να τυπώνονται τα στοιχεία στην οθόνη.

Λέξεις κλειδιά

αρχεία προέλευσης
σύμβολο ανακατεύθυνσης
προεπεξεργαστής (preprocessor)
τελεστής εξόδου (insertion operator)
τελεστής εισόδου (extraction operator ())
cout
χαρακτήρας νέας γραμμής
συνένωση των δύο τιμών

Ορισμός

Η λέξη πρόγραμμα χρησιμοποιείται με δύο τρόπους: για να περιγράψει μεμονωμένες εντολές που δημιουργούνται από τον προγραμματιστή και για να περιγράψει ένα ολοκληρωμένο, εκτελέσιμο λογισμικό.

Το πρώτο σας πρόγραμμα σε C++

Πληκτρολογήστε τον παρακάτω κώδικα προέλευσης κατευθείαν στον κειμενογράφο σας. Αφού βεβαιωθείτε ότι είναι σωστός, αποθηκεύσετε το αρχείο, μεταγλωττίστε το, συνδέστε το και τρέξτε το. Θα τυπώσει τις λέξεις hello world στην οθόνη σας.

```
1. #include <iostream>
2.
3. int main()
4. {
5.     std::cout<< "Hello World!\n";
6.     char response;
7. std::cin >> response;
8. return 0;
9. }
```

Έξοδος:

Hello World!

Ανάλυση:

Βεβαιωθείτε ότι το έχετε γράψει ακριβώς όπως το βλέπετε. Τα αρχεία που δημιουργείτε με τον κειμενογράφο σας ονομάζονται **αρχεία προέλευσης** και στη c++ ονοματίζονται με την επέκταση του **.cpp** αλλά πρώτα ελέγξτε το μεταγλωττιστή σας να δείτε τι χρειάζεται διότι μερικοί μεταγλωττιστές χρειάζονται .cp, ή .c. Το << στη γραμμή 5 είναι το **σύμβολο ανακατεύθυνσης**, που παράγεται στα περισσότερα πληκτρολόγια με το πάτημα του shift και το πάτημα του κόμματος δύο φορές. Μεταξύ των γραμμάτων std και cout στη γραμμή 5 είναι δύο άνω και κάτω τελείες (:). Οι γραμμές 5 και 6 τελειώνουν με ελληνικό ερωτηματικό (;).

Εάν εμφανιστούν λάθη, κοιτάξτε προσεχτικά τον κώδικά σας και προσδιορίστε τη διαφέρει από την προηγούμενη λίστα. Εάν δείτε ένα λάθος στη γραμμή 1, όπως cannot find file iostream, εκλέξτε την τεκμηρίωση του μεταγλωττιστή σας για οδηγίες πως να ορίσετε τη διαδρομή ή τις μεταβλητές περιβάλλοντος .

Εάν εμφανιστεί ένα λάθος ότι δεν υπάρχει κανένα πρωτότυπο για το main, προσθέστε τη γραμμή int main(); αμέσως πριν από τη γραμμή 3. Σε αυτή την περίπτωση, θα πρέπει

να προσθέσετε αυτή τη γραμμή πριν από τη συνάρτηση main, σε κάθε πρόγραμμα. Οι περισσότεροι μεταγλωττιστές δεν το απαιτούν αυτό, αλλά μερικοί το χρειάζονται.

Μερικοί προγραμματιστές που χρησιμοποιούν ολοκληρωμένα περιβάλλοντα ανάπτυξης θα δουν ότι η εκτέλεση του προγράμματος αναβοσβήνει ένα παράθυρο που εξαφανίζεται χωρίς να μπορούν να δουν το αποτέλεσμα που παράγει το πρόγραμμα. Όταν συμβαίνει αυτό προσθέτουμε αυτές τις γραμμές στον πηγαίο κώδικα :

char response;

std::cin >> response;

Αυτές οι γραμμές κάνουν το πρόγραμμα να σταματήσει μέχρι να πληκτρολογήσετε έναν χαρακτήρα. Αυτό θα το χρησιμοποιούμε σε όλα τα προγράμματα .

Εάν έχετε πολύ παλιό μεταγλωττιστή το πρόγραμμα που είδατε παραπάνω δεν θα δουλέψει, γιατί δε θα βρει τις νέες τυπικές βιβλιοθήκες ANSI. Σε αυτή τη περίπτωση αλλάξτε την επικεφαλίδα ως εξής :

#include <iostream.h>

Επίσης δεν χρειάζεται να χρησιμοποιήσετε το std εμπρός από το cout στη γραμμή 5.

Ο πρώτος χαρακτήρας που βλέπετε στο πρόγραμμα είναι το σύμβολο #, το οποίο είναι ένα σήμα σε ένα πρόγραμμα που ονομάζεται **προεπεξεργαστής (preprocessor)**. Κάθε φορά που ξεκινάτε το μεταγλωττιστή σας, πρώτα τρέχει ο προεπεξεργαστής. Ο προεπεξεργαστής διαβάζει τον κώδικα προέλευσης, ψάχνοντας για γραμμές που αρχίζουν με το σύμβολο # και ενεργεί σε αυτές τις γραμμές πριν τρέξει ο μεταγλωττιστής.

Η πρόταση **#include** λέει στον μεταγλωττιστή να ενσωματώσει τα περιεχόμενα ενός άλλου αρχείου στην αρχή του προγράμματός μας, δηλ. εδώ τα περιεχόμενα του αρχείου iostream. Είναι μια οδηγία για τον προεπεξεργαστή που λέει «αυτό που ακολουθεί είναι ένα όνομα αρχείου. Βρες αυτό το αρχείο, διάβασε το και βγάλτο εδώ».

Οι γωνιώδες αγκύλες γύρω από το όνομα του αρχείου λένε στον προεπεξεργαστή να κοιτάξει όλες τις συνηθισμένες θέσεις για αυτό το αρχείο.

Το αρχείο **iostream** (συντομογραφία για το Input-Output-System) χρησιμοποιείτε για το cout, το οποίο γράφει στην οθόνη.

Η γραμμή 3 ξεκινά στο πρόγραμμα με μια συνάρτηση που ονομάζεται **main( )**. Κάθε πρόγραμμα στη C++ έχει μια συνάρτηση main(). Μια συνάρτηση είναι μια ένα τμήμα του κώδικα που εκτελεί μια ή περισσότερες ενέργειες. Όταν ξεκινά το πρόγραμμά σας η main() καλείται αυτόματα.

Η main() όπως και όλες οι συναρτήσεις, πρέπει να δηλώσει τι είδος τιμή θα επιστρέψει. Ο τύπος της τιμής επιστροφής της main() στο hello.cpp είναι **int**, το οποίο σημαίνει ότι αυτή η συνάρτηση όταν τελειώσει θα επιστρέψει έναν ακέραιο αριθμό στο, λειτουργικό σύστημα.

Όλες οι συναρτήσεις αρχίζουν και τελειώνουν με ένα αριστερό **({)**και ένα δεξιό άγκιστρο **(})**. Οτιδήποτε είναι μεταξύ του αριστερού και του δεξιού άγκιστρου θεωρείτε μέρος της συνάρτησης.

Το αντικείμενο **cout** χρησιμοποιείτε για να τυπώσει ένα μήνυμα στην οθόνη. Αυτός ο τελεστής ονομάζεται **τελεστής εξόδου** (**insertion operator)**. Η λέξη cout αντιπροσωπεύει ένα ρεύμα εξόδου (output stream), το οποίο η C++ συνδέει με την καθιερωμένη συσκευή εξόδου (standard output) του λειτουργικού συστήματος, που μπορεί να είναι η οθόνη, ο εκτυπωτής, ένα αρχείο ή και η είσοδος ενός άλλου προγράμματος. Τα αντικείμενα **cout** και **cin** χρησιμοποιούνται για να χειρίζονται την είσοδο και την έξοδο. Το cout και το cin είναι συμβολοσειρές (οι οποίες ορίζονται εντός του iostream) και από **default** γίνεται αναφορά σε αυτές και στο βασικό ρεύμα δεδομένων εισόδου (το οποίο είναι συνδεδεμένο με την οθόνη) και στο βασικό ρεύμα δεδομένων εξόδου (το οποίο είναι συνδεδεμένο με το πληκτρολόγιο) αντίστοιχα. Αυτά τα ρεύματα ανοίγουν αυτόματα όταν το πρόγραμμα της C++ εκτελείται και είναι, επομένως, διαθέσιμα για να τα χρησιμοποιήσουμε σε οποιοδήποτε πρόγραμμα γράψουμε. Το αντικείμενο **cin** ονομάζεται **τελεστής εισόδου** (**extraction operator ())**.

Μια ματιά στην cout

Η λέξη cout είναι ένα ρεύμα δεδομένων εξόδου το οποίο η C++ συσχετίζει με τη βασική συσκευή εξόδου του συστήματος λειτουργίας. Από default το σύστημα λειτουργίας συσχετίζει τη βασική συσκευή εξόδου με τη λειτουργία εμφάνισης στην οθόνη. Για να τυπώσετε μια τιμή στην οθόνη γράφετε τη λέξη cout, ακολουθούμενη από τον τελεστή εισαγωγής (<<).

Η παρακάτω λίστα επεξηγεί πως ακριβώς λειτουργεί.

```
1. #include <iostream>
2. int main()
3. {
```

```
4.  std::cout <<"hello there.\n";
5.  std::cout <<"here is 1: "<< 1 << "\n";
6.  std::cout <<"the manipulator std::endl ";
7.  std::cout <<"writes a new line in the screen.";
8.  std::cout << std::endl;
9.  std::cout << "here is a very big number: \t"<< 70000;
10. std::cout << std::endl;
11. std::cout <<"here is the sum of 8 and 5:\t";
12. std::cout << 8+5 <<std::endl;
13. std::cout << "here's a fraction:\t\t";
14. std::cout <<(float) 5/8 <<std::endl;
15. std::cout <<"and a very big number:\t";
16. std::cout <<(double) 7000*7000 <<std::endl;
17. char response;
18. std::cin >> response;
19. return 0;
20. }
```

Έξοδος :

hello there.

here is 1: 1

the manipulator std::endl writes a new line in the screen.

here is a very big number: 70000

here is the sum of 8 and 5: 13

here's a fraction: 0.625

and a very big number: 4.9e+007

Ανάλυση:

Στη γραμμή 1, η πρόταση #include <iostream> προσθέτει το αρχείο iostream στον κώδικα προέλευσης. Αυτό είναι απαραίτητο εάν χρησιμοποιείτε το cout και τις σχετικές συναρτήσεις.

Το σύμβολο **n** είναι ένας ειδικός χαρακτήρας μορφοποίησης που λέει στην cout να τυπώσει ένα χαρακτήρα αλλαγής γραμμής στην οθόνη και ονομάζεται **χαρακτήρας νέας γραμμής**.

Στη γραμμή 5 το κενό μετά από την άνω και κάτω τελεία αποτελεί μέρος της συμβολοσειράς. Επειδή δεν υπάρχει κανένας χαρακτήρας αλλαγής γραμμής μετά την πρώτη συμβολοσειρά, η επόμενη τιμή τυπώνεται αμέσως μετά. Αυτό ονομάζεται **συνένωση των δύο τιμών**.

Ο σκοπός του **endl** είναι να γράψει μια νέα γραμμή στην οθόνη. Στη γραμμή 9, εισάγεται ένας νέος χαρακτήρας μορφοποίησης, ο **t**. Αυτό παρεμβάλει έναν χαρακτήρα στηλοθέτη που χρησιμοποιείτε στις γραμμές 9 έως 15 για να στοιχίσει την έξοδο.

Ο όρος **(float)** λέει στη cout ότι θέλετε αυτή η τιμή να υπολογίζεται σαν ένας δεκαδικός αριθμός και έτσι τυπώνεται ένα κλάσμα. Στη γραμμή 16, δίνεται η τιμή 7000*7000 στην cout και χρησιμοποιείτε ο όρος **(double)** για να πει στην cout ότι αυτή είναι μια τιμή κινητής υποδιαστολής.

Παρατηρείτε ότι είναι ενοχλητικό να χρησιμοποιείτε συνέχεια το std::. Ένας πιο εύκολος τρόπος για την παραπάνω λίστα είναι να προσθέσετε το:

using std::cout;

using std::endl;

μετά από τη main() και αντί για std::cout να γράψετε cout.

Η μόνη διαφορά είναι ότι στις γραμμές 4 και 5 προσθέσαμε τις εντολές που ενημερώνουν τον μεταγλωττιστή ότι θα χρησιμοποιηθούν δύο αντικείμενα από την τυπική βιβλιοθήκη. Αυτό γίνεται με τη λέξη **using**.

Ένας άλλος τρόπος για να αποφεύγουμε το std:: είναι να γράψετε:

using namespace std;

μετά από τη main() και μετά για τη τύπωση το cout.

Τα πλεονεκτήματα με το **using namespace std;** είναι ότι δεν χρειάζεται να αναφέρεται τα αντικείμενα που χρησιμοποιείτε, το μειονέκτημα είναι ότι διατρέχετε τον κίνδυνο να χρησιμοποιήσετε, χωρίς να το θέλετε, αντικείμενα από μια βιβλιοθήκη που δεν θέλετε.

Τύποι σχολίων

Τα σχόλια της C++ αποτελούνται από μια διπλή κάθετο (//) την οποία ακολουθεί μια σειρά χαρακτήρων. Οτιδήποτε υπάρχει μετά από αυτό το σημείο μέχρι το τέλος της ίδιας σειράς είναι το σχόλιο.

Για παράδειγμα:

// **This is a C++ comment** , είναι ένα σχόλιο της C++.

Παρατηρείστε ότι η C++ ακολουθεί τη μορφή σχολίων της C και μπορείτε να συμπεριλάβετε ένα σχόλιο της C++ εντός ενός σχολίου της C. Παρόλα αυτά σας προτείνουμε να χρησιμοποιείτε C++ σχόλια για C++ προγράμματα και C σχόλια για C προγράμματα. Ακόμη, παρατηρείστε ότι εάν ένα σχόλιο ξεπερνάει τη μια γραμμή, η διπλή κάθετος πρέπει να εμφανίζεται στην αρχή κάθε γραμμής.

Τα σχόλια των πολλών γραμμών ξεκινούν με μια κάθετο ακολουθούμενη από έναν αστερίσκο **(/*)**. Αυτό το σχόλιο λέει στο μεταγλωττιστή να αγνοήσει οτιδήποτε ακολουθεί, έως όπου βρει ένα αστερίσκο και μια κάθετο **(*/)**. Αυτά τα σημάδια μπορούν να είναι στην ίδια γραμμή ή μπορούν να εκτείνονται σε μια ή περισσότερες γραμμές. Ωστόσο, κάθε /* πρέπει να έχει και ένα αντίστοιχο */.

```
1.  #include <iostream>
2.  int main()
3.  {
4.      using std::cout;
5.      /* αυτό είναι ένα σχόλιο
6.      που εκτείνεται μέχρι τον αστερίσκο-κάθετο*/
7.      cout << "hello world\n";
8.      //αυτό είναι ένα σχόλιο που τελειώνει στο τέλος της γραμμής
9.      cout << "that comment ended\n";
10.     //τα σχόλια με την διπλή κάθετο μπορεί να είναι μόνα τους στη γραμμή
11.     /* όπως και τα σχόλια με τη κάθετο-αστερίσκο*/
12.     char response;
13. std::cin >> response;
14. return 0;
15. }
```

Έξοδος :

hello world

that comment ended

Ασκήσεις αξιολόγησης

Άσκηση 1

Γράψτε ένα πρόγραμμα που να λέει «Run C++» στην οθόνη.

Άσκηση 2

Τι λάθη εντοπίζετε στον παρακάτω κώδικα C++ ;

```
1. include iostream.h
2.
3. Main();
4.  {
5.    Double x,y,z;
6.    cout < "Enter two numbers ";
7.    cin >> a >> b
8.    cout << 'The numbers in reverse order are' << b,a;
9.  }
```

Κεφ.2 ΜΕΤΑΒΛΗΤΕΣ ΚΑΙ ΣΤΑΘΕΡΕΣ

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο θα μάθετε πως να δηλώνετε και να ορίζετε τις μεταβλητές και σταθερές, πως να δίνετε τιμές στις μεταβλητές και πως να χειρίζεστε αυτές τις τιμές καθώς επίσης και πως να γράφετε την τιμή μιας μεταβλητής στην οθόνη.

Λέξεις κλειδιά

Signed και Unsigned
Short και when
χαρακτήρας διαφυγής
κυριολεκτικές, συμβολικές και απαρίθμησες σταθερές
μπλοκ

Ορισμός μιας μεταβλητής

Στη C++ μια μεταβλητή είναι μια θέση που αποθηκεύονται πληροφορίες. Μια μεταβλητή είναι μια θέση στη μνήμη του υπολογιστή σας στην οποία μπορείτε να αποθηκεύσετε μια τιμή και από την οποία μπορείτε αργότερα να ανακτήσετε αυτή την τιμή.

Παρατηρήστε ότι οι μεταβλητές χρησιμοποιούνται για προσωρινή αποθήκευση. Όταν βγαίνετε από το πρόγραμμα ή κλείνετε τον υπολογιστή σας, αυτές οι μεταβλητές χάνονται. Η μόνιμη αποθήκευση είναι κάτι διαφορετικό. Γενικά οι μεταβλητές αποθηκεύονται μόνιμα είτε σε μια βάση δεδομένων, είτε σε ένα αρχείο στον δίσκο.

Δημιουργείτε ή ορίζετε μια μεταβλητή αναφέροντας τον τύπο της, ακολουθούμενο από ένα ή περισσότερα κενά, ακολουθούμενο από το όνομα της μεταβλητής και ένα ελληνικό ερωτηματικό. Το όνομα της μεταβλητής μπορεί να έχει οποιοδήποτε συνδυασμό γραμμάτων, αλλά μπορεί να περιέχει και κενά. Η παρακάτω εντολή ορίζει μια ακέραιη μεταβλητή που ονομάζεται myAge:

int myAge

Όταν δηλώνεται μια μεταβλητή, δεσμεύεται μνήμη για αυτή τη μεταβλητή. Η τιμή της μεταβλητής θα είναι ίση με αυτό που περιέχει η μνήμη αυτή τη στιγμή.

Παράδειγμα:

```
1. int main()
2. {
3. unsigned short x;
4. unsigned short y;
5. unsigned short z;
6. z = x * y;
7. return 0;
8. }
```

Βασικοί τύποι μεταβλητών

Διάφοροι τύποι μεταβλητών είναι ενσωματωμένοι στη C++. Αυτοί μπορεί να διαιρεθούν σε ακέραιες μεταβλητές, σε μεταβλητές κινητής υποδιαστολής και σε μεταβλητές χαρακτήρων.

Οι μεταβλητές κινητής υποδιαστολής έχουν τιμές που μπορούν να εκφραστούν σε κλάσματα, δηλαδή, είναι πραγματικοί αριθμοί. Οι μεταβλητές χαρακτήρων δεσμεύουν μόνο ένα byte και γενικά χρησιμοποιούνται για να περιέχουν 256

χαρακτήρες και σύμβολα μόνο από το σύνολο χαρακτήρων ASCII και εκτεταμένο σύνολο χαρακτήρων ASCII.

Ο παρακάτω πίνακας παρουσιάζει τον τύπο κάθε μεταβλητής, πόσο χώρο καταλαμβάνει αυτός ο τύπος στη μνήμη και ποια είδη τιμών μπορούν να αποθηκευτούν σε αυτές τις μεταβλητές. Οι τιμές που μπορούν να αποθηκευτούν προσδιορίζονται από το μέγεθος των τύπων των μεταβλητών.

Τα μεγέθη των μεταβλητών μπορεί να είναι διαφορετικά από αυτά του πίνακα, ανάλογα με το μεταγλωττιστή και τον υπολογιστή που χρησιμοποιείτε.

Τύπος	**Μέγεθος**	**Τιμές**
bool	1 byte	True ή false
Unsigned short int	2 byte	0 έως 65.535
Short int	2 byte	-32.768 έως 32.767
Unsigned long int	4 byte	0 έως 4.294.967.295
Long int	4 byte	-2.147.483.648 έως 2.147.483.647
Int (16 bit)	2 byte	-32.768 έως 32.767
Int (32 bit)	4 byte	-2.147.483.648 έως 2.147.483.647
Unsigned int(16 bit)	2 byte	0 έως 65.535
Unsigned int(32 bit)	4 byte	0 έως 4.294.967.295
Char	1 byte	256 τιμές χαρακτήρων
Float	4 byte	1.2e-38 έως 3.4e38
Double	8 byte	2.2.e-308 έως 1.8e308

Παράδειγμα:

```
1. #include <iostream>
2. int main()
3. {
4.         // δήλωση μεταβλητών
5.         int year = 32;
6.         float salary = 350000.00;
7.
```

```
8.          // κύριο σώμα του προγράμματος
9.          std::cout << "i am " << year << " years old" << std::endl;
10.         std::cout << " and my salary is " << salary << " euro" <<std::endl;
11.
12. char response;
13. std::cin >> response;
14. return 0;
15. }
```

Έξοδος :

i am 32 years old

and my salary is 350000 euro

Ανάλυση:

Η γραμμή 1 περιλαμβάνει την απαραίτητη πρόταση include και τη βιβλιοθήκη του iostream, έτσι ώστε να δουλεύει το cout. Η γραμμή 2 ξεκινά το πρόγραμμα με τη συνάρτηση main(). Οι γραμμές 5 και 6 δηλώνουν τις δύο μεταβλητές. Η μία είναι τύπου int δηλαδή ακέραιες τιμές και η δεύτερη είναι τύπου float δηλαδή μεγάλες πραγματικές τιμές. Οι γραμμές 9 και 10 ορίζουν το cout και το endl ως μέρος του τυπικού χώρου ονομάτων (std).

SIGNED και UNSIGNED

Όλοι οι τύποι ακέραιων αριθμών ανήκουν σε δύο ποικιλίες : signed και unsigned. Η ιδέα εδώ είναι ότι μερικές φορές χρειάζεστε τους αρνητικούς αριθμούς και μερικές φορές όχι.

Οι ακέραιοι αριθμοί (short and long) χωρίς τη λέξη «unsigned» υποτίθεται ότι είναι signed. Οι signed ακέραιοι αριθμοί είναι είτε αρνητικοί είτε θετικοί. Οι unsigned ακέραιοι αριθμοί είναι πάντα θετικοί.

Ένας unsigned short ακέραιος αριθμός μπορεί να χειριστεί τους αριθμούς από 0 έως 65,535. Τους μίσους από τους αριθμούς που αντιπροσωπεύονται μπορεί από signed short να είναι αρνητικός, κατά συνέπεια οι signed short μπορεί μόνο να αντιπροσωπεύσει τους αριθμούς από -32.768 έως 32.767.

Μέγεθος των ακέραιων αριθμών

Σε κάθε υπολογιστή, κάθε μεταβλητός τύπος απορροφάει ένα αμετάβλητο μέρος στη μνήμη. Δηλαδή ένας ακέραιος αριθμός είναι δύο bytes σε μια μηχανή, και τέσσερις σε άλλη, αλλά σε καθέναν υπολογιστή είναι πάντα ο ίδιος.

Ένας short ακέραιος αριθμός είναι δύο bytes στους περισσότερους υπολογιστές, ένας long ακέραιος αριθμός είναι συνήθως τέσσερις bytes, και ένας ακέραιος αριθμός μπορεί να είναι δύο ή τέσσερις bytes.

Καθορισμός μιας μεταβλητής

Δημιουργείτε ή καθορίζετε μια μεταβλητή με τη δήλωση του τύπου της, που ακολουθείτε από ένα ή περισσότερα διαστήματα, που ακολουθούνται από το μεταβλητό όνομα και μια άνω τελεία.

Το μεταβλητό όνομα μπορεί να είναι ουσιαστικά οποιοσδήποτε συνδυασμός επιστολών, αλλά δεν μπορεί να περιέχει τα διαστήματα. Τα νομικά μεταβλητά ονόματα περιλαμβάνουν το X, J23qrsnf, και το myAge. Η ακόλουθη δήλωση καθορίζει έναν ακέραιο αριθμό μεταβλητό αποκαλούμενο myAge:

Int myAge;

Προσπαθήστε να χρησιμοποιήσετε τα εκφραστικά ονόματα όπως myAge ή howMany. Τέτοια ονόματα είναι ευκολότερα να καταλάβετε.

Δείτε τα παρακάτω παραδείγματα:

```
1. main()
2. {
3. unsigned short x;
4. unsigned short y;
5. ULONG z;
6. z = x * y;
7. }
```

Παράδειγμα 2:

```
1. main ()
2. {
3. unsigned short Width;
```

4. unsigned short Length;
5. unsigned short Area;
6. Area = Width * Length;
7. }

Λέξεις κλειδιά

Μερικές λέξεις είναι δεσμευμένες από τη C++ και δεν μπορείτε να τις χρησιμοποιήσετε ως ονόματα μεταβλητών. Αυτές είναι οι λέξεις κλειδιά που χρησιμοποιούνται από το μεταγλωττιστή της C++. Οι λέξεις κλειδιά περιλαμβάνουν τις if, while, for και main. Ο παρακάτω πίνακας είναι μια λίστα των λέξεων κλειδιά της C++.

Οι λέξεις κλειδιά της C++

Asm	Else	New	This
Auto	Enum	Operator	Throw
Bool	Explicit	Private	True
Break	Export	Protected	Try
Case	Extern	Public	Typedef
Catch	False	Register	Typeid
Char	Float	Reinterpret_cast	Typename
Class	For	Return	Union
Const	Friend	Short	Unsigned
Const_cast	Goto	Signed	Using
Continue	If	Sizeof	Virtual
Default	Inline	Static	Void
Delete	Int	Static_cast	Volatile
Do	Long	Struct	Wchar_t
Double	Mutable	Switch	While
Dynamic_cast	Namespace	Template	

Επιπλέον είναι δεσμευμένες οι παρακάτω λέξεις:

And	Bitor	Not_eq	Xor
And_eq	Compl	Or	Xor_eq
Bitand	Not	Or_eq	

Πότε να χρησιμοποιείς short και πότε when

Μια πηγή σύγχυσης για τους νέους προγραμματιστές C++ είναι πότε να δηλώσει μια μεταβλητή ως long και πότε short. Ο κανόνας είναι: εάν υπάρχει οποιαδήποτε πιθανότητα ότι η αξία που θα θελήσετε να βάλετε στη μεταβλητή σας θα είναι πάρα πολύ μεγάλη για τον τύπο της, να χρησιμοποιήσει έναν μεγαλύτερο τύπο.

Όπως είδαμε και στον παραπάνω πίνακα οι unsigned short ακέραιοι αριθμοί, που υποθέτουν ότι είναι δύο bytes, μπορούν να κρατήσουν μια αξία μόνο μέχρι 65.535. Οι Signed short ακέραιοι αριθμοί μπορούν να κρατήσουν μόνο μισού αυτού. Αν και οι unsigned long ακέραιοι αριθμοί μπορούν να κρατήσουν έναν εξαιρετικά μεγάλο αριθμό (4.294.967.295) που είναι ακόμα αρκετά πεπερασμένος. Ακόμα και τα διπλάσια μπορούν να κρατήσουν τους εξαιρετικά μεγάλους αριθμούς, αλλά μόνο τα πρώτα 7 ή 19 ψηφία είναι σημαντικά στους περισσότερους υπολογιστές. Αυτός σημαίνει ότι ο αριθμός στρογγυλεύεται μακριά μετά από αυτόν πολλά ψηφία.

Παράδειγμα:

```
1.  #include <iostream>
2.   int main()
3.   {
4.     unsigned short int smallNumber;
5.     smallNumber = 65535;
6.     std::cout << "small number:" << smallNumber << std::endl;
7.     smallNumber++;
8.     std::cout << "small number:" << smallNumber << std::endl;
9.     smallNumber++;
10.    std::cout << "small number:" << smallNumber << std::endl;
11.    char response;
```

```
12.    std::cin >> response;
13.    return 0;
14. }
```

Έξοδος :

small number:65535

small number:0

small number:1

Ανάλυση :

Στη γραμμή 4, smallNumber δηλώνεται για να είναι ένα unsigned short int, το οποίο στον υπολογιστή είναι μια δύο- byte μεταβλητή, ικανή να κρατήσει μια αξία μεταξύ 0 και 65.535

Στη γραμμή 5, η μέγιστη αξία ορίζεται σε smallNumber, και είναι τυπωμένη στη γραμμή 6. Στη γραμμή 7, το smallNumber αυξάνεται κατά 1. Το σύμβολο για την αύξηση είναι ++. Κατά συνέπεια, η αξία σε smallNumber θα ήταν 65.536. Παρολαυτά, οι unsigned short ακέραιοι αριθμοί δεν μπορούν να κρατήσουν έναν αριθμό μεγαλύτερο από 65.535. Στη γραμμή 9 ο smallNumber αυξάνεται πάλι, και έπειτα η νέα αξία της 1, είναι τυπωμένη.

Ειδικοί χαρακτήρες εκτύπωσης

Ο μεταγλωττιστής της C++ αναγνωρίζει μερικούς ειδικούς χαρακτήρες για μορφοποίηση. Ο παρακάτω πίνακας παρουσιάζει τους πιο κοινούς. Τους ειδικούς χαρακτήρες τους τοποθετείται στον κώδικα σας πληκτρολογώντας την ανάποδη κάθετο (που ονομάζεται **χαρακτήρας διαφυγής**) και μετά βάζετε το χαρακτήρα. Οι ειδικοί χαρακτήρες εκτύπωσης χρησιμοποιούνται κατά την εκτύπωση στην οθόνη, σε αρχείο ή σε άλλη συσκευή εξόδου. Ένας χαρακτήρας διαφυγής **(\ )** αλλάζει τη σημασία του χαρακτήρα που ακολουθεί.

Χαρακτήρας	**Περιγραφή**
\a	Ηχητική ειδοποίηση.
\b	Διάστημα προς τα πίσω(backspace).
\f	Τροφοδοσία φόρμας (form feed)
\n	Νέα γραμμή. Τοποθετεί τον δρομέα στην αρχή της επόμενης γραμμής
\r	Επιστροφή γραμμής .Τοποθετεί τον δρομέα στην αρχή της τρέχουσας γραμμής και δεν προχωρά στην επόμενη γραμμή.
\t	Στηλοθέτης .Μετακινεί τον δρομέα στον επόμενο στηλοθέτη.
\v	Κατακόρυφος στηλοθέτης (vertical tab).
\'	Μονά εισαγωγικά (single quote).
\"	Διπλά εισαγωγικά (double quote).
\?	Ερωτηματικό (question mark)
\\	Ανάποδη κάθετος. Χρησιμοποιείται για να τυπώνεται ένας χαρακτήρας ανάποδης καθέτου (bachslash).
\000	Οκταδική σύνταξη (octal notation).
\xhh	Δεκαεξαδική (hexadecimal).

Σταθερές

Όπως και οι μεταβλητές, έτσι και οι σταθερές είναι θέσεις μνήμης. Αντίθετα από τις μεταβλητές, οι σταθερές δεν αλλάζουν αλλά παραμένουν σταθερές. Πρέπει να αρχικοποιήστε μια σταθερά όταν τη δημιουργείτε και δεν μπορείτε να της εκχωρήσετε μια νέα τιμή αργότερα. Η C++ έχει τρείς τύπους σταθερών: κυριολεκτικές, συμβολικές και απαρίθμητες.

- **Κυριολεκτικές**: μια κυριολεκτική σταθερά είναι μια τιμή που δακτυλογραφείτε κατευθείαν στο πρόγραμμα σας, οπουδήποτε απαιτείται.

Παράδειγμα:

int myAge = 39;

Το myAge είναι μια μεταβλητή τύπου int και το 39 είναι μια κυριολεκτική σταθερά.

- **Συμβολικές**: μια συμβολική σταθερά είναι μια σταθερά που αντιπροσωπεύεται από ένα όνομα, ακριβώς όπως μια μεταβλητή, ωστόσο, αφού μια σταθερά αρχικοποιηθεί, η τιμή της δεν μπορεί να αλλάξει..

Εάν το πρόγραμμά σας έχει μια ακέραιη μεταβλητή που ονομάζεται students και μια άλλη που ονομάζεται classes, θα μπορούσατε να υπολογίσετε πόσους σπουδαστές έχετε, λαμβάνοντας υπόψη ένα γνωστό αριθμό τάξεων, εάν ξέρετε ότι κάθε τάξη αποτελείτε από 15 σπουδαστές :

Students = classes * 15;

Εδώ το 15 είναι μια κυριολεκτική σταθερά.

- **Απαρίθμητες**: Οι απαρίθμητες σταθερές σας επιτρέπουν να δημιουργείτε νέους τύπους και έπειτα να ορίζετε μεταβλητές με αυτούς τους τύπους, των οποίων οι τιμές είναι περιορισμένες σε ένα σύνολο πιθανών τιμών.

Η σύνταξη για τις απαρίθμητες σταθερές είναι να γράψετε τη λέξη κλειδί **enum**, ακολουθούμενη από το όνομα του τύπου, ένα αριστερό άγκιστρο, κάθε μια από τις τιμές χωρισμένες μα κόμματα και τελικά, ένα δεξιό άγκιστρο και ένα ελληνικό ερωτηματικό. Ακολουθεί ένα παράδειγμα:

Enum COLOUR { RED, BLUE, GREEN, WHITE, BLACK };

Αυτή η πρόταση εκτελεί δύο εργασίες :

1. ορίζει το COLOUR ως όνομα μιας απαρίθμητης, δηλαδή ορίζει ένα νέο τύπο δεδομένων.
2. κάνει το RED μια συμβολική σταθερά με τιμή 0, το BLUE μια συμβολική σταθερά με τιμή 1, το GREEN μια συμβολική σταθερά με τιμή 2 κ.ο.κ.

Κάθε απαρίθμητη σταθερά έχει μια ακέραιη τιμή. Εάν δεν καθορίσετε κάτι άλλο, η πρώτη σταθερά θα έχει τιμή 0 και οι υπόλοιπες θα συνεχίζουν από εκεί. Οποιαδήποτε από τις σταθερές μπορεί να αρχικοποιηθεί με μια ιδιαίτερη τιμή, ωστόσο και αυτές που δεν αρχικοποιούνται θα συνεχίσουν να μετρούν σε συνέχεια από τις προηγούμενες τιμές. Κατά συνέπεια αν γράψετε :

Enum COLOUR { RED=100, BLUE, GREEN=500, WHITE, BLACK=700 };

Τότε το REDθα έχει τιμή 100, το BLUE θα έχει τιμή 101, το GREEN τιμή 500, το WHITE τιμή 501 και το BLACK τιμή 700.

Χρησιμοποιώντας κενά διαστήματα

Τα κενά διαστήματα είναι αόρατοι χαρακτήρες, όπως οι στηλοθέτες, τα κενά και οι αλλαγές γραμμών. Αυτοί ονομάζονται «**κενά διαστήματα**» επειδή όταν τυπώνονται σε λευκό χαρτί δεν βλέπετε τίποτα.

Τα κενά διαστήματα αγνοούνται γενικά στις προτάσεις. Τα κενά διαστήματα μπορούν να χρησιμοποιηθούν για να κάνουν τα προγράμματα σας πιο ευανάγνωστα και ευκολότερα στη συντήρηση.

Μπλοκ και σύνθετες προτάσεις

Μια σύνθετη πρόταση ονομάζεται «μπλοκ». Ένα μπλοκ αρχίζει με ένα αριστερό άγκιστρο ({) και τελειώνει με ένα δεξιό άγκιστρο (}). Αν και κάθε πρόταση στο μπλοκ πρέπει να τελειώνει με ένα ελληνικό ερωτηματικό, το ίδιο το μπλοκ δεν τελειώνει με ένα ελληνικό ερωτηματικό, όπως φαίνεται στο παρακάτω παράδειγμα.

```
{
  temp = a;
  a = b;
  b = temp;
}
```

Αυτό το μπλοκ κώδικα ενεργεί ως μια πρόταση και ανταλλάσσει τις τιμές στις μεταβλητές a και b.

Εκφράσεις

Οτιδήποτε υπολογίζει μια τιμή είναι μια έκφραση στην C++. Μια έκφραση λέμε ότι επιστρέφει μια τιμή. Κατά συνέπεια, η πρόταση 3+2; επιστρέφει την τιμή 5, έτσι είναι μια έκφραση. Όλες οι εκφράσεις είναι προτάσεις .

Οποιαδήποτε έκφραση μπορεί να χρησιμοποιηθεί στη δεξιά πλευρά ενός τελεστή εκχώρησης. Το παρακάτω είναι απολύτως σωστό στη C++:

Επειδή είναι μια έκφραση, μπορεί να είναι στη σωστή πλευρά ενός τελεστή εκχώρησης :

```
Y = x  = a + b;
```

Αυτή η γραμμή υπολογίζεται στην παρακάτω σειρά:

Πρόσθεσε το a στο b.

Δώσε το αποτέλεσμα της έκφρασης a + b στο χ.

Δώσε το αποτέλεσμα της έκφρασης εκχώρησης x = a + b στο Y

Εάν το a,b,x και y είναι όλοι ακέραιοι αριθμοί και εάν το a έχει τιμή 9 και το b έχει τιμή 7, τότε το x και το y θα πάρουν τιμή 16. Αυτό φαίνεται στην παρακάτω λίστα.

```
1.  # include <iostream>
2.  int main()
3.  {
4.      using std::cout;
5.      using std::endl;
6.
7.      int a=0, b=0, x=0, y=35;
8.      cout << "a: " << a << " b: " << b;
9.      cout << " x: " << x << " y: " << y<< endl;
10.     a=9;
11.     b=7;
12.     y=x=a+b;
13.     cout << "a: " << a << " b: " << b;
14.     cout << " x: " << x << " y: " << y<< endl;
15.     char response;
16.     std::cin >> response;
17.     return 0;
18. }
```

Έξοδος :

a: 0 b: 0 x: 0 y: 35

a: 9 b: 7 x: 16 y: 16

Ανάλυση :

Στη γραμμή 7 δηλώνονται και αρχικοποιούνται οι τέσσερις μεταβλητές. Οι τιμές τυπώνονται στις γραμμές 8 και 9. Στη γραμμή 10 το a παίρνει την τιμή 9. Στη γραμμή 11 το b παίρνει την τιμή 7.Στη γραμμή 12 , αθροίζονται οι τιμές του a και του b και το αποτέλεσμα αποδίδεται στο χ. Αυτή η έκφραση x=a+b δίνει μια τιμή και αυτή η τιμή αποδίδεται στο y. Στις γραμμές 13 και 14, επιβεβαιώνονται αυτά τα αποτελέσματα τυπώνοντας τις τιμές των τεσσάρων μεταβλητών.

Ασκήσεις αξιολόγησης

Άσκηση 1

Ποίος θα ήταν ο σωστός τύπος μεταβλητής για να αποθηκευτούν οι πληροφορίες :

1. το βάρος σας
2. το εμβαδόν του σπιτιού σας
3. ο αριθμός των αστεριών του γαλαξία
4. οι μέσες βροχοπτώσεις τον μήνα Ιανουάριο.

Άσκηση 2

Δηλώστε μια σταθερά για το pi που να είναι ίση με 3.14.

Άσκηση 3

Δηλώστε μια μεταβλητή float και αρχικοποιήστε την χρησιμοποιώντας την pi σταθερά σας.

Κεφ.3 ΤΕΛΕΣΤΕΣ

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο θα αναλύσουμε πως λειτουργούν οι τελεστές έτσι ώστε να εκτελούμε ενέργειες. Τι είναι αληθές και πως ενεργείτε σε αυτές τις περιπτώσεις και ποία είναι η προτεραιότητα των τελεστών.

Λέξεις κλειδιά

Τελεστές εκχώρησης, σχεσιακοί, μαθηματικοί, λογικοί, αύξησης και μείωσης
Πρόθεμα και επίθεμα
Τελεστής sizeof
Τελεστής κόμμα(,)
Τελεστής bit

Ορισμός τελεστών

Ένας τελεστής είναι ένα σύμβολο που αναγκάζει το μεταγλωττιστή να κάνει μια ενέργεια. Οι ενέργειες μπορεί να είναι αύξηση και μείωση, να αναφέρονται στη σχέση μεταξύ δύο αντικειμένων, να αναφέρονται στους τρόπους συνδυασμών αληθών και ψευδών προτάσεων και γενικότερα στη σχέση μεταξύ δύο τελεστών.

Οι τελεστές ενεργούν στους τελεστέους και στη C++, οποιαδήποτε έκφραση μπορεί να είναι ένας τελεστέος.

Θεμελιώδες τύποι

Στη C++ υπάρχουν διάφορες κατηγορίες τελεστών. Οι κατηγορίες είναι :

- **τελεστές εκχώρησης (=)**, αυτός ο τελεστής αναγκάζει τον τελεστέο στην αριστερή πλευρά του τελεστή εκχώρησης να πάρει την τιμή που είναι στη δεξιά πλευρά του τελεστή εκχώρησης. Η έκφραση :

  x = a + b;

  αποδίδει την τιμή που είναι το αποτέλεσμα της πρόσθεσης του a και του b στον τελεστέο x.

Τελεστής εκχώρησης	**Δείγμα Παράστασης**	**Εξήγηση**	**Εκχωρεί το**
Υποθέστε: int c = 3, d = 5, e = 4, f = 6, g = 12;			
+=	**c += 7**	**c = c + 7**	**10 στο c**
-=	**d -= 4**	**d = d - 4**	**1 στο d**
***=**	**e *= 5**	**e = e * 5**	**20 στο e**
/=	**f /= 3**	**f = f / 3**	**2 στο f**
%=	**g %= 9**	**g = g % 9**	**3 στο g**

- **σχεσιακοί τελεστές**, αναφέρονται στη σχέση μεταξύ δύο αντικειμένων

  == : ισότητα

  != : ανισότητα

  > : μεγαλύτερο από

  < : μικρότερο από

  >= : μεγαλύτερο από ή ίσο με

  <= : μικρότερο από ή ίσο με

- **μαθηματικοί τελεστές**,

  + : Πρόσθεση

  - : Αφαίρεση

  ***** : Πολλαπλασιασμός

  / : Διαίρεση

  % : Υπόλοιπο ακέραιας διαίρεσης

- **Λογικοί Τελεστές**, αναφέρονται στους τρόπους συνδυασμών αληθών και ψευδών προτάσεων.

  && είναι το λογικό AND : ΚΑΙ

  || είναι το λογικό OR : Ή

  ! είναι το λογικό NOT : ΟΧΙ

Αν με τον υπολογισμό της πρώτης παράστασης ο συνδυασμός βγαίνει αληθής ή ψευδής τότε αποφεύγεται ο υπολογισμός της δεύτερης.

Παράδειγμα:

(alpha == true) && (beta == 1)

το πρόγραμμα θα επιστρέψει αμέσως ψευδή τιμή αν η πρώτη είναι ψευδής, και θα προχωρήσει στον υπολογισμό της δεύτερης μόνο αν η πρώτη είναι αληθής.

- **Τελεστές Αύξησης και Μείωσης,** η πιο συνηθισμένη τιμή που προστίθεται ή αφαιρείται και εκχωρείτε πάλι σε μια μεταβλητή είναι το 1. Στη C++, η πρόσθεση του 1 ονομάζεται αύξηση και η αφαίρεση του 1 μείωση. Ο τελεστής αύξησης (++) αυξάνει την τιμή της μεταβλητής κατά 1 και ο τελεστής (--) την μειώνει κατά 1. Τόσο ο τελεστής αύξησης (++) όσο και ο τελεστής μείωσης (--) έρχονται σε δύο ποικιλίες: το **πρόθεμα** και το **επίθεμα**.Το πρόθεμα γράφετε πριν από το όνομα της μεταβλητής (++ myAge) και το επίθεμα γράφετε μετά (myAge++). Ο τελεστής προθέματος υπολογίζετε πριν από την εκχώρηση. Ο τελεστής επιθέματος υπολογίζετε

μετά την εκχώρηση. Η σημασία του προθέματος είναι αυτή: αύξησε την τιμή και μετά προσκόμισε την ή χρησιμοποίησε την. Η σημασία του επιθέματος είναι διαφορετική: προσκόμισε ή χρησιμοποίησε την τιμή και έπειτα αύξησε την αρχική μεταβλητή.

Δείτε τον παρακάτω πίνακα:

Τελεστής	**Ονομάζεται**	**Δείγμα παράστασης**	**Εξήγηση**
++	προ-αύξηση	++a	Αυξάνει το a κατά 1 και μετά χρησιμοποιεί τη νέα τιμή του a στην παράσταση στην οποία βρίσκεται το a.
++	μετά-αύξηση	a++	Χρησιμοποιεί την τρέχουσα τιμή του a στην παράσταση στην οποία βρίσκεται το a και αυξάνει το a κατά 1.
--	προ-μείωση	--b	Μειώνει το b κατά 1 και μετά χρησιμοποιεί τη νέα τιμή του b στην παράσταση στην οποία βρίσκεται το b.
--	μετά-μείωση	b--	Χρησιμοποιεί την τρέχουσα τιμή του b στην παράσταση στην οποία βρίσκεται το a και μειώνει το b κατά 1.

Παράδειγμα:

```
1. # include <iostream>
2. int main()
3. {
4.     using std::cout;
5.     int myAge = 39;
6.     int yourAge = 39;
7.     cout << "i am: "<< myAge << " years old.\n";
```

```
8.  cout << "you are: "<< yourAge << " years old.\n";
9.  myAge++;
10. ++yourAge;
11. cout << "one year passes...\n";
12. cout << "i am: "<< myAge << " years old.\n";
13. cout << "you are: "<< yourAge << " years old.\n";
14. cout << "another year passes...\n";
15. cout << "i am: "<< myAge++ << " years old.\n";
16. cout << "you are: "<< ++yourAge << " years old.\n";
17. cout << "lets print it again.\n";
18. cout << "i am: "<< myAge << " years old.\n";
19. cout << "you are: "<< yourAge << " years old.\n";
20. char response;
21. std::cin >> response;
22. return 0;
23. }
```

Έξοδος :

i am: 39 years old.

you are: 39 years old.

one year passes...

i am: 40 years old.

you are: 40 years old.

another year passes...

i am: 40 years old.

you are: 41 years old.

lets print it again.

i am: 41 years old.

you are: 41 years old.

Ανάλυση:

Στις γραμμές 5 και 6, δηλώνονται δύο ακέραιες μεταβλητές και κάθε μια αρχικοποιείται με την τιμή 39. Στη γραμμή 9, το myAge αυξάνεται χρησιμοποιώντας τον τελεστή αύξησης επιθέματος και στη γραμμή 10, το yourAge αυξάνεται χρησιμοποιώντας τον τελεστή αύξησης προθέματος. Στη γραμμή 15, το myAge αυξάνεται στη πρόταση εκτύπωσης χρησιμοποιώντας τον τελεστή αύξησης επιθέματος. Επειδή είναι τελεστής επιθέματος, η αύξηση συμβαίνει μετά την εκτύπωση και έτσι τυπώνεται πάλι η τιμή 40 και μετά αυξάνεται πάλι το myAge. Αντίθετα στη γραμμή 16, το yourAge αυξάνεται χρησιμοποιώντας τον τελεστή αύξησης προθέματος. Κατά συνέπεια, αυξάνεται πριν τυπωθεί και η τιμή που εμφανίζεται είναι 41.

Άλλοι τελεστές

- **Τελεστής sizeof**

Ο τελεστής sizeof δέχεται ως όρισμα μια ποσότητα ή ένα τύπο και επιστρέφει το μέγεθός τους σε bytes.

Στο παρακάτω παράδειγμα δίνονται οι τρόποι κλήσης του τελεστή sizeof:

```
Int a ;
Std:: cout << sizeof (int);
Std:: cout << sizeof (a);
Std:: cout << sizeof a;
```

Ο τελεστής ακολουθείται από το όνομα του τύπου ή της ποσότητας σε παρενθέσεις. Οι παρενθέσεις μπορεί να παραλείπονται αν το όρισμα είναι το όνομα της ποσότητας. Ο τελεστής sizeof υπολογίζεται κατά τη μεταγλώττιση του και το αποτέλεσμα του θεωρείται σταθερή ποσότητα, μπορεί επομένως να χρησιμοποιείται όπου χρειάζεται τέτοια.

§ Τελεστής κόμμα(,)

Δύο ή περισσότερες εκφράσεις μπορούν να διαχωρίζονται από τον τελεστή κόμμα. Ο υπολογισμός του γίνεται από αριστερά προς τα δεξιά και η τιμή της συνολικής έκφρασης είναι η τιμή της δεξιότερης.

§ Τελεστής bit

Υπάρχουν τελεστές που αντιμετωπίζουν τα ορίσματά τους ως σύνολο bits σε σειρά, δηλαδή ως ακολουθίες από 0 ή 1. Η δράση τους ελέγχει ή θέτει την τιμή του κάθε bit χωριστά. Τα ορίσματά τους είναι μεταβλητές με τύπο ακεραίου (short int, int, long int, bool, char) και enum, signed ή unsigned.

Οι τελεστές παρουσιάζονται στον παρακάτω πίνακα.

Τελεστής	**Όνομα**
~	Bitwise NOT
<<	Μετατόπιση αριστερά
>>	Μετατόπιση δεξιά
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
<<=	Μετατόπιση αριστερά με ανάθεση
>>=	Μετατόπιση δεξιά με ανάθεση
&=	Bitwise AND με ανάθεση
^=	Bitwise XOR με ανάθεση
|=	Bitwise OR με ανάθεση

Ο **τελεστής ~** δρώντας σε ένα ακέραιο, επιστρέφει ένα νέο ακέραιο έχοντας μετατρέψει τα 0 του αρχικού σε 1 και αντίστροφα.

Ο **τελεστής <<, >>** μετατοπίζουν τα bits του αριστερού τους ορίσματος κατά τόσες θέσεις όσες ορίζει το δεξί τους όρισμα. Τα επιπλέον bits χάνονται. Ο τελεστής << γεμίζει τις κενές θέσεις με 0 ενώ ο >> κάνει το ίδιο αν το αριστερό όρισμα είναι unsigned.

Οι **τελεστές &,** ^ και | επιστρέφουν ακέραιο με bit pattern που προκύπτει αν εκτελεστεί το AND, XOR και OR αντίστοιχα στα ζεύγη bit των ορισμάτων τους.

Η αποθήκευση bit σε ακέραιους και η χρήση τελεστών για το χειρισμό του είναι σημαντικό όταν θέλουμε να καταγράψουμε την κατάσταση ενός πλήθος αντικειμένων.

Προτεραιότητα τελεστών

Κάθε τελεστής έχει μια τιμή προτεραιότητας που παρουσιάζεται στην παρακάτω λίστα. Όταν δύο μαθηματικοί τελεστές έχουν την ίδια προτεραιότητα, εκτελούνται από αριστερά προς τα δεξιά. Μερικοί τελεστές, όπως ο τελεστής εκχώρησης, υπολογίζονται από δεξιά προς τα αριστερά. Επίσης τα στοιχεία της παρένθεσης υπολογίζονται με υψηλότερη προτεραιότητα από τους μαθηματικούς τελεστές.

Χαρακτηρισμός	**Όνομα**	**Τελεστής**
1	Επίλυση πεδίου δράσης	**::**
2	Επιλογή μέλους, δείκτης πίνακα, κλήσεις συναρτήσεων, τελεστής αύξησης και μείωσης επιθέματος.	**. ->** **()** ++ --
3	Sizeof, τελεστής αύξησης και μείωσης προθέματος, συμπλήρωμα,and,not,μοναδιαίο μείον και συν,address-of και αποαναφορά(dereferense),new,new[], delete, delete[], μετατροπή, sizeof()	++ -- **^ ! - +** **& *** **( )**
4	Επιλογή μέλους δείκτη	**.* ->***
5	Πολλαπλασιασμός, διαίρεση, υπόλοιπο	*** / %**
6	Πρόσθεση, αφαίρεση	**+ -**
7	Μετατόπιση (αριστερά και δεξιά)	<< >>
8	Σχεσιακός τελεστής ανισότητας	<<== >>==
9	Ισότητα, ανισότητα	== !=

10	AND επιπέδου bit	**&**
11	Αποκλειστικό OR επιπέδου bit	^
12	OR επιπέδου bit	|
13	Λογικό AND	**&&**
14	Λογικό OR	||
15	Συνθήκη	**?:**
16	Τελεστές εκχώρησης	= ***=** /= **%=** += **-=** <<= >>= **&=** |= ^=
17	Κόμμα	**,**

Ασκήσεις αξιολόγησης

Άσκηση 1

Εάν τα myAge, a και b είναι όλα ακέραιες μεταβλητές, ποίες οι τιμές τους μετά την εκτέλεση των προτάσεων;

myAge = 39;

a = myAge++;

b = ++myAge;

Άσκηση 2

Ποία η τιμή του:

1. 5+3*9
2. 9*5-3
3. 7/2*8+3-7

Κεφ.4 ΕΝΤΟΛΕΣ

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο θα μάθουμε να κάνουμε ελέγχους, έτσι ώστε το πρόγραμμα να εκτελεί ανάλογες προτάσεις. Τις χρησιμοποιούμε ώστε τα προγράμματα να μπορούν να εκτελούν διακλαδώσεις αν μια συνθήκη είναι αληθής ή ψευδής.

Λέξεις κλειδιά

Εντολή if

Εντολή ?

Εντολή else

Εντολή if…else

Ορισμός

Οι εντολές είναι τελεστές της C++ με τους οποίους μπορούμε να κάνουμε διάφορους ελέγχους, όπως αν το αποτέλεσμα είναι αληθές ή ψευδές, αν θέλουμε να ακολουθήσουμε μια διακλάδωση εάν μια συνθήκη είναι αληθής και άλλη διακλάδωση αν μια συνθήκη είναι ψευδής. Γενικότερα χρησιμοποιούμε τις εντολές για να κάνουμε ελέγχους ώστε το πρόγραμμά μας να εκτελεί ανάλογες προτάσεις.

Η εντολή if

Η εντολή if κάνει έναν λογικό έλεγχο χρησιμοποιώντας έναν συσχετιστικό τελεστή της C++. Ανάλογα με το αποτέλεσμα του ελέγχου, αληθές ή ψευδές, το πρόγραμμα αλλάζει κατεύθυνση και εκτελεί ανάλογες προτάσεις. Η πρόταση if σας επιτρέπει να εξετάζετε μια συνθήκη και μετά να διακλαδώνεται σε διαφορετικά μέρη του κώδικά σας ανάλογα με το αποτέλεσμα.

Η απλούστερη μορφή της if είναι η παρακάτω:

If (έκφραση)

Πρόταση;

Η έκφραση στις παρενθέσεις μπορεί να είναι οποιαδήποτε έκφραση, αλλά περιέχει συνήθως μια από τις σχεσιακές εκφράσεις. Εάν η έκφραση έχει την τιμή false η πρόταση αγνοείται. Εάν έχει την πρόταση true, η πρόταση εκτελείται.

Παράδειγμα:

If (bigNumber > smallNumber)

BigNumber = smallNumber;

Αυτός ο κώδικας συγκρίνει το bigNumber και το smallNumber. Εάν το bigNumber είναι μεγαλύτερο, η δεύτερη γραμμή ορίζει την τιμή του, στην τιμή του smallNumber. Εάν το bigNumber δεν είναι μεγαλύτερο από το smallNumber, αυτή η πρόταση αγνοείται.

Δείτε το παρακάτω πρόγραμμα :

```
1. # include <iostream>
2. int main()
3. {
4.     using std::cout;
5.     using std::cin;
6.     int MetsScore, YankeesScore;
7.     cout << "enter the score for the Mets: ";
8.     cin >> MetsScore;
9.     cout << "\nenter the score for the Yankees: ";
10.    cin >> YankeesScore;
11.    cout << "\n";
12.
13.    if (MetsScore > YankeesScore)
```

```
14.    cout << "lets go Mets\n";
15.
16.    if (MetsScore < YankeesScore)
17.    {
18.        cout << "lets go Yankees\n";
19.    }
20.    if (MetsScore = YankeesScore)
21.    {
22.        cout << "the real score for the Yankees: ";
23.        cin >> YankeesScore;
24.
25.        if (MetsScore > YankeesScore)
26.        cout << "knew it!Mets";
27.        if (MetsScore < YankeesScore)
28.        cout << "knew it!Yankees";
29.        if (MetsScore = YankeesScore)
30.        cout << "it really was a tie";
31.    }
32.    cout << "\nthanks\n";
33. char response;
34. std::cin >> response;
35. return 0;
36. }
```

‘Εξοδος:

enter the score for the Mets: 2

enter the score for the Yankees: 2

lets go Yankees

the real score for the Yankees: 3

knew it!Yankees it really was a tie

thanks

Ανάλυση:

Αυτό το πρόγραμμα ζητά από το χρήστη να εισάγει τα αποτελέσματα για δύο ομάδες. Οι μεταβλητές συγκρίνονται με την πρόταση if στις γραμμές 13, 16 και 20. Εάν το αποτέλεσμα είναι υψηλότερο από το άλλο, τυπώνεται ένα ενημερωτικό μήνυμα. Εάν τα αποτελέσματα είναι ίσα, αρχίζει το μπλοκ του κώδικα που ξεκινά στη γραμμή 21 και τελειώνει στη γραμμή 31. Εάν το αρχικό αποτέλεσμα των Yankees ήταν υψηλότερο από το αποτέλεσμα των Mets, η πρόταση if στη γραμμή 13 δίνει false και δεν καλείται η γραμμή 14. Η δοκιμή στη γραμμή 16 δίνει true και καλείται η πρόταση στη γραμμή 18. Κατόπιν εξετάζεται η πρόταση if στη γραμμή 20 και αυτή δίνει false. Αυτό το πρόγραμμα δείχνει ότι παίρνοντας ένα αποτέλεσμα true σε μια πρόταση if δε σταματά ο έλεγχος άλλων προτάσεων if. Στο πρώτο παράδειγμα για την εντολή if δεν βάλαμε άγκιστρα, διότι ένα απλό μπλοκ κώδικα δεν τα χρειάζεται.

Η εντολή ?

Ο τελεστής? είναι ένας ιδιαίτερα διαδεδομένος ιδιωματισμός της C++. Η εντολή:

If (condition)
Val = value1;
Else
Val = value2;

Ισοδυναμεί με : **Val = (condition? έκφρασηA: έκφρασηB)**

Γενικότερα η έκφραση : **(condition? έκφρασηA: έκφρασηB)**

Έχει την τιμή «έκφρασηA» όταν η συνθήκη condition είναι αληθής, ενώ έχει την τιμή «έκφρασηB» όταν η συνθήκη είναι ψευδής. Οι παρενθέσεις που περιβάλουν τον τελεστή με τα ορίσματα του στο συγκεκριμένο παράδειγμα δεν είναι απαραίτητες, βοηθούν όμως στην κατανόηση του κώδικα. Καθώς για το συγκεκριμένο τελεστή δεν μπορεί να καθοριστεί μονοσήμαντα η προτεραιότητα του σε σχέση με τον (=) πρέπει να τον χρησιμοποιούμε για να εκτελείται η επιδιωκόμενη πράξη. Εναλλακτικά, μπορούμε να εφαρμόζουμε τον εξής κανόνα: οι τελεστές (?) και (=) έχουν ίδια προτεραιότητα, δεχόμενοι ότι οι πράξεις εκτελούνται από τα δεξιά προς τα αριστερά.

Η εντολή else

Συχνά τα προγράμματα σας θα θέλετε να ακολουθούν μια διακλάδωση εάν μια συνθήκη είναι αληθής και άλλη διακλάδωση εάν είναι ψευδής. Η πρόταση else μπορεί να κάνει τον κώδικα πιο ευανάγνωστο:

If (έκφραση)

Πρόταση1;

Else

Πρόταση2;

Παράδειγμα :

```
1. # include <iostream>
2. int main()
3. {
4.     using std::cout;
5.     using std::cin;
6.     int firstNumber, secondNumber;
7.     cout << "please enter a big number: ";
8.     cin >> firstNumber;
9.     cout << "\nlease enter a smaller number: ";
10.    cin >> secondNumber;
11.    if (firstNumber > secondNumber)
12.    cout << "\nThanks!\n";
13.    else
14.    cout << "\nthe first number is not bigger";
15.
16. char response;
17. std::cin >> response;
18. return 0;
19. }
```

Έξοδος:

please enter a big number: 3

lease enter a smaller number: 4

the first number is not bigger

Ή

please enter a big number: 4

lease enter a smaller number: 3

Thanks!

Ανάλυση :

Εκτελείτε η εντολή if στη γραμμή 11. Εάν η συνθήκη είναι αληθής, εκτελείτε η πρόταση στη γραμμή 12 και το πρόγραμμα πηγαίνει στη γραμμή 15.Εαν η συνθήκη στη γραμμή 11 είναι ψευδής, ο έλεγχος πηγαίνει στην εντολή else και εκτελείτε η πρόταση στη γραμμή 14. Κάθε μια ή και οι δύο προτάσεις if και else θα μπορούσαν να αντικατασταθούν με ένα μπλοκ κώδικα μέσα σε άγκιστρα. Στην έξοδο παρατηρούμε ότι μας βγάζει διαφορετική έξοδο ανάλογα με τον αριθμό που βάζουμε για το big number και το smaller number.

Γενικά η σύνταξη της if είναι η εξής :

Α’ τρόπος

If (έκφραση)

Πρόταση;

Επόμενη_πρόταση;

Εάν η έκφραση είναι αληθής, εκτελείτε η πρόταση και το πρόγραμμα συνεχίζει με την επόμενη_πρόταση.

Β’ τρόπος

If (έκφραση)

Πρόταση1;

Else

Πρόταση2;

Επόμενη_πρόταση;

Εάν η πρόταση δώσει αληθής, εκτελείτε η πρόταση1, διαφορετικά εκτελείται η πρόταση2. Μετά το πρόγραμμα συνεχίζει με την επόμενη πρόταση.

Προχωρημένες προτάσεις if

Μπορεί να χρησιμοποιηθεί οποιαδήποτε πρόταση μέσα σε μια if ή else, ακόμα και μια άλλη πρόταση if ή else. Συνεπώς μπορεί να δείτε σύνθετες if προτάσεις με τη παρακάτω μορφή:

```
1.  If (έκφραση1)
2.      {
3.       If (έκφραση2)
4.          Πρόταση1;
5.      Else
6.        {
7.        If (έκφραση3)
8.        Πρόταση2;
9.      Else
10.      Πρόταση3;
11.       }
12.   }
13.  Else
14. Πρόταση3;
```

Αυτή η πολύπλοκη if λέει, «αν η έκφραση1 είναι αληθής και η έκφραση2 είναι αληθής, εκτέλεσε την πρόταση1. Εάν η έκφραση1 είναι αληθής αλλά η έκφραση2 δεν είναι αληθής, τότε αν η έκφραση3 είναι αληθής εκτέλεσε την πρόταση2. Αν η έκφραση1 είναι

αληθής αλλά η έκφραση2 και η έκφραση3 είναι ψευδείς, τότε εκτέλεσε την πρόταση3. Τέλος αν η έκφραση1 δεν είναι αληθής, εκτέλεσε την πρόταση4». Δείτε το παρακάτω παράδειγμα:

```
1. # include <iostream>
2. int main()
3. {
4.     using namespace std;
5.     int firstNumber, secondNumber;
6.     cout << "enter two numbers.\nFirst: ";
7.     cin >> firstNumber;
8.     cout << "\nSecond: ";
9.     cin >>secondNumber;
10.    cout <<"\n\n";
11.    if (firstNumber >= secondNumber)
12.    {
13.        if((firstNumber % secondNumber) ==0)
14.        {
15.            if(firstNumber == secondNumber)
16.               cout << "They are the same!\n";
17.            else
18.               cout << "They are evenly divisible!\n";
19.        }
20.        else
21.           cout << "They are not evenly divisible!\n";
22.    }
23.    else
24.           cout << "the second one is larger!\n";
25. char response;
26. std::cin >> response;
27. return 0;
28. }
```

Έξοδος:

enter two numbers.

First: 2

Second: 1

They are evenly divisible!

Ανάλυση :

Ζητούνται δύο αριθμοί ένας κάθε φορά και έπειτα συγκρίνονται. Η πρώτη έκφραση if στη γραμμή 11, ελέγχει αν ο πρώτος αριθμός είναι μεγαλύτερος ή ίσος με τον δέυτερο. Αν όχι εκτελείτε ο όρος else στη γραμμή 23. Εάν η πρώτη if είναι αληθής, εκτελείτε το πρώτο μπλοκ του κώδικα που αρχίζει στη γραμμή 12 και εξετάζεται μια δεύτερη πρόταση if στη γραμμή 13. Αυτό το μπλοκ ελέγχει εάν ο πρώτος αριθμός διαιρούμενος με τον δεύτερο αριθμό δεν αφήνει υπόλοιπο. Σε αυτή την περίπτωση, οι αριθμοί διαιρούνται ακριβώς μεταξύ τους ή είναι ίσοι. Η πρόταση if στη γραμμή 15 ελέγχει για ισότητα και εμφανίζει το κατάλληλο μήνυμα και στις δύο περιπτώσεις. Εάν η πρόταση if στη γραμμή 13 αποτύχει, εκτελείται η πρόταση else στη γραμμή 20.

Χρήση αγκίστρων σε ένθετες προτάσεις if

Αν και μπορείτε να παραλείπετε τα άγκιστρα σε προτάσεις if που αποτελούνται από μια μόνο πρόταση και επιτρέπεται να βάζετε ένθετες προτάσεις if, αυτό μπορεί να προκαλέσει σύγχυση. Το παρακάτω παράδειγμα δείχνει ένα επιτρεπτό κώδικα:

```
1. if (x > y)
2. if (x < z)
3. x = y;
4. else
5. x = z;
6. else
7. y = x;
```

Να θυμάστε ότι τα κενά και οι εσοχές είναι βολικές για τους προγραμματιστές και δεν έχουν καμία επίδραση στον μεταγλωττιστή.

Δείτε το παρακάτω παράδειγμα:

```
1.  #include <iostream>
2.    int main()
3.    {
4.      using namespace std;
5.      int x;
6.      cout << "Enter a number less than 10 or greater than 100: ";
7.      cin >> x;
8.      cout << "\n";
9.
10.     if (x > 10)
11.       if (x > 100)
12.           cout << "More than 100, Thanks!\n";
13.     else
14.       cout << "Less than 10, Thanks!\n";
15.
16.   char response;
17.   std::cin >> response;
18.   return 0;
19.   }
```

Έξοδος:

Enter a number less than 10 or greater than 100: 50
Less than 10, Thanks!

Ανάλυση:

Εάν η πρόταση if στη γραμμή 10 δίνει true, εκτελείται η επόμενη πρόταση. Σε αυτή την περίπτωση εκτελείται η γραμμή 11 όταν ο αριθμός που εισάγεται είναι μεγαλύτερος από 10. Η γραμμή 11 περιέχει επίσης μια πρόταση if. Αυτή η πρόταση if δίνει true εάν ο αριθμός που δίνεται είναι μεγαλύτερος από 100.

Εάν ο αριθμός που δίνεται είναι μεγαλύτερος από 100, εκτελείται η πρόταση στη γραμμή 12 τυπώνοντας το κατάλληλο μήνυμα. Εάν ο αριθμός που θα δοθεί θα είναι μικρότερος από 10, τότε η πρόταση if στη γραμμή 10 δίνει false. Ο έλεγχος

του προγράμματος πηγαίνει στην επόμενη γραμμή μετά από την πρόταση if, σε αυτή την περίπτωση στη γραμμή 15. Εάν δώσετε έναν αριθμό μικρότερο από το 10 η έξοδος θα είναι:

Enter a number less than 10 or greater than 100: 8

Όπως βλέπετε, δεν τυπώνετε κάποιο μήνυμα. Η πρόταση else στη γραμμή 13 συνδέετε με την πρόταση if της γραμμής 10 και έτσι έχει μπει ανάλογα σε εσοχή. Η πρόταση else είναι συνδεδεμένη με την πρόταση if στη γραμμή 11 και έχει πρόβλημα. Η παρακάτω λίστα διορθώνει το πρόβλημα βάζοντας τα κατάλληλα άγκιστρα.

```
1.  #include <iostream>
2.      int main()
3.      {
4.        using namespace std;
5.       int x;
6.       cout << "Enter a number less than 10 or greater than 100: ";
7.       cin >> x;
8.       cout << "\n";
9.
10.     if (x > 10)
11.     {
12.       if (x > 100)
13.            cout << "More than 100, Thanks!\n";
14.     }
15.     else
16.       cout << "Less than 10, Thanks!\n";
17.    char response;
18.    std::cin >> response;
19.    return 0;
20.   }
```

Έξοδος:

Enter a number less than 10 or greater than 100: 2

Less than 10, Thanks!

Ανάλυση :

Τα άγκιστρα στις γραμμές 11 και 14 κάνουν τα πάντα μεταξύ τους για να ανήκουν σε μια πρόταση και τώρα το else στη γραμμή 15 εφαρμόζεται στη γραμμή 10.

Εάν ο χρήστης πληκτρολογήσει έναν αριθμό μικρότερο από το 10, η πρόταση if στη γραμμή 10 είναι αληθής. Ωστόσο η πρόταση if στη γραμμή 12 είναι ψευδής, έτσι δεν τυπώνεται τίποτα. Θα ήταν καλύτερο αν ο προγραμματιστής έβαζε μια άλλη πρόταση else μετά τη γραμμή 13, έτσι ώστε να πιάνονται αυτά τα λάθη και να τυπώνεται ένα μήνυμα.

Ασκήσεις αξιολόγησης

Άσκηση1

Γράψτε μια πρόταση if που να εξετάζει δύο ακέραιες μεταβλητές και να αλλάζει τη μεγαλύτερη με την μικρότερη, χρησιμοποιώντας μόνο έναν όρο else.

Άσκηση2

Εξετάστε το πρόγραμμα και ελέγξτε την έξοδο:

```
1. # include <iostream>
2. using namepace std;
3. int main()
4. {
5. int a=2, b=2, c;
6. if (c= (a-b))
7. cout << “the value of c is: ” <<c;
8. char response;
9. std::cin >> response;
10. return 0;
11. }
```

Κεφ.5 ΔΕΙΚΤΕΣ

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο θα αναλύσουμε τι είναι οι δείκτες, πως λειτουργούν βήμα προς βήμα, πως να τους δηλώνεται και πως χρησιμοποιούνται και τι είναι ελεύθερος χώρος καθώς επίσης και πως να χειρίζεστε τη μνήμη.

Λέξεις κλειδιά

Τελεστής διεύθυνσης (&)
Δείκτης μπαλαντέρ
Έμμεση αναφορά (*)
Κλειδί new
Διαρροή μνήμης
Δείκτης this

Ορισμός

Μια συνάρτηση πρέπει να χρησιμοποιήσει δείκτες για να μπορέσει να αλλάξει την τιμή μιας παραμέτρου. Οι μεταβλητές που ορίζονται σε ένα πρόγραμμα, αποθηκεύονται σε κατάλληλες θέσεις μνήμης. Ο αριθμός της θέσης στην οποία βρίσκεται μια μεταβλητή εξάγεται με τη δράση του τελεστή διεύθυνσης στη

μεταβλητή. Ο **τελεστή διεύθυνσης (&)** αποτρέπει στο πρόγραμμα να προσδιορίσει τη διεύθυνση στη μνήμη μιας μεταβλητής, ενώ ο **τελεστής (*)** βοηθάει το πρόγραμμα να προσδιορίσει τη μεταβλητή που είναι αποθηκευμένη στη συγκεκριμένη διεύθυνση.

Ένας δείκτης είναι μια μεταβλητή που κρατά μια διεύθυνση μνήμης.

Η δήλωση του δείκτη γίνεται με τη μορφή:

Τύπος * όνομα_δείκτη

Γενικά τους δείκτες τους χρησιμοποιούμε, πιο συχνά, για τρεις εργασίες:

- διαχείριση των δεδομένων της ελεύθερης αποθήκευσης
- πρόσβαση σε δεδομένα μέλη και συναρτήσεις μέλη κλάσεων

πέρασμα μεταβλητών με αναφορά σε συναρτήσεις.

Αποθήκευση της διεύθυνσης μιας μεταβλητής

Κάθε μεταβλητή έχει μια διεύθυνση. Ακόμα και χωρίς να ξέρετε τη συγκεκριμένη διεύθυνση μιας μεταβλητής, μπορείτε να αποδώσετε τη διεύθυνση μιας μεταβλητής σε ένα δείκτη. Για να δηλώσετε ένα δείκτη με όνομα pAge που να κρατά τη διεύθυνση μιας μεταβλητής γράφετε:

Int * pAge = 0;

Αυτό δηλώνει ότι η pAge είναι ένας δείκτης σε ένα Int, δηλαδή η pAge δηλώνεται για να περιέχει τη διεύθυνση ενός ακέραιου αριθμού.

Ονόματα δεικτών

Για ονόματα των δεικτών μπορείτε να χρησιμοποιήσετε οποιοδήποτε όνομα θα χρησιμοποιούσατε στις άλλες μεταβλητές. Πολλοί προγραμματιστές χρησιμοποιούν της σύμβαση της ονομασίας όλων των δεικτών με ένα αρχικό p, όπως η pAge.

Στο παραπάνω παράδειγμα:

Int * pAge = 0;

Η pAge αρχικοποιείται στο 0. Ένας δείκτης του οποίου η τιμή είναι μηδέν καλείται κενός δείκτης. Όλοι οι δείκτες, όταν δημιουργούνται, θα πρέπει να αρχικοποιούνται σε κάτι. Ένας δείκτης που δεν είναι αρχικοποιημένος ονομάζεται **δείκτης μπαλαντέρ**.

Για να περιέχει ένας δείκτης μια διεύθυνση πρέπει να αποδοθεί στον δείκτη.

Δείτε το παρακάτω:

```
unsigned short int howOld = 50;     //φτιάχνει μια μεταβλητή
unsigned short int * pAge = 0;      //φτιάχνει ένα δείκτη
pAge = &howOld;                     //φτιάχνει τη διεύθυνση του howOld στο page.
```

Ξέρετε ότι το pAge είναι ένας δείκτης λόγω του αστερίσκου**(*)** μετά από τον τύπο της μεταβλητής και πριν από το όνομα της μεταβλητής.

Η πρόσβαση στην τιμή που είναι αποθηκευμένη σε μια μεταβλητή χρησιμοποιώντας ένα δείκτη ονομάζεται **έμμεση αναφορά**, επειδή έχετε έμμεσα πρόσβαση στη μεταβλητή με τη βοήθεια του δείκτη. Η έμμεση πρόσβαση σημαίνει ότι γίνεται πρόσβαση στην τιμή που είναι στη διεύθυνση που περιέχει ένας δείκτης.

Αποαναφορά με τον τελεστή έμμεσης αναφοράς

Ο τελεστής **έμμεσης αναφοράς (*)** ονομάζεται επίσης τελεστής αποαναφοράς. Όταν ένας δείκτης αποαναφέρεται, ανακτάται η τιμή στη διεύθυνση που έχει αποθηκευτεί στο δείκτη.

Οι κανονικές μεταβλητές παρέχουν άμεση πρόσβαση στις δικές τους τιμές. Εάν δημιουργήσετε μια νέα μεταβλητή τύπου unsigned short int με όνομα yourAge και θέλετε να αποδώσετε την τιμή της howOld σε αυτή τη νέα μεταβλητή, πρέπει να γράψετε:

```
unsigned short int yourAge;
yourAge = howOld;
```

Ένας δείκτης παρέχει έμμεση πρόσβαση στην τιμή της μεταβλητής της οποίας την διεύθυνση περιέχει. Για να εκχωρήσετε την τιμή της howOld στη νέα μεταβλητή yourAge μέσω του δείκτη pAge πρέπει να γράψετε:

```
unsigned short int yourAge;
yourAge = *pAge;
```

Ο τελεστής έμμεσης αναφοράς μπροστά από τη μεταβλητή δείκτη pAge σημαίνει «η τιμή που είναι αποθηκευμένη».

Χειρισμός δεδομένων με τη χρήση των δεικτών

Έκτος από τη χρήση του τελεστή έμμεσης αναφοράς για να δείτε τι δεδομένα είναι αποθηκευμένα σε μια θέση στην οποία δείχνει μια μεταβλητή, μπορείτε επίσης να χειριστείτε αυτά τα δεδομένα.

Στην παρακάτω λίστα βλέπετε πως εκχωρείτε η διεύθυνση μιας τοπικής μεταβλητής σε ένα δείκτη και πως μπορεί να χρησιμοποιηθεί ο τελεστής έμμεσης αναφοράς για να χειριστεί τις τιμές που είναι σε αυτή τη μεταβλητή.

```
1.  #include <iostream>
2.
3.      typedef unsigned short int USHORT;
4.      int main()
5.      {
6.         using namespace std;
7.         USHORT myAge;
8.         USHORT * pAge = 0;
9.        myAge = 5;
10.       cout << "myAge: " << myAge << "\n";
11.
12.       pAge = &myAge;
13.
14.       cout << "*pAge: " << *pAge << "\n\n";
15.
16.       cout << "*pAge = 7\n";
17.
18.       *pAge = 7;
19.
20.       cout << "*pAge: " << *pAge << "\n";
21.       cout << "myAge: " << myAge << "\n\n";
22.
23.       cout << "myAge = 9\n";
24.
25.      myAge = 9;
26.
```

```
27.       cout << "myAge: " << myAge << "\n";
28.       cout << "*pAge: " << *pAge << "\n";
29.    char response;
30.    std::cin >> response;
31.    return 0;
32. }
```

Έξοδος :

myAge: 5

**pAge: 5*

**pAge = 7*

**pAge: 7*

myAge: 7

myAge = 9

myAge: 9

**pAge: 9*

Ανάλυση :

Αυτό το πρόγραμμα δηλώνει δύο μεταβλητές: μια unsigned short, και ένα δείκτη σε μια unsigned short.

Στη γραμμή 12, η pAge παίρνει τη διεύθυνση της myAge. Στη γραμμή 14, η pAge αποαναφέρεται, χρησιμοποιώντας τον τελεστή έμμεσης αναφοράς και τυπώνεται δείχνοντας ότι η τιμή στη διεύθυνση που περιέχει η pAge είναι η τιμή 5 που είναι αποθηκευμένη στο myAge.

Στη γραμμή 18, εκχωρείτε η τιμή 7 στη μεταβλητή που είναι στη διεύθυνση που είναι αποθηκευμένο στην pAge.Θα πρέπει να παρατηρήσετε ότι η έμμεση πρόσβαση στη μεταβλητή έγινε χρησιμοποιώντας ένα αστερίσκο, που εδώ είναι ο τελεστής έμμεσης αναφοράς.

Δέσμευση χώρου με τη λέξη κλειδί new

Δεσμεύεται μνήμη στον ελεύθερο χώρο χρησιμοποιώντας τη λέξη κλειδί new. Η new ακολουθείται από τον τύπο του αντικειμένου που θέλετε να δεσμεύσετε, έτσι ώστε ο μεταγλωττιστής να ξέρει πόση μνήμη απαιτείται. Η τιμή επιστροφής από τη new είναι μια δέσμευση μνήμης. Για να γράψετε ένα unsigned short στον ελεύθερο χώρο, θα πρέπει να γράψετε:

unsigned short int * pPointer;

pPointer = new unsigned short int;

Μπορείτε επίσης να το κάνετε αυτό σε μια γραμμή αρχικοποιώντας τον δείκτη την ίδια στιγμή που τον δηλώνεται:

unsigned short int * pPointer = new unsigned short int;

Αποδέσμευση μνήμης με τη λέξη κλειδί delete

Όταν τελειώσετε με μια περιοχή της μνήμης, πρέπει να την αποδώσετε πάλι στο σύστημα. Η delete επιστρέφει τη μνήμη στον ελεύθερο χώρο. Η μνήμη που δεσμεύεται από την new δεν ελευθερώνεται αυτόματα. Αν μια μεταβλητή δείκτη δείχνει σε μνήμη στον ελεύθερο χώρο και ο δείκτης φύγει από το πεδίο δράσης, η μνήμη δεν επιστρέφει αυτόματα στον ελεύθερο χώρο. Αντίθετα θεωρείται δεσμευμένη και επειδή ο δείκτης δεν είναι πλέον διαθέσιμος, δεν μπορείτε να έχετε πρόσβαση σε αυτή τη μνήμη.

Αυτή η κατάσταση ονομάζεται **διαρροή μνήμης**, επειδή αυτή η μνήμη δεν μπορεί να ανακτηθεί έως ότου τελειώσει το πρόγραμμα. Για να αποτρέψετε τις διαρροές μνήμης, θα πρέπει να επαναφέρεται οποιαδήποτε μνήμη δεσμεύσατε ξανά στον ελεύθερο χώρο. Αυτό το κάνετε χρησιμοποιώντας τη λέξη κλειδί delete.

Κάθε φορά που το πρόγραμμά σας καλεί την new, θα πρέπει να υπάρχει και μια αντίστοιχη κλήση στη delete. Είναι σημαντικό να παρακολουθείτε ποίος δείκτης κατέχει κάθε περιοχή της μνήμης και να εξασφαλίζετε ότι η μνήμη επιστρέφει στον ελεύθερο χώρο όταν τελειώσετε μαζί του.

Όταν καλείται τη **delete** για ένα δείκτη που δείχνει σε ένα αντικείμενο στον ελεύθερο χώρο, καλείται η συνάρτηση αποδιάρθρωσης αυτού του αντικειμένου πριν ελευθερωθεί η μνήμη. Αυτό δίνει στην κλάση σας τη δυνατότητα να κάνει τον καθαρισμό ακριβώς όπως γίνεται με τα αντικείμενα που καταστρέφονται.

Πρόσβαση σε δεδομένα μέλη

Έχετε πρόσβαση σε δεδομένα μέλη και συναρτήσεις χρησιμοποιώντας τον τελεστή τελεία(.). Η πρόσβαση στα μέλη ενός αντικειμένου μέσω ενός δείκτη είναι πιο περίπλοκη. Για να έχετε πρόσβαση στο αντικείμενο Cat στον ελεύθερο χώρο πρέπει πρώτα να κάνετε αποαναφορά του δείκτη. Μετά χρησιμοποιείτε την τιμή αποαναφοράς, δηλαδή την τιμή στην οποία έδειχνε ο δείκτης, μαζί με τον τελεστή τελεία για να έχετε πρόσβαση στο αντικείμενο. Επομένως για να έχετε πρόσβαση στη συνάρτηση μέλος GetAge ενός αντικειμένου στο οποίο δείχνει ο pRags, θα γράφατε:

(*pRags).GetAge();

Η C++ παρέχει ένα τελεστή για έμμεση πρόσβαση: τον **τελεστή πρόσβασης σε μέλος κλάσης (->)**.

Εκτός από τη δημιουργία αντικειμένων στον ελεύθερο χώρο, μπορείτε να δημιουργήσετε και δεδομένα μέλη μέσα σε ένα αντικείμενο στον ελεύθερο χώρο. Ένα ή περισσότερα από τα δεδομένα μέλη μιας κλάσης μπορούν να είναι ένας δείκτης σε ένα αντικείμενο στον ελεύθερο χώρο.

Ο δείκτης this

Κάθε συνάρτηση μέλος κλάσης έχει μια κρυφή παράμετρο: τον **δείκτη this**. Ο δείκτης this δείχνει σε «αυτό» το μεμονωμένο αντικείμενο. Επομένως σε κάθε κλήση στην GetAge και την SetAge, κάθε συνάρτηση λαμβάνει το this για το αντικείμενο της ως κρυφή παράμετρο. Το this είναι ένας δείκτης που περιέχει τη διεύθυνση μνήμης ενός αντικειμένου.

Είναι δυνατόν να χρησιμοποιηθεί ο δείκτης στον this άμεσα, όπως επεξηγεί η παρακάτω λίστα:

```
1.  #include <iostream>
2.
3.      class Rectangle
4.      {
5.      public:
6.          Rectangle();
7.          ~Rectangle();
8.          void SetLength(int length) { this->itsLength = length; }
```

```
9.          int GetLength() const { return this->itsLength; }
10.
11.         void SetWidth(int width) { itsWidth = width; }
12.         int GetWidth() const { return itsWidth; }
13.
14.     private:
15.         int itsLength;
16.         int itsWidth;
17.     };
18.
19.     Rectangle::Rectangle()
20.     {
21.        itsWidth = 5;
22.        itsLength = 10;
23.     }
24.     Rectangle::~Rectangle()
25.     {
26.     }
27.     int main()
28.     {
29.         using namespace std;
30.         Rectangle theRect;
31.         cout << "theRect is " << theRect.GetLength() << " feet long.\n";
32.         cout << "theRect is " << theRect.GetWidth() << " feet wide.\n";
33.         theRect.SetLength(20);
34.         theRect.SetWidth(10);
35.         cout << "theRect is " << theRect.GetLength()<< " feet long.\n";
36.         cout << "theRect is " << theRect.GetWidth()<< " feet wide.\n";
37.     char response;
38.     std::cin >> response;
39.     return 0;
40. }
```

Έξοδος :

theRect is 10 feet long.

theRect is 5 feet wide.

theRect is 20 feet long.

theRect is 10 feet wide.

Ανάλυση :

Η συνάρτηση ιδιωτικής πρόσβασης SetLength() στη γραμμή 8 και η συνάρτηση ιδιωτικής πρόσβασης GetLength() στη γραμμή 9 χρησιμοποιούν και οι δύο τον δείκτη this για πρόσβαση σε μεταβλητές μέλη του αντικειμένου Rectangle. Οι συναρτήσεις ιδιωτικής πρόσβασης SetWidth() και GetWidth() στις γραμμές 11-12 δεν το χρησιμοποιούν. Δεν υπάρχει καμία διαφορά στη συμπεριφορά τους, αν και η σύνταξη είναι ευκολότερο να κατανοηθεί.

Χρήση σταθερών δεικτών

Μπορείτε να χρησιμοποιήσετε τη λέξη κλειδί **const** σε δείκτες πριν από τον τύπο τους, μετά από τον τύπο τους, ή και στα δύο μέρη. Τα παρακάτω παραδείγματα είναι έγκυρες δηλώσεις:

```
const int * pOne;
int * const pTwo;
const int * const pThree;
```

Κάθε ένας από αυτούς, ωστόσο, κάνει κάτι διαφορετικό:

- Ο **pOne** είναι ένα δείκτης σε ένα σταθερό ακέραιο. Η τιμή στην οποία δείχνει δεν μπορεί να αλλάξει.
- Ο **pTwo** είναι ένα σταθερός δείκτης σε ένα ακέραιο. Ο ακέραιος μπορεί να αλλάξει, αλλά ο **pTwo** δεν μπορεί να δείξει σε κάτι άλλο.
- Ο **pThree** είναι ένα σταθερός δείκτης σε ένα σταθερό ακέραιο. Η τιμή στην οποία δείχνει δεν μπορεί να αλλάξει και ο **pThree** δεν μπορεί να αλλάξει και να δείξει σε κάτι άλλο.

Το τέχνασμα για να καταλάβετε αυτό είναι να κοιτάξετε δεξιά από τη λέξη κλειδί const για να ανακαλύψετε τι είναι αυτό που δηλώνεται ότι είναι σταθερό. Εάν είναι ο τύπος δεξιά της λέξης κλειδί const, τότε η τιμή του είναι σταθερή. Εάν η μεταβλητή είναι δεξιά της λέξης κλειδί const, τότε είναι η ίδια μεταβλητή δείκτης που είναι σταθερή.

.

Ασκήσεις αξιολόγησης

Άσκηση1

Τι κάνουν αυτές οι προτάσεις;

1. int * pOne;
2. int vTwo;
3. int * pThree = &vTwo;

Άσκηση2

Γράψτε ένα μικρό πρόγραμμα που δηλώνει ένα ακέραιο και ένα δείκτη σε ακέραιο. Αποδώστε τη διεύθυνση του ακέραιου στο δείκτη. Χρησιμοποιήστε τον δείκτη για να ορίσετε μια τιμή στην ακέραιη μεταβλητή.

Άσκηση3

Τι είναι λάθος σε αυτόν τον κώδικα;

```
1. # include <iostream>
2. using namespace std;
3. int main()
4. {
5. int  *pint;
6. *pint = 9;
7. cout << “the value at pInt: ” <<*pint
8. char response;
9. std::cin >> response;
10. return 0;
11. }
```

Κεφ.6 ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο θα κάνουμε μια ανάλυση για το τι είναι κλάσεις, πως να ορίζετε μια νέα κλάση και πως να δημιουργείται αντικείμενα αυτής της κλάσης. Τι είναι συναρτήσεις μέλη και τι δεδομένα μέλη καθώς επίσης και τι είναι συναρτήσεις δημιουργίας και πως να τις χρησιμοποιείτε.

Λέξεις κλειδιά

Κλάση

its

public και private

Συνάρτηση

const

inline

Ορισμός

Ο αντικειμενοστραφής προγραμματισμός είναι μια μεθοδολογία προγραμματισμού. Δεν έχει να κάνει με κάποια συγκεκριμένη γλώσσα. Απλά οι αντικειμενοστραφής γλώσσες (Java, C++, Python καθώς και η Perl- όσον αφορά την υποστήριξη) παρέχουν υποστήριξη αυτής της μεθοδολογίας σε επίπεδο μεταγλωττιστή. Σε μη αντικειμενοστραφείς γλώσσες αυτή η δουλειά πέφτει στον προγραμματιστή.

Η αντικειμενοστραφής ανάπτυξη λογισμικού προσφέρει ένα εργαλείο για να σας βοηθήσει να αντιμετωπίσετε τις προκλήσεις της ανάπτυξης λογισμικού. Αν και δεν υπάρχουν έτοιμες λύσεις σχετικά με την περίπλοκη ανάπτυξη λογισμικό, οι αντικειμενοστραφείς γλώσσες προγραμματισμού δημιουργούν μια ισχυρή σύνδεση μεταξύ των δομών δεδομένων και των μεθόδων που χειρίζονται αυτά τα δεδομένα και είναι πιο κοντά στον τρόπο που σκέπτονται οι άνθρωποι, βελτιώνοντας την επικοινωνία και την ποιότητα λογισμικού. Στον αντικειμενοστραφή προγραμματισμό, δε σκέφτεστε πλέον τις δομές δεδομένων και τις συναρτήσεις χειρισμού τους, αλλά σκέφτεστε τα αντικείμενα, σα να ήταν πραγματικά αντικείμενα.

Η C++ στόχος της είναι να παρέχει μια αντικειμενοστραφή σχεδίαση σε μια γρήγορη, εμπορική πλατφόρμα ανάπτυξης λογισμικού, με ειδική εστίαση στην υψηλή απόδοση.

Κλάση: Είναι ένα σύνολο από μεταβλητές - συνήθως διαφορετικών τύπων - που συνδυάζονται με ένα σύνολο σχετικών συναρτήσεων. Μια κλάση σας επιτρέπει να ενσωματώνεται τα διάφορα μέρη και τις διάφορες συναρτήσεις σε μια συλλογή, η οποία ονομάζεται αντικείμενο. Μια κλάση μπορεί να αποτελείται από οποιδήποτε συνδυασμό τύπων μεταβλητών και επίσης από άλλους τύπους κλάσεων. Οι μεταβλητές μιας κλάσης αναφέρονται ως μεταβλητές ή ως δομημένα μέλη. Μια κλάση μπορεί να περιέχει συναρτήσεις που ονομάζονται συναρτήσεις μέλη ή μέθοδοι.

Δήλωση κλάσης: Η δήλωση μιας κλάσης μιλά στο μεταγλωττιστή για την κλάση. Για να δηλώσετε μια κλάση χρησιμοποιείται τη λέξη κλειδί class, ακολουθούμενη από το όνομα της κλάσης, ένα αριστερό άγκιστρο και έπειτα μια λίστα από τα δεδομένα μέλη και τις μεθόδους της κλάσης. Τελειώνετε τη δήλωση με ένα δεξιό άγκιστρο και ένα ελληνικό ερωτηματικό.

Παράδειγμα:

```
class cat
{
unsigned int itsAge;
unsigned int itsWeight;
void Meow();
};
```

“its” : βοηθά να διακρίνονται οι μεταβλητές μέλη από τις μη μεταβλητές μέλη.

Για να έχεις πρόσβαση στα μέλη του αντικειμένου χρησιμοποιείται τον τελεστή τελείας (.) π.χ.

Frisky.itsWeight = 50;(Για να εκχωρήσω 50 στη μεταβλητή μέλους Weight)

Η λέξη κλειδί class

Η σύνταξη για τη λέξη κλειδί class είναι η ακόλουθη :

```
class class_name
{
};
```

Η τάξη (class) είναι ένα πρωταρχικό εργαλείο του αντικειμενοστραφούς προγραμματισμού (object-oriented programming) και μοιάζει πολύ με μια δομή στο ότι ομαδοποιεί μέλη διαφόρων τύπων. Στον αντικειμενοστραφή προγραμματισμό ασχολούμαστε με πληροφορίες, δεδομένα, μέλη και πεδία που αποτελούν ένα σύστημα καθώς με τις πράξεις (συναρτήσεις, μέθοδοι) που μπορούμε να κάνουμε με αυτά.

Σε μια τάξη μπορούμε να αποθηκεύσουμε δεδομένα καθώς και συναρτήσεις (μεθόδους, methods) που εκτελούν διάφορες πράξεις πάνω στα δεδομένα.

Μια τάξη στη C++ ορίζεται ως εξής :

1. **class Cat**
2. **{**
3. **public:**
4. **unsigned int Age;**
5. **unsigned int Weight;**
6. **void Meow();**

```
7. };
8. Cat  Frisky;
9. Frisky.Age = 8;
10. Frisky.Weight = 18;
11. Frisky.Meow();
```

Ιδιωτική ως προς δημόσια πρόσβαση

Άλλες λέξεις κλειδιά στη δήλωση μιας κλάσης είναι **public** και **private** και χρησιμοποιούνται με τα μέλη μιας κλάσης τόσο σε δεδομένα μέλη, όσο και με μεθόδους μέλη. Τα ιδιωτικά μέλη μπορούν να προσπεραστούν μόνο μέσα από μεθόδους της ίδιας κλάσης. Τα δημόσια μέλη προσπελάζονται μέσω οποιοδήποτε αντικειμένου της κλάσης.

Παράδειγμα:

```
1. Class Cat
2. {
3. public:
4. unsigned int itsAge;
5. unsigned int itsWeight;
6. void Meow();
7. };
```

Ο τρόπος που χρησιμοποιείται το Cat ώστε να μπορείτε να έχετε πρόσβαση στα δεδομένα μέλη είναι να κάνετε δημόσια κάποια μέλη.

Η παρακάτω λίστα δείχνει τη δήλωση μιας κλάσης Cat με δημόσιες μεταβλητές μέλη.

```
1. # include <iostream>
2.
3. class Cat
4. {
5.       public:
6.             int itsAge;
7.             int itsWeight;
8. };
```

```
9.  int main()
10. {
11.     Cat Frisky;
12.     Frisky.itsAge=5;
13.     std::cout << "Frisky is a cat who is " ;
14.     std::cout << Frisky.itsAge << " years old.\n";
15.     char response;
16. std::cin >> response;
17. return 0;
18. }
```

Έξοδος :

Frisky is a cat who is 5 years old.

Ανάλυση :

Η γραμμή 3 περιέχει τη λέξη κλειδί class. Αυτό λέει στο μεταγλωττιστή ότι αυτό που ακολουθεί είναι μια δήλωση. Το όνομα της νέας κλάσης πηγαίνει μετά τη λέξη κλειδί class.

Το σώμα της δήλωσης αρχίζει με το αριστερό άγκιστρο στη γραμμή 4 και τελειώνει με ένα δεξιό άγκιστρο και ένα ελληνικό ερωτηματικό στη γραμμή 8. Η γραμμή 5 περιέχει τη λέξη κλειδί public, ακολουθούμενη από μια άνω και κάτω τελεία, που υποδεικνύει ότι όσα ακολουθούν είναι δημόσια μέχρι τη λέξη κλειδί private ή μέχρι το τέλος της δήλωσης της κλάσης.

Η γραμμή 9 αρχίζει με τη συνάρτηση main(). Ο Frisky ορίζεται στη γραμμή 11 ως ένα στιγμιότυπο του Cat δηλαδή ως ένα αντικείμενο Cat. Στις γραμμές 13 και 14 χρησιμοποιείται η μεταβλητή μέλους itsAge για να εκτυπώσει ένα μήνυμα για τον Frisky. Παρατηρήστε στις γραμμές 12 και 14 πως προσπελάζεται το μέλος του αντικειμένου Frisky. Το itsAge προσπελάζεται χρησιμοποιώντας το όνομα του αντικειμένου, ακολουθούμενο από μια τελεία και μετά το όνομα του μέλους.

Καθιστώντας Ιδιωτικά τα μέλη δεδομένων

Για να έχετε πρόσβαση σε ιδιωτικά μέλη μιας κλάσης θα πρέπει να δημιουργήσετε δημόσιες συναρτήσεις, που ονομάζονται μέθοδοι ιδιωτικής πρόσβασης (accesor). Αυτές οι μέθοδοι ιδιωτικής πρόσβασης είναι οι συναρτήσεις μέλη, που άλλα μέρη του προγράμματος σας καλούν για να πάρουν και να ορίσουν τις ιδιωτικές μεταβλητές μέλη. Μια δημόσια μέθοδος ιδιωτικής πρόσβασης είναι μια συνάρτηση μέλος μιας κλάσης, που χρησιμοποιείται είτε για να διαβάσει την τιμή μιας ιδιωτικής μεταβλητής μέλους κλάσης, είτε για να ορίσει την τιμή της. Οι συναρτήσεις ιδιωτικής πρόσβασης σας επιτρέπουν να διαχωρίσετε τις λεπτομέρειες για το πως αποθηκεύονται τα δεδομένα από το πως χρησιμοποιούνται. Χρησιμοποιώντας συναρτήσεις ιδιωτικής πρόσβασης , μπορείτε αργότερα να αλλάξετε τον τρόπο με τον οποίο αποθηκεύονται τα δεδομένα, χωρίς να πρέπει να ξαναγραφτούν οι συναρτήσεις που χρησιμοποιούν τα δεδομένα.

Χειρισμός μεθόδων κλάσεων

Ορισμός της συνάρτησης: αρχίζει παρόμοια με τον ορισμό μιας κανονικής συνάρτησης. Πρώτα ορίζετε τον τύπο επιστροφής που θα επιστραφεί από τη συνάρτηση ή το void αν δεν επιστρέφει τίποτα. Αυτό ακολουθείται από το όνομα της κλάσης δύο άνω κάτω τελείες, το όνομα της συνάρτησης και μετά τις παραμέτρους του.

Η παρακάτω λίστα παρουσιάζει την πλήρη δήλωση μιας απλής κλάσης Cat και η υλοποίηση της συνάρτησης ιδιωτικής πρόσβασης και μια γενική συνάρτηση μέλους.

```
1.  # include <iostream>
2.
3.  class Cat
4.  {
5.      public:
6.          int GetAge();
7.          void SetAge (int age);
8.          void Meow();
9.      private:
10.         int itsAge;
11. };
```

```
12. int Cat::GetAge()
13. {
14.     return itsAge;
15. }
16. void Cat::SetAge(int age)
17. {
18.      itsAge = age;
19. }
20. void Cat::Meow()
21. {
22.      std::cout <<"Meow.\n";
23. }
24.
25. int main()
26. {
27.     Cat Frisky;
28.     Frisky.SetAge(5);
29.     Frisky.Meow();
30.     std::cout<< "Frisky is a cat who is ";
31.     std::cout<< Frisky.GetAge() << " years old.\n";
32.     Frisky.Meow();
33.     char response;
34. std::cin >> response;
35. return 0;
36. }
```

Έξοδος :

Meow.

Frisky is a cat who is 5 years old.

Meow.

Ανάλυση :

Οι γραμμές 5 – 11 περιέχουν τον ορισμό της κλάσης Cat.Η γραμμή 5 περιέχει τη λέξη κλειδί public. Η GetAge() παρέχει πρόσβαση στην ιδιωτική μεταβλητή μέλος itsAge, που δηλώνεται στη γραμμή 10. Η SetAge παίρνει έναν ακέραιο αριθμό ως όρισμα και δίνει στο itsAge την τιμή αυτού του ορίσματος. Η Meow() δεν είναι μια μέθοδος ιδιωτικής πρόσβασης. Εδώ είναι μια γενική μέθοδος που τυπώνει τη λέξη «Meow» στην οθόνη.

Στη γραμμή 9 αρχίζει το ιδιωτικό τμήμα. Οι γραμμές 12 – 15 περιέχουν τον ορισμό της συνάρτησης μέλους GetAge(). Αυτή η μέθοδος δεν παίρνει καμία παράμετρο και επιστρέφει έναν ακέραιο. Στη γραμμή 12 οι μέθοδοι της κλάσης περιλαμβάνουν το όνομα της κλάσης, ακολουθούμενο από δύο άνω και κάτω τελείες και το όνομα της συνάρτησης. Αυτή η σύνταξη λέει στο μεταγλωττιστή ότι η συνάρτηση GetAge() είναι αυτή που δηλώσατε στην κλάση Cat.

Η συνάρτηση GetAge() παίρνει μόνο μια γραμμή και επιστρέφει την τιμή του itsAge.Η συνάρτηση main() δεν μπορεί να έχει πρόσβαση στο itsAge, επειδή το itsAge είναι ιδιωτικό στην κλάση Cat.

Η γραμμή 16 περιέχει τον ορισμό της συνάρτησης μέλους SetAge(). Μπορείτε να δείτε ότι αυτή η συνάρτηση παίρνει μια ακέραιη παράμετρο, που ονομάζεται age και δεν επιστρέφει τιμές, όπως υποδεικνύεται από τη void.

Η γραμμή 20 είναι μια συνάρτηση μιας γραμμής που τυπώνει τη λέξη «Meow» στην οθόνη, ακολουθούμενη από μια αλλαγή γραμμής. Η «Meow» ορίζεται όπως και οι συναρτήσεις ιδιωτικής πρόσβασης γιατί ξεκινά με τον τύπο επιστροφής, το όνομα της κλάσης, το όνομα της συνάρτησης και τις παραμέτρους.

Η γραμμή 25 αρχίζει το σώμα του προγράμματος. Στη γραμμή 27 δηλώνεται ένα αντικείμενο που ονομάζεται Frisky, τύπου Cat. Στη γραμμή 28 η τιμή 5 εκχωρείτε στη μεταβλητή μέλος itsAge μέσω της μεθόδου πρόσβασης SetAge(). Παρατηρήστε ότι η μέθοδος καλείται χρησιμοποιώντας το όνομα του αντικειμένου Frisky, ακολουθούμενο από τον τελεστή μέλους (.) και το όνομα της μεθόδου SetAge().

Συμπερίληψη συναρτήσεων μελών const

Έχετε χρησιμοποιήσει τη λέξη κλειδί const για να δηλώσετε μεταβλητές που δεν αλλάζουν ή με συναρτήσεις μέλη μιας κλάσης. Αν δηλώσετε μια μέθοδο κλάσης const, θεωρείτε ότι η μέθοδος δεν θα αλλάξει τιμή σε κανένα από τα μέλη της κλάσης.

Για να δηλώσετε **σταθερή** μια μέθοδο κλάσης, βάλτε τη λέξη κλειδί const μετά από τις παρενθέσεις των παραμέτρων, αλλά πριν από το ελληνικό ερωτηματικό που τερματίζει τη δήλωση της μεθόδου.

Για παράδειγμα :

void SomeFunction() const;

Αυτό δηλώνει μια σταθερή μέθοδο κλάσης που ονομάζεται SomeFuction() και δεν παίρνει ορίσματα και επιστρέφει το void. Γνωρίζετε ότι αυτή η μέθοδος δεν θα αλλάξει κάποια από τα δεδομένα μέλη της ίδιας κλάσης, επειδή έχει δηλωθεί ως const.

Οι μέθοδοι ιδιωτικής πρόσβασης που λαμβάνουν μόνο τιμές, συνήθως δηλώνονται ως σταθερές συναρτήσεις χρησιμοποιώντας το τροποποιητικό const. Νωρίτερα είδατε ότι η κλάση Cat έχει δύο μεθόδους ιδιωτικής πρόσβασης :

void SetAge(int anAge);

int GetAge();

Η **SetAge()** δεν μπορεί να είναι const επειδή αλλάζει η μεταβλητή μέλος itsAge. Η **GetAge(),** μπορεί και πρέπει να είναι const επειδή δεν αλλάζει καθόλου την κλάση. Η GetAge() επιστρέφει απλώς την τρέχουσα τιμή της μεταβλητής μέλους itsAge.Επομένως η δήλωση αυτών των συναρτήσεων θα πρέπει να γραφτεί ως εξής:

void SetAge(int anAge);

int GetAge() const;

Εάν δηλώσετε μια συνάρτηση ως const και η υλοποίηση αυτής της συνάρτησης αλλάξει το αντικείμενο αλλάζοντας την τιμή ενός από τα μέλη του, ο μεταγλωττιστής το σημειώνει ως λάθος.

Ενσωματωμένη υλοποίηση

Όπως μπορείτε να ζητήσετε από το μεταγλωττιστή σας να ενσωματώσει (**inline**) μια κανονική συνάρτηση, έτσι μπορείτε να ενσωματώσετε και τις μεθόδους μιας κλάσης. Η λέξη κλειδί inline εμφανίζεται πριν από τον τύπο επιστροφής. Η inline υλοποίηση της συνάρτησης GetWeight παραδείγματος χάριν, μοιάζει ως εξής:

```
1. inline int Cat::GetWeight()
2. {
3. return itsWeight;
4. }
```

Μπορείτε επίσης να βάλετε τον ορισμό μιας συνάρτησης στη δήλωση της κλάσης, πράγμα που κάνει αυτόματα αυτή τη συνάρτηση inline.
Παραδείγματος χάριν:

```
1. class Cat
2. {
3. public:
4. int GetWeight() { return itsWeight; }    // inline
5. void SetWeight(int aWeight);
6. };
```

Το σώμα της inline συνάρτησης αρχίζει αμέσως μετά τη δήλωση της μεθόδου της κλάσης. Δε χρησιμοποιούμε ελληνικό ερωτηματικό μετά τις παρενθέσεις. Ο ορισμός αρχίζει με ένα αριστερό άγκιστρο και τελειώνει με ένα δεξιό άγκιστρο. Τα κενά δεν έχουν σημασία. Θα μπορούσατε επίσης να την γράψετε ως εξής:

```
1. class Cat
2. {
3. public:
4. int GetWeight()
5. {
6. return itsWeight;
7. }                    // inline
8. void SetWeight(int aWeight);
9. };
```

Ασκήσεις αξιολόγησης

Άσκηση1

Γράψτε τον κώδικα που δηλώνει μια κλάση που ονομάζεται Employee, με αυτά τα δεδομένα μέλη: itsAge, itsYearOfService και itsSalary.

Άσκηση2

Γράψτε ένα πρόγραμμα με τη κλάση Employee που να δημιουργεί δύο υπαλλήλους. Ορίστε το itsAge, itsYearOfService και itsSalary και τυπώστε τις τιμές τους. Θα πρέπει επίσης να προσθέσετε τον κώδικα για τις μεθόδους ιδιωτικής πρόσβασης.

Άσκηση3

Γιατί δεν είναι πολύ χρήσιμη η παρακάτω δήλωση κλάσης;

```
1. Class Cat
2. {
3. int GetAge() const;
4. private:
5. int itsAge;
6. };
```

Κεφ.7 ΣΥΝΑΡΤΗΣΕΙΣ

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο θα ασχοληθούμε με το τι είναι συνάρτηση και ποια είναι τα μέρη, πως να δηλώνεται και να ορίζετε συναρτήσεις, πως να περνάτε παραμέτρους σε συναρτήσεις και πως να επιστρέφετε μια τιμή από μια συνάρτηση.

Λέξεις κλειδιά

Καθολικές συναρτήσεις
Κλήση συνάρτησης
Λίστα παραμέτρων
Ορίσματα
Τοπικές μεταβλητές και καθολικές μεταβλητές
Προκαθορισμένη τιμή
Υπερφόρτωση συναρτήσεων, Ενσωματωμένες Συναρτήσεις.

Ορισμός

Συνάρτηση είναι ένα υποπρόγραμμα που μπορεί να ενεργεί σε δεδομένα και επιστρέφει μια τιμή. **Καθολικές** όταν δεν αποτελούν μέρος ενός αντικειμένου δηλαδή μπορούν να προσπελαστούν από οπουδήποτε στο πρόγραμμα. Κάθε συνάρτηση έχει το

δικό της όνομα και όταν συναντάτε αυτό το όνομα στο πρόγραμμα η εκτέλεση διακλαδώνεται στο σώμα αυτής της συνάρτησης. Αυτό λέγεται **κλήση της συνάρτησης**.

Όταν στέλνετε τιμές σε μια συνάρτηση, αυτές οι τιμές λειτουργούν ως μεταβλητές που μπορείτε να χειριστείτε μέσα από τη συνάρτηση. Η περιγραφή των τιμών που στέλνετε ονομάζεται **λίστα παραμέτρων**. **Ορίσματα** είναι οι πραγματικές τιμές που περνάτε στη συνάρτηση.

Δήλωση και ορισμός συναρτήσεων

Η χρήση συναρτήσεων στο πρόγραμμα απαιτεί να δηλώνετε πρώτα η συνάρτηση και έπειτα να ορίζετε τη συνάρτηση. Η δήλωση λέει στο μεταγλωττιστή το όνομα, τον τύπο επιστροφής και τις παραμέτρους της συνάρτησης. Ο ορισμός λέει στο μεταγλωττιστή πως λειτουργεί η συνάρτηση.
Η δήλωση μιας συνάρτησης λέγεται πρωτότυπο (μια έκφραση που τελειώνει με ;)

Υπάρχουν τρεις τρόποι για να δηλώσετε μια συνάρτηση:

- Για να γράψετε το πρωτότυπό σας σε ένα αρχείο και χρησιμοποιήστε έπειτα την οδηγία #include για να συμπεριλάβετε στο πρόγραμμά σας.
- Γράψτε το πρωτότυπο στο αρχείο στο οποίο χρησιμοποιείται η συνάρτησή σας.
- Ορίστε τη συνάρτηση πριν κληθεί από οποιαδήποτε άλλη συνάρτηση. Όταν το κάνετε αυτό, ο ορισμός ενεργεί ως το πρωτότυπό της.

Αν και μπορείτε να ορίσετε τη συνάρτηση πριν τη χρησιμοποιήσετε και να αποφύγετε έτσι την ανάγκη για ένα πρωτότυπο συνάρτησης, αυτή δεν είναι ορθή πρακτική προγραμματισμού για τρεις λόγους :

1. Είναι κακή ιδέα να εμφανίζονται οι συναρτήσεις σε ένα αρχείο σε μια συγκεκριμένη σειρά. Αυτό θα δυσκολέψει τη συνάρτηση του προγράμματος όταν θα αλλάζουν οι απαιτήσεις.
2. Είναι δυνατό η συνάρτηση A() να μπορεί να καλεί τη συνάρτηση B(), αλλά η συνάρτηση B() να μπορεί επίσης να καλεί τη συνάρτηση A() κάτω από κάποιες περιστάσεις. Δεν είναι δυνατό να οριστεί η συνάρτηση A() πριν ορίσετε τη συνάρτηση B() και επίσης να ορίσετε

τη συνάρτηση B() πριν ορίσετε τη συνάρτηση A(), έτσι τουλάχιστον η μια πρέπει να δηλωθεί σε οποιαδήποτε περίπτωση.

3. Τα πρωτότυπα των συναρτήσεων είναι καλή και δυνατή τεχνική αποσφαλμάτωσης. Εάν το πρωτότυπό σας δηλώνει ότι η συνάρτηση σας παίρνει ένα συγκεκριμένο σύνολο παραμέτρων ή ότι επιστρέφει ένα συγκεκριμένο τύπο τιμής και έπειτα η συνάρτησή σας δεν ταιριάζει με το πρωτότυπο, ο μεταγλωττιστής μπορεί να σημειώσει το λάθος σας αντί να περιμένετε να παρουσιαστεί όταν θα τρέξετε το πρόγραμμα. Το πρωτότυπο και ο ορισμός ελέγχουν το ένα το άλλο μειώνοντας την πιθανότητα ότι ένα απλό τυπογραφικό λάθος θα καταλήξει στη δημιουργία ενός σφάλματος στο πρόγραμμά σας.

Σημειώστε ότι όλες οι συναρτήσεις έχουν ένα τύπο επιστροφής. Εάν δε δηλωθεί άμεσα ένας τύπος, ο τύπος επιστροφής είναι εξ'ορισμού int. Τα προγράμματα σας θα είναι πιο ευκολονόητα, ωστόσο εάν δηλώσετε άμεσα τον τύπο επιστροφής της κάθε συνάρτησης συμπεριλαμβανομένης της main().

Εάν η συνάρτηση δεν επιστρέφει μια τιμή δηλώνεται τον τύπο επιστροφής της ως void. Όταν χρησιμοποιείται την void, τίποτα δεν επιστρέφεται.

Οι μεταβλητές που δηλώνεται μέσα στο σώμα της συνάρτησης ονομάζονται «τοπικές», επειδή υπάρχουν μόνο τοπικά μέσα στην ίδια τη συνάρτηση.

Πως ορίζουμε τη συνάρτηση

Ο ορισμός μια συνάρτησης αποτελείται από την επικεφαλίδα της συνάρτησης και το σώμα της. Η επικεφαλίδα είναι όπως το πρωτότυπο μιας συνάρτησης εκτός ότι οι παράμετροι πρέπει να έχουν ονόματα και δεν χρειάζεται τερματικό ελληνικό ερωτηματικό. Το σώμα της συνάρτησης είναι ένα σύνολο προτάσεων που περιέχονται σε άγκιστρα. Δείτε το παρακάτω παράδειγμα:

```
1. #include <iostream>
2. int Area(int length, int width);
3.
4.    int main()
5.    {
```

```
6.          using std::cout;
7.          using std::cin;
8.
9.        int lengthOfYard;
10.       int widthOfYard;
11.       int areaOfYard;
12.
13.     cout << "\nHow wide is your yard? ";
14.     cin >> widthOfYard;
15.     cout << "\nHow long is your yard? ";
16.     cin >> lengthOfYard;
17.
18.     areaOfYard= Area(lengthOfYard,widthOfYard);
19.
20.     cout << "\nYour yard is ";
21.     cout << areaOfYard;
22.     cout << " square feet\n\n";
23.               char response;
24. std::cin >> response;
25. return 0;
26.   }
27.
28.  int Area(int len, int wid)
29.   {
30.       return len * wid;
31.       char response;
32. std::cin >> response;
33. return 0;
34. }
```

Έξοδος :

How wide is your yard? 1

How long is your yard? 2

Your yard is 2 square feet

Ανάλυση :

Το πρωτότυπο της Area είναι στη γραμμή 2. Συγκρίνετε το πρωτότυπο με τον ορισμό της συνάρτησης στη γραμμή 28. Παρατηρήστε ότι το όνομα, ο τύπος επιστροφής και οι τύποι των παραμέτρων είναι ίδιοι. Εάν ήταν διαφορετικοί θα είχε εμφανιστεί ένα λάθος από το μεταγλωττιστή. Η μόνη απαιτούμενη διαφορά είναι ότι το πρωτότυπο της συνάρτησης τελειώνει με ένα ελληνικό ερωτηματικό και δεν έχει σώμα.

Επίσης σημειώστε ότι τα ονόματα των παραμέτρων στο πρωτότυπο είναι length και width, αλλά τα ονόματα των παραμέτρων στο ορισμό είναι len και wid.Τα ονόματα του πρωτότυπου δεν χρησιμοποιούνται. Υπάρχουν εκεί ως πληροφορία για τον προγραμματιστή.

Τοπικές μεταβλητές

Μπορείτε να περάσετε μεταβλητές σε μια συνάρτηση, αλλά μπορείτε επίσης να δηλώσετε μεταβλητές μέσα στο σώμα της συνάρτησης. Οι μεταβλητές που δηλώνονται μέσα στο σώμα της συνάρτησης ονομάζονται «**τοπικές**» επειδή υπάρχουν μόνο τοπικά μέσα στην ίδια τη συνάρτηση. Οι παράμετροι που περνούν μέσα στη συνάρτηση θεωρούνται επίσης τοπικές μεταβλητές.

Παράδειγμα:

```
1.  #include <iostream>
2.
3.      float Convert(float);
4.      int main()
5.      {
6.        using namespace std;
7.
8.        float TempFer;
9.        float TempCel;
10.
11.       cout << "Please enter the temperature in Fahrenheit: ";
12.      cin >> TempFer;
13.      TempCel = Convert(TempFer);
14.      cout << "\nHere's the temperature in Celsius: ";
```

```
15.      cout << TempCel << endl;
16.      char response;
17.      std::cin >> response;
18.      return 0;
19.    }
20.
21.    float Convert(float TempFer)
22.    {
23.      float TempCel;
24.      TempCel = ((TempFer - 32) * 5) / 9;
25.      return TempCel;
26.  }
```

Έξοδος :

Please enter the temperature in Fahrenheit: 32
Here's the temperature in Celsius: 0

Ανάλυση :

Στις γραμμές 8 και 9 δηλώνονται δύο μεταβλητές float.Ζητείται από το χρήστη να εισάγει μια θερμοκρασία Φαρενάιτ στη γραμμή 11 και αυτή η τιμή περνά στη συνάρτηση Convert() στη γραμμή 13. Με την κλίση της Convert(), η εκτέλεση μεταπηδά στη πρώτη γραμμή της συνάρτησης Convert() στη γραμμή 21, όπου δηλώνεται μια τοπική μεταβλητή, που ονομάζεται επίσης TempCel. Αυτή η τοπική μεταβλητή δεν είναι ίδια με τη μεταβλητή TempCel στη γραμμή 9. Αυτή η μεταβλητή υπάρχει μόνο μέσα στη συνάρτηση Convert(). Η τιμή που πέρασε ως παράμετρο η TempFer είναι ένα τοπικό αντίγραφο της μεταβλητής που περνά από τη main().

Η τοπική μεταβλητή συνάρτησης TempCel ορίζεται στη τιμή που προκύπτει από την αφαίρεση του 32 από την παράμετρο TempFer, πολλαπλασιάζοντας με 5 και διαιρώντας έπειτα με 9. Στη γραμμή 13 αυτή η τιμή επιστροφής δίνεται στη μεταβλητή TempCel στη συνάρτηση main().

Οι παράμετροι είναι τοπικές μεταβλητές

Τα ορίσματα που περνούν μέσα στη συνάρτηση είναι τοπικά για τη συνάρτηση. Οι αλλαγές που γίνονται στα ορίσματα δεν επηρεάζουν τις τιμές της καλούσας συνάρτησης. Αυτό είναι γνωστό ως πέρασμα με τιμή, δηλαδή γίνεται ένα τοπικό αντίγραφο κάθε ορίσματος μέσα στη συνάρτηση.

void swap(int x, int y);.....swap(x,y);.....void swap(int x, int y);

temp = x;

x=y;

y=temp;

εναλλάσσονται οι τιμές, δηλαδή το χ παίρνει την τιμή του ψ και το ψ την τιμή του χ.

Καθολικές μεταβλητές

Οι μεταβλητές που ορίζονται έξω από οποιοδήποτε συνάρτηση έχουν καθολικό πεδίο δράσης και έτσι είναι διαθέσιμες σε οποιαδήποτε συνάρτηση του προγράμματος συμπεριλαμβανομένης και της main(). Οι τοπικές μεταβλητές που έχουν το ίδιο όνομα με τις καθολικές μεταβλητές δεν αλλάζουν τις καθολικές μεταβλητές.

Μια τοπική μεταβλητή με το ίδιο όνομα με μια καθολική μεταβλητή κρύβει ωστόσο, την καθολική μεταβλητή. Εάν μια συνάρτηση έχει μια μεταβλητή με το ίδιο όνομα με μια καθολική, το όνομα αναφέρεται στην τοπική και όχι στην καθολική όταν χρησιμοποιείται μέσα στη συνάρτηση.

Η παρακάτω λίστα επεξηγεί αυτά τα σημεία:

```
1. #include <iostream>
2. void myFunction();
3.
4. int x = 5, y = 7;
5. int main()
6. {
7.     using namespace std;
8.     cout << "x from main: " << x << endl;
9.     cout << "y from main: " << y << endl << endl;
10.    myFunction();
11.    cout << "back from myFunction!"<< endl << endl;
```

```
12.     cout << "x from main: " << x << endl;
13.     cout << "y from main: " << y << endl;
14.     char response;
15.     std::cin >> response;
16.     return 0;
17. }
18. void myFunction()
19. {
20.      using std::cout;
21.
22.      int y = 10;
23.
24.     cout << "x from myFunction: " << x << endl;
25.     cout << "y from myFunction: " << y << endl << endl;
26. }
```

Έξοδος :

x from main:5

y from main:7

x from myFunction:5

y from myFunction:10

back from myFunction!

x from main:5

y from main:7

Ανάλυση :

Στη γραμμή 4, δηλώνονται δύο καθολικές μεταβλητές , οι χ και y. Η καθολική τιμή χ αρχικοποιείται με την τιμή 5 και η y με την τιμή 7.

Στις γραμμές 8 και 9 στη συνάρτηση main(), αυτές οι τιμές τυπώνονται στην κονσόλα. Παρατηρήστε ότι η συνάρτηση main() δεν ορίζει καμιά μεταβλητή, αλλά επειδή είναι καθολικές, είναι διαθέσιμές στη συνάρτηση main().

Όταν καλείται η myFunction() στη γραμμή 10, η εκτέλεση του προγράμματος περνά στη γραμμή 18 και στη γραμμή 22 ορίζεται και αρχικοποιείται μια τοπική μεταβλητή, η y με την τομή 10.

Στη γραμμή 24 η myFunction() τυπώνει την τιμή της μεταβλητής χ και η καθολική μεταβλητή χ χρησιμοποιείται ακριβώς όπως ήταν μέσα στη main(). Στη γραμμή 25, ωστόσο, όταν χρησιμοποιείται το όνομα της μεταβλητής y, χρησιμοποιείται η τοπική μεταβλητή y, κρύβοντας την καθολική μεταβλητή με το ίδιο όνομα.

Περισσότερα για τις τιμές επιστροφής

Οι συναρτήσεις επιστρέφουν μια τιμή ή void. Το void είναι ένα σήμα στο μεταγλωττιστή ότι δεν επιστρέφει καμιά τιμή.

Για να επιστρέψει μια τιμή από μια συνάρτηση, γράψτε τη λέξη κλειδί **return** ακολουθούμενη από τη τιμή που θέλετε να επιστρέψετε.

Όταν συναντάτε τη λέξη κλειδί return, επιστρέφει ως τιμή της συνάρτησης η έκφραση που ακολουθεί το return. Η εκτέλεση του προγράμματος επιστρέφει αμέσως στην καλούσα συνάρτηση και οποιασδήποτε προτάσεις υπάρχουν μετά την return δεν εκτελούνται. Η παρακάτω λίστα επεξηγεί τα παραπάνω:

```
1.  #include <iostream>
2.
3.      int Doubler(int AmountToDouble);
4.
5.      int main()
6.      {
7.          using std::cout;
8.          int result = 0;
9.          int input;
10.
11.         cout << "Enter a number between 0 and 10,000 to double: ";
12.         std::cin >> input;
13.
14.         cout << "\nBefore doubler is called... ";
15.         cout << "\ninput: " << input << " doubled: " << result << "\n";
```

```
16.
17.         result = Doubler(input);
18.
19.         cout << "\nBack from Doubler...\n";
20.         cout << "\ninput: " << input << "   doubled: " << result << "\n";
21.         char response;
22.         std::cin >> response;
23.         return 0;
24.     }
25.
26.     int Doubler(int original)
27.     {
28.         if (original <= 10000)
29.             return original * 2;
30.         else
31.             return -1;
32.         std::cout << "You can't get here!\n";
33.  }
```

Έξοδος :

Enter a number between 0 and 10,000 to double: 5000

Before doubler is called...

input: 5000 doubled: 0

Back from Doubler...

input: 5000   doubled: 10000

ή

Enter a number between 0 and 10,000 to double: 20000

Before doubler is called...

input: 20000 doubled: 0

Back from Doubler...

input: 20000   doubled: -1

Ανάλυση :

Ζητείται ένας αριθμός στις γραμμές 11 και 12 και τυπώνονται στις γραμμές 14 και 15, μαζί με την τοπική μεταβλητή return. Η συνάρτηση Doubler() καλείται στη γραμμή 19 και η τιμή εισόδου περνά ως παράμετρος .

Στη γραμμή 28, στη συνάρτηση Doubler(), ελέγχεται η παράμετρος για το αν είναι μικρότερη ή ίση με 10000. Εάν είναι η συνάρτηση επιστρέφει δύο φορές τον αρχικό αριθμό. Εάν η τιμή original είναι μεγαλύτερη από 10000 η συνάρτηση επιστρέφει το –1 ως τιμή λάθους.

Η πρόταση στη γραμμή 32 δεν εκτελείται ποτέ επειδή, ανεξάρτητα από το αν η τιμή είναι μικρότερη ή ίση με 10000 ή μεγαλύτερη από 10000 η συνάρτηση επιστρέφει είτε στη γραμμή 29 είτε στη γραμμή 31, πριν φτάσει στη γραμμή 32. Ένας καλός μεταγλωττιστής προειδοποιεί ότι αυτή η πρόταση δεν μπορεί να εκτελεστεί και ένας καλός προγραμματιστής θα την αφαιρέσει.

Προκαθορισμένες παράμετροι

Μια **προκαθορισμένη τιμή** είναι η τιμή που χρησιμοποιείται εάν δεν παρασχεθεί μια τιμή. Το όνομα της προκαθορισμένης παραμέτρου στο πρωτότυπο δε χρειάζεται να είναι το ίδιο με το όνομα στην επικεφαλίδας της συνάρτησης, γιατί η προκαθορισμένη τιμή εκχωρείται από τη θέση και όχι από το όνομα. Οποιοσδήποτε ή όλες οι παράμετροι μιας συνάρτησης μπορούν να οριστούν σε προκαθορισμένες τιμές. Ο μόνος περιορισμός είναι: εάν οποιαδήποτε από τις παραμέτρους δεν έχει μια προκαθορισμένη τιμή, καμιά προηγούμενη παράμετρος δεν μπορεί να έχει μια προκαθορισμένη τιμή.

Το παρακάτω παράδειγμα δείχνει τη χρήση προκαθορισμένων τιμών:

```
1.  #include <iostream>
2.
3.  int AreaCube(int length, int width = 25, int height = 1);
4.
5.  int main()
6.  {
7.      int length = 100;
8.      int width = 50;
```

```
9.      int height = 2;
10.     int area;
11.
12.     area = AreaCube(length, width, height);
13.     std::cout << "First area equals: " << area << "\n";
14.
15.     area = AreaCube(length, width);
16.     std::cout << "Second time area equals: " << area << "\n";
17.
18.     area = AreaCube(length);
19.     std::cout << "Third time area equals: " << area << "\n";
20.     char response;
21.     std::cin >> response;
22.     return 0;
23.  }
24.
25.  AreaCube(int length, int width, int height)
26.  {
27.
28.     return (length * width * height);
29.  }
```

Έξοδος :

First area equals: 10000

Second time area equals: 5000

Third time area equals: 2500

Ανάλυση :

Στη γραμμή 3 το πρωτότυπο AreaCube καθορίζει ότι η συνάρτηση AreaCube παίρνει τρεις ακέραιες μεταβλητές. Οι τελευταίες δύο έχουν προκαθορισμένες τιμές. Αυτή η συνάρτηση υπολογίζει τον όγκο του κύβου του οπίου περνούν οι διαστάσεις. Στις γραμμές 7 έως 9, αρχικοποιείται το μήκος, το ύψος και το πλάτος και περνούν στη συνάρτηση AreaCube() στη γραμμή 12.

Η εκτέλεση συνεχίζει στη γραμμή 15, όπου καλείται η AreaCube(), αλλά χωρίς τιμή για το ύψος. Χρησιμοποιείται η προκαθορισμένη τιμή και υπολογίζονται πάλι οι διαστάσεις και τυπώνονται.

Η εκτέλεση συνεχίζει έπειτα στη γραμμή 18 και αυτή τη φορά δεν περνά ούτε το πλάτος ούτε το ύψος. Με αυτή την κλήση της AreaCube(), η εκτέλεση διακλαδώνεται για τρίτη φορά στη γραμμή 25. Χρησιμοποιούνται οι προκαθορισμένες τιμές και υπολογίζεται ο όγκος. Ο έλεγχος επιστρέφει στη συνάρτηση main() όπου τυπώνεται η τελική τιμή.

Υπερφόρτωση συναρτήσεων

Υπερφόρτωση συναρτήσεων(overloading function) είναι όταν δημιουργείται περισσότερο από μια συνάρτηση με το ίδιο όνομα , όπου οι τύποι επιστροφής μπορεί να είναι ίδιοι ή διαφορετικοί. Η υπερφόρτωση ή αλλιώς πολυμορφισμός συναρτήσεων (function polymorphism) αναφέρεται στη ικανότητα να «υπερφορτώνεται» μια συνάρτηση με περισσότερες από μια έννοια.

Η υπερφόρτωση συναρτήσεων είναι μια τεχνική προγραμματισμού της C++ για να παρέχουμε παραπάνω από έναν ορισμό για ένα δεδομένο όνομα μιας συνάρτησης. Αυτό το χαρακτηριστικό μας επιτρέπει να αναπτύξουμε μια ομάδα συναρτήσεων οι οποίες επιτελούν παρόμοιες λειτουργίες χρησιμοποιώντας το ίδιο όνομα. Το μονό που μένει είναι να αποφασίσει ο μεταγλωττιστής ποιες από τις συναρτήσεις πρέπει να χρησιμοποιηθούν για να επιτελεστεί το έργο.

Στη C++ μπορούμε να ορίσουμε περισσότερες από μία συναρτήσεις με το ίδιο όνομα και τον ίδιο επιστρεφόμενο τύπο, αλλά με διαφορετικό αριθμό ή τύπο παραμέτρων. Κατά τη μεταγλώττιση, η C++ καλεί την κατάλληλη συνάρτηση ανάλογα με τον αριθμό ή τον τύπο των παραμέτρων που της μεταβιβάζονται.

Ένα παράδειγμα είναι να χρησιμοποιεί υπέρβαση στη συνάρτηση add_ values(), όπου ο πρώτος ορισμός της συνάρτησης προσθέτει δύο τιμές τύπου int, ενώ ο δεύτερος ορισμός της συνάρτησης προσθέτει τρεις τιμές τύπου int. Η C++ είναι σε θέση να διακρίνει ποια είναι η συνάρτηση που πρέπει να χρησιμοποιήσει με βάση τον αριθμό ή τον τύπο των παραμέτρων.

Ενσωματωμένες Συναρτήσεις

Inline: Ο μεταγλωττιστής δε δημιουργεί μια πραγματική συνάρτηση, αλλά αντιγράφει τον κώδικα από την ενσωματωμένη συνάρτηση κατευθείαν στην καλούσα συνάρτηση. Η λέξη κλειδί inline είναι μια νύξη στον μεταγλωττιστή ότι θέλετε η συνάρτηση να είναι ενσωματωμένη. Ο μεταγλωττιστής είναι ελεύθερος να αγνοήσει τη νύξη και να κάνει μια πραγματική κλίση σε συνάρτηση.

Τη λέξη κλειδί inline μπορούμε να την τοποθετούμε μπροστά από το όνομα μιας συνάρτησης και όταν τη συναντήσει ο μεταγλωττιστής της C++, αντικαθιστά κάθε κλήση της συνάρτησης με τις προτάσεις που αυτή περιέχει, σε γλώσσα μηχανής στο εκτελέσιμο πρόγραμμα. Έτσι το πρόγραμμα απαλλάσσεται από την επιβάρυνση που προκαλούν οι κλήσεις των συναρτήσεων. Με αυτόν τον τρόπο, το πρόγραμμα συνεχίζει να είναι ευανάγνωστο επειδή χρησιμοποιεί συναρτήσεις, αλλά έχει και βελτιωμένη απόδοση επειδή έχει απαλλαγή από τις κλήσεις των συναρτήσεων και εκτελείται ταχύτερα. Η inline συνάρτηση είναι σα μια κανονική συνάρτηση εκτός ότι:
α. χρησιμοποιεί τη λέξη-κλειδί «inline» ως πιστοποίηση (qualifier) της συνάρτησης. Δηλαδή πρώτη λέξη στο πρωτότυπο της συνάρτησης είναι το «inline»
β. Αμέσως μετά τον ορισμό της συνάρτησης ακολουθεί το κύριο μέρος της συνάρτησης

Ασκήσεις αξιολόγησης

Άσκηση1

Γράψτε το πρωτότυπο μιας συνάρτησης που ονομάζεται Perimeter(), η οποία επιστρέφει ένα unsigned long int και παίρνει δύο παραμέτρους και τις δύο unsigned short int.

Άσκηση2

Γράψτε τον ορισμό της συνάρτησης Perimeter() όπως περιγράφεται στην άσκηση1.Οι δύο παράμετροι αντιπροσωπεύουν το μήκος και το πλάτος ενός ορθογωνίου. Κάνετε τη συνάρτηση να επιστρέφει την περίμετρο(δύο φορές το μήκος συν δύο φορές το πλάτος).

Άσκηση3

Γράψτε μια συνάρτηση που να παίρνει δύο ακέραια ορίσματα Unsigned short και να επιστρέφει το αποτέλεσμα της διαίρεσης του πρώτου με το δεύτερο. Μην κάνετε τη διαίρεση εάν ο δεύτερος αριθμός είναι μηδέν, αλλά επιστρέψτε το –1.

Κεφ.8 ΡΟΗ ΠΡΟΓΡΑΜΜΑΤΩΝ

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο θα αναλύσουμε τι είναι βρόγχος και πως χρησιμοποιούνται, πως να δημιουργείτε διάφορους βρόγχους.

Λέξεις κλειδιά

Βρόγχος goto
Βρόγχος while
Continue και break
do…while
Πολλαπλή αρχικοποίηση και αυξήσεις
Κενές προτάσεις σε βρόγχους for
Κενοί βρόγχοι for
Ένθετοι βρόγχοι
Switch

Ορισμός

Πολλά προβλήματα προγραμματισμού λύνονται ενεργώντας συνεχώς τα ίδια δεδομένα. Δύο τρόποι για να γίνει αυτό είναι η αναδρομή και η επανάληψη. Η

επανάληψη σημαίνει να γίνεται το ίδιο πράγμα ξανά και ξανά. Η κύρια μέθοδος επανάληψης είναι ο βρόγχος.

Βρόγχοι και οι ρίζες των βρόγχων: goto

Οι βρόγχοι αποτελούνται απλώς από ένα όνομα που ακολουθείται από μια άνω και κάτω τελεία **(:)**. Η ετικέτα τοποθετείται αριστερά μιας πρότασης. Μια μεταπήδηση επιτυγχάνεται γράφοντας το goto ακολουθούμενο από το όνομα μιας ετικέτας.

Δείτε το παρακάτω παράδειγμα:

```
1.  #include <iostream>
2.
3.      int main()
4.      {
5.            int counter = 0;
6.      loop:  counter ++;
7.            std:: cout << "counter: " << counter << "\n";
8.            if (counter < 5)
9.               goto loop;
10.
11.          std:: cout << "Complete. Counter: " << counter << ".\n";
12.       char response;
13.       std::cin >> response;
14.       return 0;
15.  }
```

Έξοδος :

counter: 1
counter: 2
counter: 3
counter: 4
counter: 5
Complete. Counter: 5.

Ανάλυση :

Στη γραμμή 5, ο counter αρχικοποιείται στο 0. Υπάρχει μια ετικέτα loop στη γραμμή 6, που σημειώνει την κορυφή του βρόγχου. Ο counter αυξάνεται και τυπώνεται η νέα τιμή του στη γραμμή 7. Η τιμή του counter εξετάζεται στη γραμμή 8. Εαν είναι μικρότερη από 5, η πρόταση if δίνει true και εκτελείται η πρόταση goto.Αυτό αναγκάζει την εκτέλεση του προγράμματος να μεταπηδήσει ξανά στην γραμμή 6. Το πρόγραμμα συνεχίζει το βρόγχο μέχρι ο counter να είναι ίσος με 5, οπότε βγαίνει από το βρόγχο και τυπώνεται η τελική έξοδος.

Σημείωση: Οι προτάσεις goto μπορούν να κάνουν μια μεταπήδηση σε οποιαδήποτε θέση στον κώδικα προέλευσης, προς τα πίσω ή προς τα εμπρός. Η χωρίς διάκριση χρήση των προτάσεων goto έχει προκαλέσει τη δημιουργία μπερδεμένων, άσχημων, αδύνατον να διαβαστούν προγραμμάτων, γνωστά ως «προγράμματα σπαγγέτι».

Βρόγχοι while

Ένας βρόγχος while αναγκάζει το πρόγραμμα σας να επαναλαμβάνει μια σειρά προτάσεων εφόσον η αρχική συνθήκη παραμένει αληθής. Η συνθήκη που εξετάζεται από έναν βρόγχο while μπορεί να είναι τόσο σύνθετη, όσο οποιαδήποτε έκφραση.

Το παρακάτω παράδειγμα δείχνει το ίδιο πρόγραμμα με το προηγούμενο χρησιμοποιώντας το βρόγχο while.

```
1.  #include <iostream>
2.
3.      int main()
4.      {
5.        using namespace std;
6.        int counter = 0;
7.
8.        while(counter < 5)
9.         {
10.          counter++;
11.          cout << "counter: " << counter << "\n";
12.      }
```

```
13.
14.       cout << "Complete. Counter: " << counter << ".\n";
15.        char response;
16.        std::cin>>response;
17.        return 0;
18.  }
```

Έξοδος :

counter: 1

counter: 2

counter: 3

counter: 4

counter: 5

Complete. Counter: 5.

Ανάλυση *:*

Στη γραμμή 6, δημιουργείται και αρχικοποιείται μια ακέραιη μεταβλητή που ονομάζεται counter. Αυτή εξετάζεται μετά ως μέρος μιας συνθήκης και εάν είναι αληθής, εκτελείται το σώμα του βρόγχου while. Η συνθήκη στη γραμμή 8 είναι εάν ο counter είναι μικρότερος από 5. Εαν η συνθήκη είναι αληθής, εκτελείτε το σώμα του βρόγχου. Στη γραμμή 10, αυξάνεται ο μετρητής και στη γραμμή 11 τυπώνεται η τιμή. Καλύτερα είναι να χρησιμοποιείται πάντα άγκιστρα γύρω από τα μπλοκ που εκτελείται από έναν βρόγχο.

Continue και break

Η πρόταση Continue μεταπηδά στην κορυφή του βρόγχου. Αν θελήσετε να βγείτε από τον βρόγχο πριν ικανοποιηθούν οι συνθήκες εξόδου χρησιμοποιείται η πρόταση break.

Η παρακάτω λίστα δείχνει τη χρήση αυτών των προτάσεων. Ο χρήστης εισάγει ένα μικρό αριθμό και ένα μεγάλο αριθμό, ένα αριθμό παράβλεψης και ένα αριθμό στόχο. Ο μικρός αριθμός θα αυξάνεται κατά ένα και ο μεγάλος αριθμός θα μειώνεται κατά δύο. Η μείωση θα αγνοείται κάθε φορά που ο μικρός αριθμός είναι ένα πολλαπλάσιο του αριθμού παράβλεψης. Το παιχνίδι τελειώνει εάν ο small γίνει

μεγαλύτερο από το large. Εάν ο μεγάλος αριθμός φτάσει ακριβώς στο στόχο, τυπώνεται μια πρόταση και το παιχνίδι τελειώνει.

```
1.  #include <iostream>
2.
3.      int main()
4.      {
5.        using namespace std;
6.        unsigned short small;
7.        unsigned long  large;
8.        unsigned long  skip;
9.        unsigned long target;
10.       const unsigned short MAXSMALL=65535;
11.
12.       cout << "Enter a small number: ";
13.       cin >> small;
14.       cout << "Enter a large number: ";
15.       cin >> large;
16.       cout << "Enter a skip number: ";
17.       cin >> skip;
18.       cout << "Enter a target number: ";
19.       cin >> target;
20.
21.     cout << "\n";
22.
23.       while (small < large && large > 0 && small < 65535)
24.
25.       {
26.
27.         small++;
28.
29.           if (small % skip == 0)
30.           {
31.             cout << "skipping on " << small << endl;
32.             continue;
```

```
33.         }
34.
35.         if (large == target)
36.         {
37.           cout << "Target reached!";
38.           break;
39.         }
40.
41.         large-=2;
42.       }
43.
44.      cout << "\nSmall: " << small << " Large: " << large << endl;
45.     char response;
46.     std:: cin >> response;
47.     return 0;
48.  }
```

Έξοδος :

Enter a small number: 2
Enter a large number: 20
Enter a skip number: 4
Enter a target number: 6
skipping on 4
skipping on 8
Small: 10 Large: 8

Ανάλυση :

Σε αυτό το πρόγραμμα ο χρήστης έχασε. Το small αυτό έγινε μεγαλύτερο από το large πριν φτάσει στον αριθμό στόχο, το 6.

Στη γραμμή 23, εξετάζονται οι συνθήκες while. Αν το small συνεχίζει να είναι μικρότερο από το large δεν έχει υπερβεί τη μέγιστη τιμή για ένα μικρό int, το πρόγραμμα μπαίνει στον βρόγχο while.

Στη γραμμή 29, βρίσκεται η τιμή του small modulo skip(τιμή παράβλεψης). Εάν το small είναι πολλαπλάσιο του skip, εκτελείται η πρόταση continue και η εκτέλεση

του προγράμματος μεταπηδά πάλι στην κορυφή. Έτσι αγνοείται ο έλεγχος για το στόχο και η μείωση του large.

Στη γραμμή 35, εξετάζεται το target (τιμή στόχος) ως προς την τιμή του large. Εάν οι τιμές είναι ίδιες, ο χρήστης κερδίζει. Τυπώνεται ένα μήνυμα και βρίσκεται και εκτελείται η πρόταση break. Αυτή η πρόταση σπάει το βρόγχο while και η εκτέλεση του προγράμματος συνεχίζει στη γραμμή 42.

Σημείωση: η Continue και break είναι επικίνδυνες εντολές για τον ίδιο λόγο όπως και με την εντολή goto. Καλύτερα είναι να χρησιμοποιείται την εντολή if.

Χειρισμός βρόγχων do…while

Είναι δυνατό το σώμα ενός while να μην εκτελεστεί ποτέ. Η πρόταση while ελέγχει τη συνθήκη της πριν εκτελέσει οποιεσδήποτε από τις προτάσεις της και αν η συνθήκη δώσει ψευδές αγνοείται ολόκληρο το σώμα του βρόγχου while.

Το παρακάτω παράδειγμα παρουσιάζει αυτό το θέμα:

```
1.  #include <iostream>
2.
3.      int main()
4.      {
5.        int counter;
6.        std::cout << "How many hellos?: ";
7.        std::cin >> counter;
8.        while (counter > 0)
9.        {
10.         std::cout << "Hello!\n";
11.         counter--;
12.       }
13.      std::cout << "Counter is OutPut: " << counter;
14.       char response;
15.       std::cin>> response;
16.       return 0;
17. }
```

Έξοδος :

How many hellos?: 2

Hello!

Hello!

Counter is OutPut: 0

Ανάλυση :

Ζητείται από τον χρήστη μια αρχική τιμή στη γραμμή 6.Αυτή η αρχική τιμή αποθηκεύεται στην ακέραιη μεταβλητή counter. Η τιμή του counter ελέγχεται στη γραμμή 8 και στο σώμα του βρόγχου while.

Αν θέλετε να εξασφαλίσετε ότι το Hello θα τυπώνεται πάντα μπορείτε να το επιτύχετε με μια πρόταση **if** αμέσως πριν αρχίσει ο βρόγχος while.

if (counter < 1) //το αναγκάζει να πάρει μια ελάχιστη τιμή

counter = 1;

Βρόγχοι με την πρόταση for

Κατά τον προγραμματισμό με βρόγχους while, συχνά θα ακολουθείται τρία βήματα:

- θα διαμορφώνετε μια αρχική συνθήκη
- θα ελέγχετε εάν η συνθήκη είναι αληθής
- και θα αυξάνετε ή θα αλλάζετε μια μεταβλητή κάθε φορά που περνάτε από τον βρόγχο.

Η παρακάτω λίστα σας δείχνει αυτό.

```
1.  #include <iostream>
2.
3.      int main()
4.      {
5.        int counter = 0;
6.
7.        while(counter < 5)
8.        {
9.            counter++;
10.           std::cout << "Looping!  ";
```

```
11.      }
12.
13.      std::cout << "\nCounter: " << counter << ".\n";
14.       char response;
15.       std::cin >> response;
16.       return 0;
17.  }
```

Έξοδος :

Looping! Looping! Looping! Looping! Looping!
Counter: 5.

Ανάλυση :

Στη γραμμή 5 ορίζεται η αρχική συνθήκη και το counter αρχικοποιείται στο 0. Στη γραμμή 7, γίνεται ο έλεγχος της συνθήκης και ελέγχεται αν το counter είναι μικρότερο από το 5. Τέλος η μεταβλητή counter αυξάνεται στη γραμμή 9.

Ένας βρόγχος for συνδυάζει τα τρία βήματα σε μια πρόταση. Τα τρία βήματα είναι η αρχικοποίηση, ο έλεγχος και η αύξηση. Μια πρόταση for αποτελείται από τη λέξη κλειδί for, ακολουθούμενη από ένα ζευγάρι παρενθέσεις.

For (αρχικοποίηση; Έλεγχος; Ενέργεια ;)
{
.....
}

Η αρχικοποίηση, μπορεί να είναι οποιαδήποτε έγκυρη πρόταση στη C++, αλλά γενικά χρησιμοποιείται για να δημιουργήσει και να αρχικοποιήσει μια μεταβλητή καταμέτρησης. Ο έλεγχος, είναι εκεί που γίνεται έλεγχος. Η ενέργεια, είναι η ενέργεια που θα γίνει.

Προχωρημένοι βρόγχοι for

§ Πολλαπλή αρχικοποίηση και αυξήσεις

Δεν είναι ασυνήθιστο να αρχικοποιήσετε περισσότερες από μια μεταβλητές, να εξετάζετε μια σύνθετη λογική έκφραση και να εκτελείτε περισσότερες από μια προτάσεις.

Δείτε το παρακάτω παράδειγμα:

```
1.  #include <iostream>
2.
3.    int main()
4.    {
5.        for (int i=0, j=0; i<3; i++, j++)
6.            std::cout << "i: " << i << " j: " << j << std::endl;
7.        char response;
8.        std::cin >> response;
9.       return 0;
10.  }
```

Έξοδος :

i: 0 j: 0

i: 1 j: 1

i: 2 j: 2

Ανάλυση :

Στη γραμμή 5, αρχικοποιούνται οι δύο μεταβλητές i και j σε τιμή 0. Χρησιμοποιείτε ένα κόμμα για να διαχωριστούν οι δύο διαφορετικές εκφράσεις. Βλέπετε ότι οι αρχικοποιήσεις χωρίζονται με ένα ελληνικό ερωτηματικό.

Υπολογίζεται ο έλεγχος (i<3) και επειδή είναι αληθής, εκτελείται το σώμα της πρότασης for.

§ Κενές προτάσεις σε βρόγχους for

Οποιασδήποτε ή όλες οι προτάσεις ενός βρόγχου for μπορούν να παραληφθούν, για να το επιτύχετε χρησιμοποιήστε μια κενή πρόταση. Μια κενή πρόταση είναι απλώς ένα ελληνικό ερωτηματικό.

Δείτε το παρακάτω:

```
1.  #include <iostream>
2.
3.      int main()
4.      {
5.          int counter = 0;
6.
7.          for( ; counter < 5; )
8.          {
9.              counter++;
10.             std::cout << "Looping!  ";
11.         }
12.         std::cout << "\nCounter: " << counter << ".\n";
13.     char response;
14.     std::cin >> response;
15.     return 0;
16. }
```

Έξοδος :

Looping! Looping! Looping! Looping! Looping!

Counter: 5

Ανάλυση :

Η πρόταση for στη γραμμή 7 δεν αρχικοποιεί κάποιες τιμές, αλλά περιλαμβάνει ένα έλεγχο για το αν το counter < 5. Δεν υπάρχει καμία πρόταση αύξησης, έτσι αυτός ο βρόγχος συμπεριφέρεται ακριβώς σαν να είχατε γράψει:

While (counter < 5)

- **Κενοί βρόγχοι for**

Επειδή γίνονται τόσα πολλά στην επικεφαλίδα μιας πρότασης for, μερικές φορές δεν θα χρειάζεται να βάλετε τίποτα στο σώμα του βρόγχου. Σε αυτή την περίπτωση, θα πρέπει να βάλετε μια κενή πρόταση (;) για σώμα του βρόγχου. Το ελληνικό

ερωτηματικό μπορεί να είναι στην ίδια γραμμή με την επικεφαλίδα, αλλά είναι εύκολο να το παραβλέψετε αυτό.

Δείτε το παρακάτω:

```
1.  #include <iostream>
2.      int main()
3.      {
4.        for (int i = 0; i<5; std::cout << "i: " << i++ << std::endl)
5.        ;
6.        char response;
7.        std::cin >> response;
8.        return 0;
9.   }
```

Έξοδος :

i: 0

i: 1

i: 2

i: 3

i: 4

Ανάλυση :

Ο βρόγχος for στη γραμμή 4 περιλαμβάνει τρεις προτάσεις: η πρόταση της αρχικοποίησης ορίζει το μετρητή i και τον αρχικοποιεί στο 0. Η πρόταση του ελέγχου ελέγχει αν το i < 5 και η πρόταση ενέργειας τυπώνει την τιμή του i και την αυξάνει.

Αυτό θα μπορούσε να ξαναγραφεί καλύτερα ως εξής:

```
for ( int i = 0;  i<5;  i++ )
   cout << "i: " << i << endl;
```

- **Ένθετοι βρόγχοι**

Οι βρόγχοι μπορούν να τοποθετηθούν ένθετοι, ο ένας μέσα στο σώμα του άλλου. Ο εσωτερικός βρόγχος θα εκτελείται πλήρως σε κάθε εκτέλεση του εξωτερικού βρόγχου.

Δείτε το παρακάτω:

```
1.  #include <iostream>
2.
```

```
3.  int main()
4.  {
5.      using namespace std;
6.      int rows, columns;
7.      char theChar;
8.      cout << "How many rows? ";
9.      cin >> rows;
10.     cout << "How many columns? ";
11.     cin >> columns;
12.     cout << "What character? ";
13.     cin >> theChar;
14.     for (int i = 0; i<rows; i++)
15.     {
16.       for (int j = 0; j<columns; j++)
17.           cout << theChar;
18.       cout << "\n";
19.     }
20.     char response;
21.     std::cin >> response;
22.     return 0;
23. }
```

Έξοδος :

How many rows? 3

How many columns? 14

What character? $

$$$$$$$$$$$$$

$$$$$$$$$$$$$

$$$$$$$$$$$$$

Ανάλυση :

Σε αυτή τη λίστα ζητείται από τον χρήστη να δώσει τον αριθμό των γραμμών και των στηλών και ένα χαρακτήρα που θα τυπωθεί. Ο πρώτος βρόγχος for, στη γραμμή 14, αρχικοποιεί ένα μετρητή i στο 0 και μετά τρέχει το σώμα του εξωτερικού βρόγχου for.

Στη γραμμή 16, ορίζεται ένας άλλος βρόγχος for. Αρχικοποιείται ένας δεύτερος μετρητής j στο 0 και εκτελείται το σώμα του εσωτερικού βρόγχου for.

Στη γραμμή 17 τυπώνεται ο επιλεγμένος χαρακτήρας και ο έλεγχος επιστρέφει στην επικεφαλίδα του εσωτερικού βρόγχου for. Εξετάζετε η συνθήκη j<columns και εάν είναι αληθής, αυξάνεται το j και τυπώνεται ο επόμενος χαρακτήρας. Αυτό συνεχίζεται μέχρι το j να γίνει ίσο με τον αριθμό των στηλών.

Όταν ο εσωτερικός βρόγχος for δώσει ψευδές στον έλεγχό του, σε αυτήν την περίπτωση αφού τυπωθούν τα 14$, η εκτέλεση πηγαίνει κατευθείαν στην γραμμή 18 και τυπώνεται μια νέα γραμμή. Ο εξωτερικός βρόγχος for επιστρέφει τώρα στη επικεφαλίδα του, όπου ελέγχεται η συνθήκη του i<rows. Εάν αυτό δώσει αληθές, το i αυξάνετε και εκτελείται το σώμα του βρόγχου.

Στη δεύτερη επανάληψη του εξωτερικού βρόγχου for, αρχίζει ο εσωτερικός βρόγχος for. Κατά συνέπεια, το j αρχικοποιείται πάλι στο 0 και τρέχει πάλι ολόκληρος ο εσωτερικός βρόγχος.

Έλεγχος της ροής με προτάσεις switch

Οι προτάσεις switch σας επιτρέπουν να διακλαδώνεστε με διάφορες τιμές. Η γενική μορφή της switch είναι :

```
switch (έκφραση)
{
case valueOne: πρόταση;
               break;
case valueTwo: πρόταση;
               break;
....
case valueN:   πρόταση;
               break;
default:       πρόταση;
}
```

Η έκφραση είναι οποιαδήποτε έγκυρη έκφραση στη C++ και οι προτάσεις είναι έγκυρες προτάσεις στη C++ ή μπλοκ από προτάσεις που υπολογίζουν μια ακέραιη

τιμή. Σημειώστε ωστόσο ότι ο υπολογισμός γίνεται μόνο για την ισότητα. Δεν μπορούν να χρησιμοποιηθούν εδώ σχεσιακοί τελεστές ούτε λογικές πράξεις.

Εάν μια από τις τιμές **case** ταιριάζει με την έκφραση, η εκτέλεση του προγράμματος μεταπηδά σε αυτές τις προτάσεις και συνεχίζει έως το τέλος του μπλοκ switch, εκτός και αν βρεθεί μια πρόταση **break**. Εάν τίποτα δεν ταιριάζει με την έκφραση, η εκτέλεση διακλαδώνεται στην προαιρετική πρόταση **default**. Εάν δεν υπάρχει η default και καμία τιμή δεν ταιριάζει με την έκφραση, η εκτέλεση περνά έξω από την εντολή switch και η πρόταση τελειώνει.

Πρέπει να σημειωθεί ότι αν δεν υπάρχει καμία πρόταση break στο τέλος μια πρότασης case, η εκτέλεση θα πάει στην επόμενη πρόταση case.

Ασκήσεις αξιολόγησης

Άσκηση1

Γράψτε ένα βρόγχο for που να τυπώνει ένα μοτίβο 10 x 10 από 0.

Άσκηση2

Γράψτε ένα βρόγχο while που να μετρά από 100 έως 200 κατά δύο.

Άσκηση3

Γράψτε ένα βρόγχο do…while που να μετρά από 100 έως 200 κατά δύο.

Άσκηση4

Τι είναι λάθος σε αυτόν τον κώδικα;

```
1. int counter = 100;
2. while (counter < 10)
3. {
4. cout << “counter now: ” << counter;
5. counter--;
6. }
```

Κεφ.9 ΠΙΝΑΚΕΣ

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο θα μάθουμε τι είναι ένας πίνακας και πως τον δηλώνουμε, τη σχέση μεταξύ πινάκων και δεικτών, πως να χρησιμοποιείτε την αριθμητική δεικτών και τι είναι οι συνδεδεμένες λίστες.

Λέξεις κλειδιά

Πολυδιάστατους Πίνακες
Αριθμητική δεικτών
Μέθοδο cin.get()
strcpy() και strncpy()
Κλάσεις συμβολοσειρών

Ορισμός

Ένας πίνακας (array) είναι μια σειριακή συλλογή θέσεων αποθήκευσης δεδομένων, όπου κάθε μια περιέχει τον ίδιο τύπο δεδομένων. Κάθε θέση αποθήκευσης ονομάζεται στοιχείο (element) του πίνακα. Δηλώνετε έναν πίνακα γράφοντας τον τύπο του, ακολουθούμενο από το όνομα του πίνακα και του δείκτη του πίνακα (subscript).Ο δείκτης του πίνακα είναι ο αριθμός των στοιχείων του πίνακα, μέσα σε τετράγωνες αγκύλες. Για παράδειγμα,

long LongArray[25];

Δηλώνει ένα πίνακα με 25 long ακέραιους που ονομάζεται LongArray.

Πρόσβαση σε στοιχεία του πίνακα

Έχετε πρόσβαση σε ένα στοιχείο ενός πίνακα αναφέροντας τη μετατόπιση του από την αρχή του πίνακα. Οι μετατοπίσεις των στοιχείων του πίνακα μετρούνται από το μηδέν. Επομένως το πρώτο στοιχείο του πίνακα αναφέρεται ως arrayName[0]. Γενικότερα ο πίνακας SomeArray[n] έχει n στοιχεία που είναι αριθμημένα από το SomeArray[0] έως το SomeArray[n-1]. Επομένως το LongArray[25] αριθμείται από το LongArray[0] έως το LongArray[24].

Η παρακάτω λίστα δείχνει πως να δηλώνεται έναν πίνακα από πέντε ακέραιους και να δίνετε σε κάθε ακέραιο μια τιμή.

```
1. #include <iostream>
2.
3. int main()
4. {
5.     int myArray[5];
6.     int i;
7.     for (i=0; i<5; i++)
8.     {
9.         std::cout << "value for myArray[" << i << "]: ";
10.        std::cin >> myArray[i];
11. }
12. for (i = 0; i<5; i++)
13. std::cout << i << ": " << myArray[i] << std::endl;
```

```
14. char response;
15. std::cin >> response;
16. return 0;
17. }
```

Έξοδος :

value for myArray[0]: 3
value for myArray[1]: 6
value for myArray[2]: 9
value for myArray[3]: 12
value for myArray[4]: 15
0: 3
1: 6
2: 9
3: 12
4: 15

Ανάλυση :

Στη γραμμή 5 δηλώνεται ο πίνακας και είναι τύπου ακεραίου. Στη γραμμή 7 αρχίζει ένας βρόγχος for που μετρά από το 0 έως το 4. Στη γραμμή 10 βλέπετε ότι κάθε στοιχείο προσπελάζεται με το όνομα του πίνακα, ακολουθούμενο από τετράγωνες αγκύλες, που περιέχουν τη μετατόπιση. Κάθε ένα από αυτά τα στοιχεία μπορεί έπειτα να τα χειριστείτε σαν μια μεταβλητή τύπου πίνακα

Γράφοντας μετά το τέλος του πίνακα

Όταν γράφετε σε ένα στοιχείο ενός πίνακα, ο μεταγλωττιστής υπολογίζει που θα αποθήκευση τη τιμή ανάλογα με το μέγεθος κάθε στοιχείου και του δείκτη του πίνακα. Υποθέστε ότι θέλετε να γράψετε την τιμή στο LongArray[5], το οποίο είναι το έκτο στοιχείο. Ο μεταγλωττιστής πολλαπλασιάζει τη μετατόπιση (5) με το μέγεθος κάθε στοιχείου- σε αυτή την περίπτωση, 4 byte. Έπειτα προχωρά κατά τόσα byte (20) από την αρχή του πίνακα και γράφει τη νέα τιμή σε αυτή τη θέση.

Αρχικοποιώντας πίνακες

Μπορείτε να αρχικοποιήσετε έναν απλό πίνακα σε ενσωματωμένους τύπους, όπως ακέραιους και χαρακτήρες, όταν δηλώνετε αρχικά τον πίνακα. Μετά από το όνομα του πίνακα, βάζετε ένα σύμβολό ίσον (=) και μια λίστα τιμών χωρισμένων σε κόμματα μέσα σε άγκιστρα.

Παράδειγμα:

int IntegerArray[5] = {10, 20, 30, 40, 50};

Δεν μπορείτε να αρχικοποιήστε περισσότερα στοιχεία από όσα έχετε δηλώσει για τον πίνακα.

Παράδειγμα:

int IntegerArray[5] = {10, 20, 30, 40, 50, 60};

παράγει ένα λάθος μεταγλωττιστή επειδή έχετε δηλώσει έναν πίνακα πέντε στοιχείων και αρχικοποιείτε έξι τιμές.

Δήλωση πινάκων

Για να δηλώσετε έναν πίνακα, γράφετε τον τύπο του αντικειμένου που θα περιέχει, ακολουθούμενο από το όνομα του πίνακα και έναν δείκτη πίνακα με τον αριθμό των αντικειμένων που θα περιέχονται στον πίνακα.

Παράδειγμα:

int MyIntegerArray[90];

Για να έχετε πρόσβαση στα μέλη του πίνακα, χρησιμοποιήστε τον τελεστή δείκτη πίνακα (subscript). Οι πίνακες μπορεί να έχουν ως όνομα οποιοδήποτε σωστό όνομα μεταβλητής, αλλά δεν μπορεί να έχουν το ίδιο όνομα με μια άλλη μεταβλητή ή πίνακα στο ίδιο πεδίο δράσης. Επομένως δεν μπορείτε να ονομάσετε συγχρόνως έναν πίνακα myCats[5] και μια μεταβλητή myCats.

Επιπλέον όταν δηλώνεται τον αριθμό των στοιχείων, εκτός από τη χρησιμοποίηση κυριολεκτικών αριθμών, μπορείτε να χρησιμοποιήσετε μια σταθερά (constant) ή μισά απαριθμητή (enumeration). Είναι πραγματικά καλύτερο να χρησιμοποιήσετε αυτές αντί να χρησιμοποιήσετε έναν κυριολεκτικό αριθμό επειδή σας δίνει μια μόνο θέση από την οποία μπορείτε να ελέγχετε τον αριθμό των στοιχείων. Αν θέλετε να αλλάξετε το TargetArray ώστε να περιέχει μόνο 20 στοιχεία αντί για 25, θα πρέπει να

αλλάξετε αρκετές γραμμές του κώδικα. Εάν χρησιμοποιήσετε μια σταθερά, θα πρέπει να αλλάξετε μόνο την τιμή της σταθεράς σας.

Η δημιουργία του αριθμού των στοιχείων, ή το μέγεθος της διάστασης του πίνακα, με μια απαριθμητή γίνεται λίγο διαφορετικά. Η παρακάτω λίστα επεξηγεί αυτό δημιουργώντας έναν πίνακα που περιέχει τιμές - μια για κάθε μέρα της εβδομάδας.

```
1. #include <iostream>
2.
3. int main()
4. {
5.     enum WeekDays {Sun, Mon, Tue, Wed, Thu, Fri, Sat, DayInWeek };
6.     int ArrayWeek[DayInWeek] = {10, 20, 30, 40, 50, 60, 70};
7.     std::cout << "The value at Tuesday is: " << ArrayWeek[Tue];
8.     char response;
9.     std::cin >> response;
10.    return 0;
11. }
```

Έξοδος :

The value at Tuesday is: 30

Ανάλυση :

Η γραμμή 4 δημιουργεί μια απαριθμητή που ονομάζεται WeekDays. Στη γραμμή 5, δηλώνεται ένας πίνακας, που ονομάζεται ArrayWeek με DayInWeek στοιχεία το οποίο είναι ίσο με 7. Η γραμμή 6χρησιμοποιεί την απαριθμητή σταθερά Tue σαν μετατόπιση στον πίνακα.

Αρχικοποιώντας Πολυδιάστατους Πίνακες

Είναι δυνατό να έχετε πίνακες με περισσότερες από μια διαστάσεις. Κάθε διάσταση αντιπροσωπεύεται από έναν δείκτη πίνακα στον πίνακα. Για να αρχικοποιήσετε πολυδιάστατους πίνακες αντιστοιχείτε τη λίστα τιμών σε στοιχεία του πίνακα με τη σειρά, αλλάζοντας τον τελευταίο δείκτη πίνακα ενώ μένουν σταθεροί οι πρώτοι δείκτες του πίνακα. Επομένως αν έχετε έναν πίνακα:

int theArray[5][3];

Τα πρώτα τρία στοιχεία πηγαίνουν στο theArray[0];, τα επόμενα τρία theArray[1]; και λοιπά.

Κατά την αρχικοποίηση στοιχείων ενός πίνακα, κάθε τιμή πρέπει να χωρίζεται με ένα κόμμα αδιαφορώντας για τα άγκιστρα. Ολόκληρο το σύνολο που αρχικοποιείται πρέπει να είναι μέσα σε άγκιστρα και πρέπει να τελειώνει με ένα ελληνικό ερωτηματικό.

Η παρακάτω λίστα δημιουργεί ένα δισδιάστατο πίνακα. Η πρώτη διάσταση είναι το σύνολο αριθμών από 0-4. Η δεύτερη διάσταση αποτελείται από το διπλάσιο κάθε τιμής της πρώτης διάστασης.

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main()
5.  {
6.      int SomeArray[2][5]= { {0,1,2,3,4}, {0,2,4,6,8}};
7.      for (int i=0; i<2; i++)
8.      {
9.          for (int j=0; j<5; j++)
10.         {
11.             cout << "SomeArray[" << i << "][" << j << "]: ";
12.             cout << SomeArray[i][j]<<endl;
13.         }
14.     }
15.     char response;
16.     std::cin >> response;
17.     return 0;
18. }
```

Έξοδος :

SomeArray[0][0]: 0
SomeArray[0][1]: 1
SomeArray[0][2]: 2
SomeArray[0][3]: 3
SomeArray[0][4]: 4

SomeArray[1][0]: 0
SomeArray[1][1]: 2
SomeArray[1][2]: 4
SomeArray[1][3]: 6
SomeArray[1][4]: 8

Ανάλυση :

Η γραμμή 6 δηλώνει το SomeArray ως ένα δισδιάστατο πίνακα. Η πρώτη διάσταση υποδεικνύει ότι θα υπάρχουν δύο σύνολα και η δεύτερη διάσταση αποτελείται από 5 ακέραιους αριθμούς. Οι τιμές βασίζονται στα δύο σύνολα αριθμών. Το πρώτο σύνολο είναι οι αρχικοί αριθμοί και το δεύτερο σύνολο είναι οι διπλασιασμένοι αριθμοί. Οι γραμμές 7 και 9 δημιουργούν ένα ένθετο βρόγχο for. Ο εξωτερικός βρόγχος for περνά από κάθε μέλος της πρώτης διάστασης, που είναι κάθε ένα από τα δύο σύνολα ακέραιων αριθμών. Για κάθε μέλος σε αυτή τη διάσταση, ο εσωτερικός βρόγχος for περνά από κάθε μέλος της δεύτερης διάστασης. Η πρώτη διάσταση αυξάνεται μόνο όταν περάσει η δεύτερη διάσταση από όλες τις αυξήσεις. Κατόπιν αρχίζει ο υπολογισμός της δεύτερης διάστασης.

Δημιουργία πινάκων από δείκτες

Οι πίνακες που συζητήθηκαν μέχρι τώρα αποθηκεύουν όλα τα μέλη τους στην στοίβα. Συνήθως, η μνήμη της στοίβας είναι πιο περιορισμένη, ενώ η ελεύθερη μνήμη είναι πολύ μεγαλύτερη. Είναι δυνατό να δηλωθεί κάθε αντικείμενο στην ελεύθερη μνήμη και έπειτα να αποθηκευτεί μόνο ένας δείκτης για το αντικείμενο στον πίνακα.

Η παρακάτω λίστα αποθηκεύει όλα τα αντικείμενα στην ελεύθερη μνήμη.

```
1. #include <iostream>
2. using namespace std;
3.
4. class Cat
5. {
6.       public:
7.             Cat() {itsAge = 1; itsWeight = 5; }
8.             ~Cat() {}
```

```
9.              int GetAge() const {return itsAge; }
10.             int GetWeight() const {return itsWeight; }
11.             void SetAge(int age) {itsAge = age; }
12.
13.      private:
14.              int itsAge;
15.              int itsWeight;
16. };
17. int main()
18. {
19.     Cat * Family[200];
20.     int i;
21.     Cat * pCat;
22.     for (i=0; i<200; i++)
23.     {
24.         pCat=new Cat;
25.         pCat->SetAge(2*i +1);
26.         Family[i]=pCat;
27.     }
28. for (i=0; i<200; i++)
29.     {
30.          cout << "Cat #" <<i+1 << ": ";
31.          cout << Family[i]->GetAge() << endl;
32.      }
33.      char response;
34.     std::cin >> response;
35.     return 0;
36. }
```

Έξοδος :

Cat #1: 1

Cat #2: 3

Cat #3: 5

.....

Cat #198: 395
Cat #199: 397
Cat #200: 399

Ανάλυση :

Στη γραμμή 19 δηλώνεται ο πίνακας που ονομάζεται Family και δηλώνεται να περιέχει 200 στοιχεία. Αυτά τα στοιχεία είναι δείκτες σε αντικείμενα Cat. Στον αρχικό βρόγχο στις γραμμές 22-27 δημιουργούνται 200 νέα αντικείμενα Cat στην ελεύθερη μνήμη και η ηλικία καθενός είναι δύο φορές ο δείκτης συν ένα. Αφού δημιουργηθεί ο δείκτης η γραμμή 26 εκχωρεί τον δείκτη στον πίνακα. Επειδή ο πίνακας έχει δηλωθεί να περιέχει δείκτες, προστίθεται ο δείκτης και όχι η αποαναφερθείσα τιμή του δείκτη-στον πίνακα.

Ο δεύτερος βρόγχος στις γραμμές 28-32 τυπώνει κάθε μια από τις τιμές. Στη γραμμή 31, ο δείκτης προσπελάζεται χρησιμοποιώντας το δείκτη πίνακα Family[i].

Δήλωση πινάκων στη ελεύθερη μνήμη

Είναι δυνατόν να βάλετε ολόκληρο τον πίνακα στην ελεύθερη μνήμη, που είναι επίσης γνωστή ως σωρός. Το κάνετε αυτό δημιουργώντας ένα δείκτη στον πίνακα. Δημιουργείται τον δείκτη καλώντας το new και χρησιμοποιώντας τον τελεστή δείκτη πίνακα. Το αποτέλεσμα είναι ένας δείκτης σε μια περιοχή στην ελεύθερη μνήμη που περιέχει τον πίνακα.

Παράδειγμα:

Cat *Family = new Cat[500];

Δηλώνει το Family να είναι ένας δείκτης στο πρώτο στοιχείο του πίνακα των 500 Cat.

Το πλεονέκτημα της χρήσης του Family κατά αυτό τον τρόπο είναι ότι μπορείτε να χρησιμοποιήσετε αριθμητική δεικτών για να έχετε πρόσβαση σε κάθε μέλος του Family.

Ονόματα δεικτών και πινάκων

Στη C++, ένα όνομα πίνακα είναι ένας σταθερός δείκτης στο πρώτο στοιχείο του πίνακα. Επομένως στη δήλωση:

Cat Family[500];

Το Family είναι ένας δείκτης στο &Family[0], το οποίο είναι η διεύθυνση του πρώτου στοιχείου του πίνακα Family.

Ο μεταγλωττιστής κάνει όλες τις πράξεις όταν προσθέτετε, αυξάνετε και μειώνετε δείκτες. Η διεύθυνση που προσπελάζεται όταν γράφετε Family + 4 δεν είναι 4 byte μετά την διεύθυνση του Family - είναι 4 αντικείμενα. Εάν κάθε αντικείμενο είναι 4 byte, τότε Family + 4 είναι 16 byte μετά την αρχή του πίνακα. Εάν κάθε αντικείμενο είναι ένα Cat που έχει τέσσερις μεταβλητές μέλη long των τεσσάρων byte η κάθε μια και δύο μεταβλητές μέλη short των δύο byte η κάθε μια, τότε κάθε Cat είναι 20 byte και το Family + 4 είναι 80 byte μετά την αρχή του πίνακα.

Πίνακες και συμβολοσειρές char

Υπάρχει ένας τύπος πίνακα που χρειάζεται ιδιαίτερη προσοχή. Αυτός είναι ένας πίνακας χαρακτήρων που τερματίζει με ένα null. Αυτός ο πίνακας θεωρείτε μια «συμβολοσειρά στυλ C». Μπορείτε να δηλώσετε και να αρχικοποιήσετε μια συμβολοσειρά στυλ C όπως θα κάνατε με οποιονδήποτε άλλο πίνακα.

Για παράδειγμα:

char Greeting[]= {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd','\0' };

Σε αυτή την περίπτωση, το Greeting δηλώνεται ως ένας πίνακας χαρακτήρων και αρχικοποιείται με διάφορους χαρακτήρες. Η C++ σας επιτρέπει να χρησιμοποιήσετε μια μορφή συντόμευσης της προηγούμενης γραμμής κώδικα. Αυτή είναι:

char Greeting[] = "Hello World";

Πρέπει να σημειώσετε δύο πράγματα με αυτή τη σύνταξη:

- Αντί για χαρακτήρες σε μονά εισαγωγικά που χωρίζονται με κόμματα και περικλείονται από άγκιστρα, έχετε μια συμβολοσειρά με διπλά εισαγωγικά στυλ C, χωρίς κόμματα και χωρίς άγκιστρα.
- Δεν χρειάζεται να προσθέσετε τον χαρακτήρα null, επειδή ο μεταγλωττιστής τον προσθέτει αυτόματα

Όταν δηλώνετε μια συμβολοσειρά, πρέπει να εξασφαλίσετε ότι θα την κάνετε αρκετά μεγάλη. Το μήκος της συμβολοσειράς στυλ C περιλαμβάνει τον αριθμό των χαρακτήρων, συμπεριλαμβανομένου του χαρακτήρα null. Μπορείτε επίσης να δημιουργήσετε μη αρχικοποιημένους πίνακες χαρακτήρων.

Η λίστα δείχνει τη χρήση μη αρχικοποιημένου buffer.

```
1.  #include <iostream>
2.
3.  int main()
4.  {
5.      char buffer[80];
6.      std::cout << "Enter the string: ";
7.      std::cin >> buffer;
8.      std::cout << "Here's the buffer: " << buffer << std::endl;
9.      char response;
10.     std::cin >> response;
11.     return 0;
12. }
```

Έξοδος :

Enter the string: ***Hello World***

Here's tje buffer: Hello

Ανάλυση :

Στη γραμμή 5 δημιουργείται ένας πίνακας χαρακτήρων που θα ενεργεί ως buffer για 80 χαρακτήρες. Στη γραμμή 6 ζητείται από τον χρήστη να εισάγει μια συμβολοσειρά στυλ C, η οποία εισάγεται στο buffer στη γραμμή 7. Το cin γράφει ένα τερματικό null στο buffer αφού γράψει τη συμβολοσειρά.

Δύο προβλήματα εμφανίζονται στο πρόγραμμα της παραπάνω λίστας.

- Εάν ο χρήστης εισάγει περισσότερους από 79 χαρακτήρες, το cin γράφει μετά από το τέλος του buffer.
- Εάν ο χρήστης εισάγει ένα κενό, το cin νομίζει ότι είναι το τέλος της συμβολοσειράς και σταματά να γράφει στο buffer.

Για να λύσετε αυτά τα προβλήματα, πρέπει να καλέσετε μια ειδική μέθοδο στο cin , που ονομάζεται get(). Η **cin.get()** παίρνει τρεις παραμέτρους:

- Το buffer που θα γεμίσει
- Το μέγιστο αριθμό χαρακτήρων
- Το χαρακτήρα οριοθέτησης που τερματίζει την εισαγωγή.

Η παρακάτω λίστα δείχνει τη **χρήση της get()** :

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     char buffer[80];
7.     std::cout << "Enter the string: ";
8.     cin.get(buffer, 79);
9.     std::cout << "Here's the buffer: " << buffer << std::endl;
10.    char response;
11.    std::cin >> response;
12.    return 0;
13. }
```

Έξοδος :

Enter the string: ***Hello World***

Here's the buffer: Hello World

Ανάλυση :

Η γραμμή 8 καλεί τη μέθοδο get() του cin.Το buffer που δηλώνεται στη γραμμή 6 περνά μέσα ως το πρώτο όρισμα. Το δεύτερο όρισμα είναι ο μέγιστος αριθμός χαρακτήρων που μπορούν να εισαχθούν. Σε αυτή την περίπτωση, δεν πρέπει να είναι μεγαλύτερο από 79, για να μπορεί να υπάρχει και το τερματικό null. Εάν εισάγεται κενά, στηλοθέτες, ή άλλους κενούς χαρακτήρες, αυτοί θα εισαχθούν στη συμβολοσειρά. Ένας χαρακτήρας αλλαγής γραμμής τερματίζει την εισαγωγή.

Χρήση των μεθόδων strcpy() και strncpy()

Είναι διαθέσιμες διάφορες υπάρχουσες συναρτήσεις στη βιβλιοθήκη της C++ για χειρισμό συμβολοσειρών. Μεταξύ των πολλών παρεχόμενων συναρτήσεων, υπάρχουν δύο για αντιγραφή μιας συμβολοσειράς σε μια άλλη: η strcpy() και η strncpy(). Η strcpy() αντιγράφει ολόκληρα τα περιεχόμενα μιας συμβολοσειράς σε έναν υποδεικνυόμενο buffer. Η strncpy() αντιγράφει διάφορους χαρακτήρες από μια συμβολοσειρά σε μια άλλη.

Η παρακάτω λίστα δείχνει τη χρήση της **strcpy()**:

```
1. #include <iostream>
2. #include <string.h>
3. using namespace std;
4.
5. int main()
6. {
7.     char String1[] = "No man is an island";
8.     char String2[80];
9.
10.    strcpy(String2, String1);
11.
12.    cout << "String1: " << String1 << endl;
13.    cout << "String2: " << String2 << endl;
14.
15.    char response;
16.    std::cin >> response;
17.    return 0;
18. }
```

Έξοδος :

String1: No man is an island

String2: No man is an island

Ανάλυση :

Το αρχείο επικεφαλίδας string.h περιέχει το πρωτότυπο της συνάρτησης strcpy().Η strcpy() παίρνει δύο πίνακες χαρακτήρων, ο ένας είναι ο προορισμός και ακολουθεί η προέλευση. Στη γραμμή 10, χρησιμοποιείται αυτή η συνάρτηση για να αντιγράψει το String1 στο String2.

Πρέπει να είστε προσεκτικοί όταν χρησιμοποιείτε τη συνάρτηση strcpy(). Εάν η προέλευση είναι μεγαλύτερη από τον προορισμό, η strcpy() γράφει και μετά το τέλος του buffer.Για να προστατευτείτε από αυτό, η Standart Library περιλαμβάνει επίσης την strncpy(). Αυτή η παραλλαγή παίρνει ένα μέγιστο αριθμό χαρακτήρων που θα αντιγραφούν. Η strncpy() αντιγράφει μέχρι τον πρώτο χαρακτήρα null ή το μέγιστο αριθμό χαρακτήρων που καθορίζονται στο buffer προορισμού.

Η λίστα που ακολουθεί επεξηγεί τη χρήση της **strncpy()**:

```
1.  #include <iostream>
2.  #include <string.h>
3.
4.  int main()
5.  {
6.      const int MaxLength = 80;
7.      char String1[] = "No man is an island";
8.      char String2[MaxLength+1];
9.
10.     strncpy(String2, String1, MaxLength);
11.
12.     std::cout << "String1: " << String1 << std::endl;
13.     std::cout << "String2: " << String2 << std::endl;
14.
15.     char response;
16.     std::cin >> response;
17.     return 0;
18. }
```

Έξοδος :

String1: No man is an island

String2: No man is an island

Ανάλυση :

Στη γραμμή 10 η strncpy() παίρνει μια τρίτη παράμετρο: τον μέγιστο αριθμό χαρακτήρων που θα αντιγραφούν. Το buffer String2 δηλώνεται ότι θα παίρνει MaxLength+1 χαρακτήρες. Ο πρόσθετος χαρακτήρας είναι για το null, που προσθέτουν αυτόματα και οι δύο strcpy() και strncpy() στο τέλος της συμβολοσειράς.

Ασκήσεις αξιολόγησης

Άσκηση1

Τι είναι λάθος σε αυτό το τμήμα κώδικά;

```
unsigned  short SomeArray[5][4];
for (int i=0; i<4; i++)
      for (int j=0; j<5; j++)
            SomeArray[i][j] = i+j;
```

Άσκηση2

Δηλώστε ένα δισδιάστατο πίνακα που αντιπροσωπεύει ένα πίνακα του παιχνιδιού τρίλιζας.

Άσκηση3

Τι είναι λάθος σε αυτό το τμήμα κώδικά;

```
unsigned  short SomeArray[5][4];
for (int i=0; i<=5; i++)
      for (int j=0; j<=4; j++)
            SomeArray[i][j] = 0;
```

Κεφ.10 Λίστες

Σκοπός κεφαλαίου

Σε αυτό το κεφάλαιο αναλύουμε τις λίστες και τις διαδικασίες εισαγωγής, διαγραφής και της επανάληψης.

Λέξεις κλειδιά

Δήλωση λίστας

Εισαγωγή στοιχείων στη λίστα

Διαγραφή στοιχείων από τη λίστα

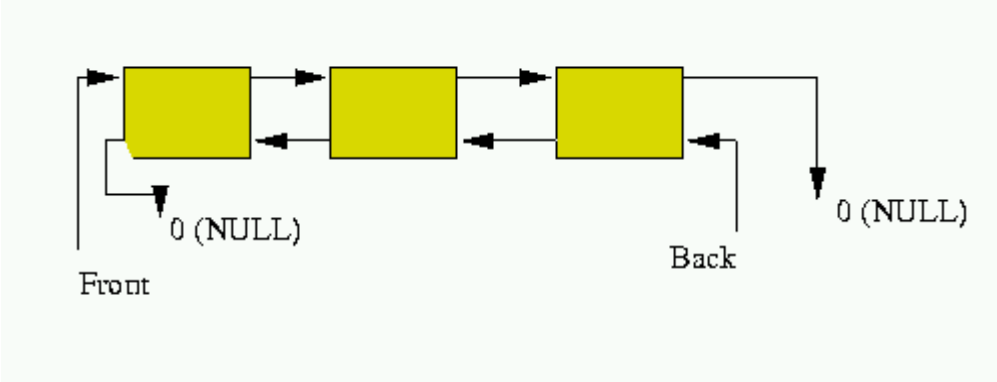
iterator

Συνδεμένη λίστα

Ορισμός

Μια λίστα είναι ένα κοντέινερ που έχει σχεδιαστεί να λειτουργεί βέλτιστα όταν εισάγετε και διαγράφετε στοιχεία συχνά κλάση κοντέινερ list του STL ορίζεται στο αρχείο επικεφαλίδας <list> στο namespace std. Η κλάση list υλοποιείται ως μια διπλή συνδεμένη λίστα, όπως κάθε κόμβος έχει συνδέσεις και στον προηγούμενο κόμβο και στον επόμενο κόμβο της λίστας.

Η κλάση list παίρνει όλες τις συναρτήσεις μέλη από την κλάση vector. Μπορείτε να διασχίσετε μια λίστα ακολουθώντας τις συνδέσεις που παρέχονται σε κάθε κόμβο. Γενικά, οι συνδέσεις υλοποιούνται χρησιμοποιώντας δείκτες. Η τυπική κλάση κοντέινερ list χρησιμοποιεί ένα μηχανισμό που ονομάζεται επαναλήπτης (iterator) (θα αναλυθεί παρακάτω) για τον ίδιο σκοπό.



Οι λίστες είναι μια ιδιομορφία των πινάκων. Οι mfc λίστες κλάσεων εφαρμόζουν τις συνδεμένες λίστες, που χρησιμοποιούν τους δείκτες για να συνδέσουν τα στοιχεία τους (αποκαλούμενα κόμβους) παρά ανάλογα με τις παρακείμενες θέσεις μνήμης με τις τιμές διαταγής, λίστες είναι μια καλύτερη δομή δεδομένων για να χρησιμοποιήσουν πότε πρέπει να είστε σε θέση να παρεμβάλετε και να διαγράψετε τα στοιχεία γρήγορα. Εντούτοις, η εύρεση των στοιχείων σε μια λίστα μπορεί να είναι πιο αργή από βρίσκοντας τα στοιχεία σε ένα πίνακα, επειδή μια λίστα πρέπει συχνά να διαπερνηθεί διαδοχικά προκειμένου να ακολουθηθούν οι δείκτες από ένα στοιχείο στο επόμενο.

Κατά τη χρησιμοποίηση των λιστών, πρέπει να ξέρετε κάποιο νέο λεξιλόγιο. Συγκεκριμένα, πρέπει να ξέρετε ότι το κεφάλι(head) ενός καταλόγου είναι ο πρώτος κόμβος στον κατάλογο, και η ουρά του καταλόγου είναι ο τελευταίος κόμβος στην λίστα. Κάθε κόμβος ξέρει πώς να φθάσει στον επόμενο κόμβο. Θα δείτε αυτούς τους όρους χρησιμοποιούμενους συχνά, δεδομένου ότι ερευνάτε τις κατηγορίες mfc λιστών.

Η Mfc παρέχει τρεις κατηγορίες λίστες κλάσεων που μπορείτε να χρησιμοποιήσετε για να δημιουργήσετε τις λίστες σας. Αυτές οι κατηγορίες είναι **CObList** (που αντιπροσωπεύουν μια λίστα αντικειμένων), **CPtrList** (που αντιπροσωπεύουν μια λίστα δεικτών) και **CStringList** (που αντιπροσωπεύουν μια

λίστα σειρών). Κάθε μια από αυτές τις κλάσεις έχει τις παρόμοιες λειτουργίες μελών, και οι κλάσεις διαφέρουν στον τύπο στοιχείων που μπορούν να φυλάξουν στις λίστες τους. Στον παρακάτω πίνακα παρουσιάζονται οι λίστες και περιγράφονται οι λειτουργίες των μελών από τις λίστες κλάσεων.

Λειτουργίες	**Περιγραφή**
AddHead()	Προσθέτει ένα κόμβο στο κεφάλι της λίστας, κάνοντας τον κόμβο το νέο κεφάλι.
AddTail()	προσθέτει έναν κόμβο στην ουρά της λίστας, κάνοντας τον κόμβο τη νέα ουρά.
Find()	Ψάχνει τη λίστα για να βρει διαδοχικά το δεδομένο δείκτη αντικειμένου. Επιστρέφει μια αξία ΘΕΣΗΣ.
FindIndex()	Ανιχνεύει τη λίστα διαδοχικά, σταματά στον κόμβο που υποδεικνύεται από το δεδομένο δείκτη. Επιστρέφει μια αξία ΘΕΣΗΣ για τον κόμβο.
GetAt()	Δίνει στον κόμβο μια συγκεκριμένη θέση
GetCount()	Δίνει τον αριθμό των κόμβων στη λίστα
GetHead()	Παίρνει την κεφαλή της λίστας
GetHeadPosition()	Παίρνει τη θέση της κεφαλής του κόμβου
GetNext()	Παίρνει τον επόμενο κόμβο της λίστα κατά την επανάληψη πέρα από μια λίστα.
GetPrev()	Παίρνει τον προηγούμενο κόμβο της λίστα κατά την επανάληψη πέρα από μια λίστα
GetTail()	Παίρνει τον κόμβο ουρών της λίστας
GetTailPosition()	Παίρνει τη θέση του κόμβου ουρών.
InsertAfter()	Εισάγει ένα νέο κόμβο μετά από τη συγκεκριμένη θέση.
InsertBefore()	Εισάγει ένα νέο κόμβο πριν από τη συγκεκριμένη θέση.
IsEmpty()	Επιστρέφει TRUE εάν η λίστα είναι κενή αλλιώς FALSE.
RemoveAll()	Αφαιρεί τους κόμβους όλης της λίστας.
RemoveAt()	Αφαιρεί μόνο τον κόμβο από τη λίστα.
RemoveHead()	Αφαιρεί την κεφαλή του κόμβου της λίστας.
RemoveTail()	Αφαιρεί την ουρά του κόμβου της λίστας.
SetAt()	Θέτει τον κόμβο στη συγκεκριμένη θέση.

Δηλώνοντας λίστες

Η έκφραση std::list είναι παρόμοια με τη vector ή με τη deque, εκτός του ότι το αντικείμενο της λίστα παρέχει μια μη τυχαία είσοδο. Ωστόσο, το std::list αντικείμενο είναι αποδοτικό να τοποθετήσει τα στοιχεία μέσα ή να αφαιρέσει τα στοιχεία οπουδήποτε από την ακολουθία. Επίσης η λίστα μπορεί να επαναταξινομηθεί μόνη της δυναμικά αν χρειαστεί. Μπορείτε να δημιουργήσετε μια λίστα με διάφορους τρόπους, όπως φαίνεται παρακάτω:

std::list <type> name;

std::list <type> name (size);

std::list <type> name (size, value);

std::list <type> name (mylist);

std::list <type> name (first, last);

Το πρώτο παράδειγμα δημιουργεί μια κενή λίστα ονομαζόμενη name, η οποία κρατά τα δεδομένα από το type.

Για παράδειγμα αν θέλεις να δημιουργήσεις μια λίστα για ακέραιους μπορείς να γράψεις:

list<int> intList;

Το δεύτερο παράδειγμα δημιουργεί μια λίστα με αρχικοποίηση το size, το τρίτο παράδειγμα δημιουργεί μια λίστα που πάλι αρχικοποιεί το size, όπου κάθε στοιχείο είναι αρχικοποιημένο στο value.Το τέταρτο χρησιμοποιεί το αντίγραφο κατασκευής, το οποίο φτιάχνει ένα αντικείμενο λίστας από την ίδια υπάρχουσα λίστα, mylist. Το τελευταίο παράδειγμα δημιουργεί μια λίστα από ένα εύρος στοιχείων, τα οποία διευκρινίζονται από τους επαναλήπτες first και last.

Δείτε το παρακάτω παράδειγμα:

```
1. # include <iostream>
2. # include <list>
3. int main()
4. {
5.     std::list<int> intList(10, 1);
6.     int x=0;
7.     std::list<int>::iterator iter;
8.     for (iter = intList.begin();
9.     iter !=intList.end(); iter++)
10.    {
```

```
11.         std::cout <<"Element #"<<x++<<": " <<*iter<<std::endl;
12.     }
13.     char response;
14. std::cin >> response;
15. return 0;
16. }
```

Έξοδος:

Element #0: 1
Element #1: 1
Element #2: 1
Element #3: 1
Element #4: 1
Element #5: 1
Element #6: 1
Element #7: 1
Element #8: 1
Element #9: 1

Ανάλυση:

Η γραμμή 2 περιέχει την κατάλληλη επικεφαλίδα για να δημιουργήσουμε μια λίστα. Το πρόγραμμα δημιουργεί 10 στοιχεία τα οποία αρχικοποιούνται στη τιμή 1. Στη γραμμή 5 αρχικοποιεί τη λίστα ώστε να περιέχει 10 στοιχεία και η αρίθμηση να ξεκινά από το 0 (γραμμή 6). Στη γραμμή 7 χρησιμοποιεί το iterator ο οποίος είναι ο λεγόμενος επαναληπτής, με τον οποίο θα ασχοληθούμε αργότερα αναλυτικότερα.

Εισαγωγή στοιχείων στη λίστα

Για να εισάγεις νέα στοιχεία μέσα στη λίστα, μπορείς να χρησιμοποιήσεις το ως μέλη συναρτήσεων το push_front(), push_back() ή insert(). Όλες οι διαδικασίες εισαγωγής είναι αποδοτικές. Στο παρακάτω παράδειγμα δημιουργούμε μια κενή λίστα με μια ακολουθία από τους χαρακτήρες J έως A και επιδεικνύει το περιεχόμενο.

```
1. # include <iostream>
2. # include <list>
3. int main()
4. {
5.     std::list<char> charList;
6.     int x=0;
7.     for (int i=0; i<10; ++i)
8.     charList.push_front(65 + i);
9.
10.    std::list<char>::iterator iter;
11.    for (iter = charList.begin();
12.    iter !=charList.end(); iter++)
13.    {
14.        std::cout <<"Element #"<<x++<<": " <<*iter<<std::endl;
15.    }
16.    char response;
17. std::cin >> response;
18. return 0;
19. }
```

Έξοδος:

Element #0: J

Element #1: I

Element #2: H

Element #3: G

Element #4: F

Element #5: E

Element #6: D

Element #7: C

Element #8: B

Element #9: A

Ανάλυση:

Μπορείτε να διευκρινίσετε ακριβώς πού να εισαχθούν τα νέα στοιχεία όταν καλείται τη συνάρτηση εισαγωγής, η οποία υποστηρίζει ως επιχειρήματα τη θέση στην οποία να αρχίσει την εισαγωγή, τον αριθμό των στοιχείων στην εισαγωγή και την αξία της εισαγωγής. Η γραμμή 2 περιέχει την κατάλληλη επικεφαλίδα για να δημιουργήσουμε μια λίστα. Το πρόγραμμα δημιουργεί 10 στοιχεία τα οποία αρχικοποιούνται και εισάγουν τα γράμματα από το A έως το J. Στις γραμμές 5-9 δημιουργεί και αρχικοποιεί τη λίστα.

Διαγραφή στοιχείων από τη λίστα

Για να διαγράψεις στοιχεία από τη λίστα μπορείς να χρησιμοποιήσεις τις pop_front(), pop_back(), erase() ή remove() συναρτήσεις μέλη. Το παράδειγμα που ακολουθεί περιγράφει τη διαδικασία διαγραφής στοιχείων.

```
1. # include <iostream>
2. # include <list>
3. int main()
4. {
5.     std::list<char> charList;
6.     for (int x=0; x<10; ++x)
7.     charList.push_back(65 + x);
8.
9.     std::cout <<"original list: ";
10.    std::list<char>::iterator iter;
11.    for (iter = charList.begin();
12.    iter !=charList.end(); iter++)
13.    {
14.        std::cout <<*iter;
15.    }
16.    std::cout << std::endl;
17.    charList.remove('E');
18.        std::cout <<"resultant list: ";
19.        for (iter = charList.begin();
20.        iter !=charList.end(); iter++)
```

```
21.         {
22.             std::cout <<*iter;
23.         }
24.         std::cout << std::endl;
25.     char response;
26. std::cin >> response;
27. return 0;
28. }
```

Έξοδος:

original list: ABCDEFGHIJ

resultant list: ABCDFGHIJ

Ανάλυση:

Στην αρχή δηλώνουμε τη λίστα και την αρχικοποιούμε. Στις γραμμές 9-17 παρουσιάζουμε το τι θα φαίνεται στη λίστα. Στη γραμμή 17 είναι η διαδικασία διαγραφής και αφαιρούμε το γράμμα Ε, στις επόμενες γραμμές επαναπροσδιορίζουμε τη λίστα εφόσον έχουμε διαγράψει ένα στοιχείο της.

Επανάληψη της λίστας (iterator)

Ένας iterator είναι μια γενίκευση ενός δείκτη και προσπαθεί να αποφύγει μερικούς από τους κινδύνους ενός δείκτη.

Συχνά, θα θελήσετε επαναλάβετε μια λίστα. Παραδείγματος χάριν, όπως συμβαίνει με τη λίστα, αν θελήστε να δείξετε τις τιμές σε κάθε κόμβο της λίστας, που αρχίζει από το κεφάλι της λίστας και λειτουργεί τον τρόπο σας στην ουρά. Η εφαρμογή λιστών κάνει ακριβώς αυτό με τη συνάρτηση OnDraw (), όπως φαίνεται παρακάτω:

```
1. #include <iostream>
2. #include <list>
3. using namespace std;
4.
5. typedef list <int> IntegerList;
6.
```

```
7.  int main()
8.  {
9.      IntegerList intList;
10.     for (int i=1; i<=10; ++i)
11.     intList.push_back(i*2);
12.
13.     for (IntegerList::const_iterator ci = intList.begin();
14.                 ci !=intList.end(); ++ci)
15.                 cout << *ci << " ";
16. char response;
17. std::cin >> response;
18. return 0;
19. }
```

Έξοδος :

2 4 6 8 10 12 14 16 18 20

Ανάλυση:

Στο παραπάνω παράδειγμα χρησιμοποιεί το πρότυπο λίστας του STL.Στη γραμμή 2 περιλαμβάνει το απαραίτητο αρχείο #include. Αυτό φέρνει τον κώδικα για το πρότυπο λίστας του STL. Στη γραμμή 5, βλέπετε τη χρήση του typedef. Σε αυτή την περίπτωση, αντί να χρησιμοποιηθεί το list <int> σε όλη τη λίστα, το typedef σας επιτρέπει να χρησιμοποιήσετε το IntegerList.

Στη γραμμή 9, ορίζεται το intList ως μια λίστα ακέραιων χρησιμοποιώντας το typedef που μόλις δημιουργήθηκε. Προστίθενται οι πρώτοι 10 θετικοί ζυγοί αριθμοί στη λίστα χρησιμοποιώντας τη συνάρτηση push_back() στις γραμμές 10 και 11.

Στις γραμμές 13-15, κάθε κόμβος της λίστας προσπελάζεται χρησιμοποιώντας ένα σταθερό iterator. Εάν θέλετε να αλλάξετε ένα κόμβο στον οποίο δείχνει ένα iterator, πρέπει, αντίθετα, να χρησιμοποιήσετε ένα μη- const iterator:

intList:: iterator

Η συνάρτηση μέλος begin() επιστρέφει ένα iterator που δείχνει στον πρώτο κόμβο της λίστας. Μπορεί να χρησιμοποιηθεί ο τελεστής αύξησης ++ για να κάνει ένα iterator να δείχνει στον επόμενο κόμβο. Η συνάρτηση μέλος end() επιστρέφει ένα

iterator που δείχνει σε ένα κόμβο μετά τον τελευταίο της λίστας. Πρέπει να είστε σίγουροι ότι το iterator δεν θα φτάσει στο end()!.

Η αποαναφορά ενός iterator γίνεται όπως και ενός δείκτη, για να επιστρέψει τον κόμβο στον οποίο έδειχνε, όπως φαίνεται στη γραμμή 15.

Συνδεμένη λίστα ή Συνδεδεμένη λίστα

Μια συνδεμένη λίστα είναι μια δομή δεδομένων που αποτελείται από μικρά κοντέινερ που έχουν σχεδιαστεί να συνδέονται μεταξύ τους, ανάλογα με τις απαιτήσεις. Η ιδέα είναι να γράψετε μια κλάση που να περιέχει ένα αντικείμενο από τα δεδομένα σας, που να μπορεί να δείχνει στο κοντέινερ του ίδιου τύπου. Δημιουργείτε ένα κοντέινερ για κάθε αντικείμενο που θέλετε να αποθηκεύσετε και να συνδέετε μεταξύ όπως απαιτείται.

Τα κοντέινερ ονομάζονται κόμβοι (nodes). Ο πρώτος κόμβος της λίστας ονομάζεται κεφαλή (head) και ο τελευταίος κόμβος της λίστας ονομάζεται ουρά (tail).

Οι λίστες έχουν τρεις βασικές μορφές:

- Απλά συνδεμένες λίστες(singly linked)
- Διπλά συνδεμένες λίστες (doubly linked)
- Δέντρα (trees)

Σε μια απλά συνδεμένη λίστα, κάθε κόμβος δείχνει προς τα εμπρός στον επόμενο κόμβο, αλλά όχι και προς τα πίσω. Για να βρείτε ένα συγκεκριμένο κόμβο, αρχίζετε από την κορυφή και πηγαίνετε από κόμβο σε κόμβο.

Μια διπλά συνδεμένη λίστα σας επιτρέπει να μετακινείστε και προς τα πίσω και προς τα εμπρός στην αλυσίδα.

Ένα δέντρο είναι μια σύνθετη δομή που αποτελείται από κόμβους, όπου κάθε ένας μπορεί να δείχνει σε δύο ή σε περισσότερες κατευθύνσεις.

Ασκήσεις αξιολόγησης

Άσκηση1

Δηλώστε τρία αντικείμενα λίστας: μια λίστα από String, μια λίστα από Cat και μια λίστα από int.

Άσκηση 2

Δημιουργήστε μια κενή λίστα με μια ακολουθία από χαρακτήρες A έως F. Έπειτα διαγράψτε το χαρακτήρα E.

Κεφ.11 Στοίβες

Σκοπός κεφαλαίου

Στο κεφάλαιο αυτό αναφερόμαστε τη στοίβα και αναλύουμε τις διαδικασίες της εισαγωγής, εξαγωγής και την αναδρομή.

Λέξεις κλειδιά

FIFO (last-in, first-out)
Εισαγωγή-ώθηση (push)
Εξαγωγή – Απώθηση (pop)
Αναδρομή

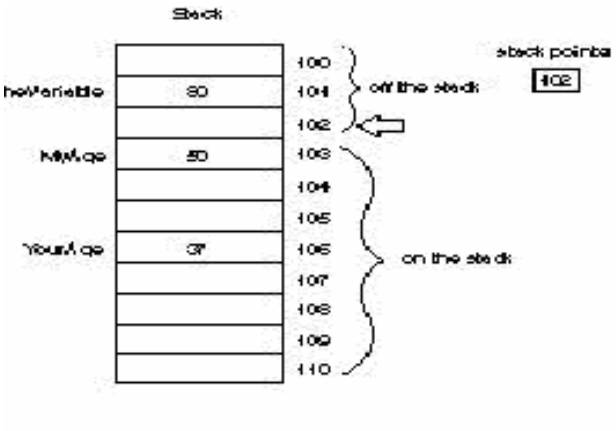
Ορισμός

Μια από τις πιο συχνά χρησιμοποιημένες δομές δεδομένων στον προγραμματισμό είναι η **στοίβα** (stack).

Η στοίβα είναι μια ειδική περιοχή της μνήμης που δεσμεύεται από το πρόγραμμά σας για να κρατά τα δεδομένα που απαιτούνται από κάθε μια από τις συναρτήσεις του προγράμματος. Ονομάζεται στοίβα επειδή είναι μια ουρά αναμονής **FIFO (last-in, first-out)**.

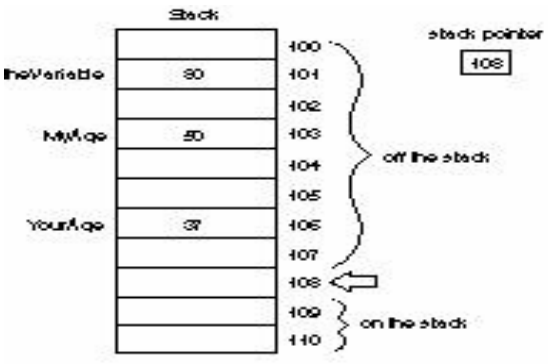
Το last-in, first-out σημαίνει ότι οτιδήποτε προστίθεται τελευταίο στη στοίβα είναι το πρώτο πράγμα που θα βγει. Όταν ωθούνται (**push**) τα δεδομένα στη στοίβα, η στοίβα μεγαλώνει. Καθώς βγαίνουν (**pop**) τα δεδομένα από τη στοίβα, η στοίβα μικραίνει.

Η στοίβα θα μπορούσαμε να πούμε ότι είναι ένα ντουλάπι με συρτάρια. Κάθε ένα από τα συρτάρια έχει μια συνεχόμενη διεύθυνση και μια από αυτές τις διευθύνσεις κρατιέται στον καταχωρητή του δείκτη της στοίβας. Τα πάντα που είναι κάτω από αυτή τη μαγική διεύθυνση, που είναι γνωστή ως κορυφή της στοίβας, θεωρούνται ότι ανήκουν στη στοίβα. Ότι είναι πάνω από την κορυφή της στοίβας θεωρείται ότι είναι εκτός της στοίβας και είναι άκυρο. Η εικόνα επεξηγεί αυτή την ιδέα.



Όταν τα δεδομένα μπουν μέσα στη στοίβα, τοποθετούνται σε ένα συρτάρι επάνω από το δείκτη της στοίβας και έπειτα ο δείκτης της στοίβας μετακινείται στα ίδια δεδομένα. Όταν τα δεδομένα βγαίνουν από τη στοίβα, αυτό που συμβαίνει πραγματικά είναι ότι η διεύθυνση του δείκτη της στοίβας αλλάζει και μετακινείται προς τα κάτω στη στοίβα.

Η παρακάτω εικόνα κάνει σαφή αυτόν τον κανόνα.



Τα δεδομένα πάνω από τον δείκτη της στοίβας (έξω από τη στοίβα) μπορεί να αλλάξουν οποιαδήποτε στιγμή. Αυτές οι τιμές τους αναφέρονται ως «σκουπίδια» επειδή η τιμή τους δεν είναι πλέον αξιόπιστη.

Η στοίβα δεν υλοποιείται ως μια ανεξάρτητη κλάση κοντέινερ, αντίθετα υλοποιείται ως κάτι γενικότερο ενός κοντέινερ. Η κλάση προτύπου stack ορίζεται στο αρχείο επικεφαλίδας <stack> του namespace std. Στη πραγματικότητα, μπορεί να χρησιμοποιηθεί οποιοδήποτε σειριακό κοντέινερ υποστηρίζει τις λειτουργίες back(), push_back() και pop_back() για χειρισμό μια στοίβας.

Η κλάση προτύπου «stack» του STL, έχει σχεδιαστεί για να περιέχει οποιονδήποτε τύπο αντικειμένων. Ο μόνος περιορισμός είναι ότι όλα τα στοιχεία πρέπει να είναι του ίδιου τύπου.

Σχεδιάζοντας μια στοίβα

Αρχίζετε να σχεδιάζεται μια στοίβα από τη σχεδίαση της δομή δεδομένων. Αυτή η δομή θα χρειαστεί μια θέση για να βάλει τα δεδομένα και μια αρίθμηση του αριθμού στοιχείων που θα ωθούνται αυτήν την στιγμή στη στοίβα.

Δείτε το παρακάτω:

```
const int STACK_SIZE = 100;
struct stack
{
int count;
int data[STACK_SIZE];
};
```

Έπειτα πρέπει να δημιουργήσετε τις διαδικασίες push και pop. Οι συναρτήσεις push αποθηκεύουν το στοιχείο στη στοίβα και αυξάνουν έπειτα την αρίθμηση στοιχείων.

Το πρότυπο στοίβα

Όσοι προγραμματίζουν για πολύ καιρό θα γνωρίζουν ότι η στοίβα είναι μια τάξη που εφαρμόζει τη διαδικασία το πρώτο που μπαίνει είναι το τελευταίο που βγαίνει. Η στοίβα είναι ένα πολύ χρήσιμο εργαλείο των δομών δεδομένων. Η στοίβα εφαρμόζεται στο STL όχι από ένα νέο πρότυπο κατηγορίας κοντέινερ όπως

αναμένετε, αλλά σαν έναν προσαρμοστή κοντέινερ όπως ονομάζεται, std::stack. Μπορεί να χρησιμοποιήσετε τον κοντέινερ στοίβας std::stack για να δημιουργήσετε στοίβες από std::vector, std::deque, ή std::list αντικείμενα.

Δημιουργείτε το std:: stack όπως φαίνεται παρακάτω:

std::stack <type, container > name;

Ο **type** παράμετρος είναι ένας τύπος δεδομένων που χρειάζεται η στοίβα και ο **container** είναι ένας τύπος κοντέινερ που χρησιμοποιεί η στοίβα σαν εργαλείο. Για παράδειγμα, αν θέλετε να δημιουργήσετε ένα std::stack αντικείμενο για ακέραιους βασισμένο στο std::list αντικείμενο μπορείτε να γράψετε :

std::stack <int, std::list<int> > intStack;

Δείτε το παρακάτω:

```
1.  # include <iostream>
2.  # include <list>
3.  # include <stack>
4.
5.  int main()
6.  {
7.      std::stack<int, std::list<int> > intStack;
8.      std::cout <<"Values pushed onto stack: " <<std::endl;
9.      for (int x=1; x<11; ++x)
10.     {
11.         intStack.push(x*100);
12.         std::cout <<x*100 <<std::endl;
13.     }
14.     std::cout<<"Values popped from stack: " <<std::endl;
15.     int size = intStack.size();
16.     for (int x=0; x<size; ++x)
17.     {
18.         std::cout<<intStack.top() <<std::endl;
19.         intStack.pop();
20.     }
```

```
21.     char response;
22. std::cin >> response;
23. return 0;
24. }
```

Έξοδος :

Values pushed onto stack:
100
200
300
400
500
600
700
800
900
1000
Values popped from stack:
1000
900
800
700
600
500
400
300
200
100

Ανάλυση :

Στη γραμμή 7 δηλώνουμε τη στοίβα σαν ακέραιο βασισμένο σε αντικείμενο της λίστας. Στη γραμμή 9 έχουμε σαν περιορισμό ότι ο ακέραιος αρχίζει από το 1, είναι μικρότερο από το 11 και ότι ο ακέραιος αυξάνεται. Στη γραμμή 11 γίνεται η

διαδικασία της εισαγωγής (push) και τον ακέραιο τον πολλαπλασιάζουμε με 100. Στη συνέχεια εφαρμόζουμε τα ίδια, όμως με τη διαδικασία της αφαίρεσης(pop).

Σημείωση: Επειδή η στοίβα είναι μια πολύ απλή δομή δεδομένων, απαιτεί μόνο βασικές διαδικασίες. Γι'αυτό το λόγο η std::stack κοντέινερ καθορίζει μόνο το empty(). Size(), top(), push() και pop() συναρτήσεις.

Πράξεις που μπορούμε να εφαρμόσουμε στις στοίβες

Οι βασικές πράξεις που μπορούμε να εφαρμόσουμε στη στοίβα είναι η **εισαγωγή αντικειμένου** και η **διαγραφή αντικειμένου** που βρίσκεται στην κορυφή του. Άλλες χρήσιμες πράξεις είναι η **επιλογή του αντικειμένου** που βρίσκεται στην κορυφή χωρίς να μετακινηθεί από αυτόν, το **καθάρισμα από τα περιεχόμενά της**, και ο **έλεγχος κενότητας**.

- **Εισαγωγή νέου στοιχείου**

Η εφαρμογή νέου στοιχείου λέγεται **push**. Πριν προσθέσουμε το νέο αντικείμενο θα πρέπει να ελέγξουμε αν υπάρχει διαθέσιμος χώρος στη στοίβα. Αυτό γίνεται συγκρίνοντας την τιμή του δείκτη TOS με την τιμή της μεταβλητής MAX η οποία αντιπροσωπεύει το μέγιστο αριθμό αντικειμένων που μπορεί να χωρέσει ο πίνακας ITEMS. Αν τα περιεχόμενα των μεταβλητών TOS και MAX είναι ίσα και προσπαθήσουμε να εισάγουμε το νέο αντικείμενο, παραβιάζουμε το μέγεθος του πίνακα ITEMS και προκαλούμε τη συνθήκη **υπερχείλισης**.

```
1. void push(pinakas& items, int& tos, string x)
2. {
3.     if(tos==max)
4.     cout<<"η στοίβα είναι πλήρης  \n";
5.     else
6.     {
7.        tos=tos+1;
8.        stpcpy(items[tos],x);
9.     };
10. };
```

Ø Διαγραφή αντικειμένου

Η πράξη της διαγραφής λέγεται **pop**. Το μόνο στοιχείο που μπορεί να διαγραφεί είναι αυτό που βρίσκεται στην κορυφή. Πριν ξεκινήσουμε τη διαδικασία της διαγραφής πρέπει να ελέγξουμε αν η στοίβα είναι κενή. Αυτό γίνεται με απλή σύγκριση της τιμής του δείκτη TOS με το μηδέν 0. Αν είναι μηδέν και προσπαθήσουμε να το διαγράψουμε, παραβιάζουμε τον πίνακα και προκαλούμε συνθήκη υπερχείλισης. Αν η τιμή του TOS είναι μεγαλύτερη από μηδέν, τότε εξάγουμε το στοιχείο από τη στοίβα και στη συνέχεια μειώνουμε την αξία του TOS κατά 1.

```
1. void pop(pinakas& items, int& tos, string x)
2. {
3.     if(tos<0)
4.     cout<<"η στοίβα είναι άδεια \n";
5.     else
6.     {
7.        stpcpy(x,items[tos]);
8.        tos=tos-1;
9.     };
10. };
```

Η στοίβα και οι συναρτήσεις

Τα παρακάτω είναι μια προσέγγιση αυτών που συμβαίνουν όταν διακλαδώνεται το πρόγραμμά σας σε μια συνάρτηση. (Οι λεπτομέρειες μπορεί να διαφέρουν ανάλογα με το λειτουργικό σας σύστημα και τον μεταγλωττιστή.)

1. Η διεύθυνση του δείκτη εντολών αυξάνεται για να αποκτήσει τη διεύθυνση της επόμενης εντολής μετά από την κλήση της συνάρτησης. Αυτή η διεύθυνση τοποθετείται μέσα στην στοίβα και θα είναι η διεύθυνση επιστροφής όταν τελειώσει η συνάρτηση.
2. Γίνεται χώρος στη στοίβα για τον τύπο επιστροφής που έχετε δηλώσει. Σε ένα σύστημα με ακέραιους των δύο byte, εάν ο τύπος επιστροφής είναι int, προστίθενται άλλα δύο byte στη στοίβα, αλλά δεν τοποθετείται καμιά τιμή σε αυτά τα byte.

3. Η διεύθυνση της κληθείσας συνάρτησης, που κρατιέται σε μια ειδική περιοχή της μνήμης που δεσμεύεται για αυτό το σκοπό, φορτώνεται στον δείκτη εντολών, έτσι η επόμενη εντολή που εκτελείται θα είναι στην κληθείσα συνάρτηση.
4. Η τρέχουσα κορυφή της στοίβας σημειώνεται τώρα και κρατιέται σε ένα ειδικό δείκτη που ονομάζεται πλαίσιο στοίβας (stack frame). Όσα προστίθενται στη στοίβα από τώρα μέχρι τη στιγμή επιστροφής της συνάρτησης, θεωρούνται «τοπικά» για τη συνάρτηση.
5. Όλα τα ορίσματα της συνάρτησης τοποθετούνται στη στοίβα.
6. Εκτελείται η εντολή που είναι τώρα στο δείκτη εντολών, οπότε εκτελείται η πρώτη εντολή της συνάρτησης.
7. Οι τοπικές μεταβλητές ωθούνται στη στοίβα όταν ορίζονται.

Όταν η συνάρτηση είναι έτοιμη να επιστρέψει, η τιμή επιστροφής τοποθετείται στην περιοχή της στοίβας που δεσμεύθηκε στο βήμα 2. Η στοίβα αδειάζει έπειτα μέχρι το δείκτη πλαισίου της στοίβας, ο οποίος πετάει όλες τις τοπικές μεταβλητές και τα ορίσματα της συνάρτησης.

Η τιμή επιστροφής βγαίνει από τη στοίβα και εκχωρείται ως τιμή στην ίδια τη κλήση της συνάρτησης και η διεύθυνση που κρατήθηκε στο βήμα 1 ανακτάται και μπαίνει στον δείκτη εντολών. Το πρόγραμμα συνεχίζει συνεπώς από τη θέση που είναι αμέσως μετά την κλήση της συνάρτησης, έχοντας την τιμή της συνάρτησης.

Η στοίβα καθαρίζεται αυτόματα όταν επιστρέφει μια συνάρτηση. Όλες οι τοπικές μεταβλητές βγαίνουν εκτός πεδίου δράσης και αφαιρούνται από τη στοίβα. Ο ελεύθερος χώρος δεν καθαρίζεται ως ότου τελειώσει το πρόγραμμά σας και είναι δική σας ευθύνη να ελευθερώσετε οποιαδήποτε μνήμη έχετε δεσμεύσει, όταν δεν την χρειάζεστε πλέον. Εκεί είναι απόλυτα απαραίτητες οι συναρτήσεις αποδιάρθρωσης, επειδή παρέχουν ένα μέρος όπου μπορεί να ελευθερωθεί μνήμη από το σωρό που είναι δεσμευμένη σε μια κλάση.

Εφαρμογές και χρήση της στοίβας

Οι στοίβες βρίσκουν μεγάλη χρήση στην επιστήμη των υπολογιστών. Οι πλέον γνωστές χρήσεις των στιβάδων στους Η/Υ είναι οι ακόλουθες:

- Υπολογισμός αριθμητικών παραστάσεων

- Αποθήκευση της διεύθυνσης επιστροφής κατά την κλήση υποπρογραμμάτων
- **Αναδρομή**, αναδρομικά λέγονται τα υποπρογράμματα που έχουν την ιδιότητα να καλούν τον εαυτό τους. Σε κάθε αναδρομική διαδικασία, πριν από κάθε κλήση του εαυτού της φυλάσσονται σε μια στοίβα οι τιμές όλων των χρησιμοποιημένων μεταβλητών. Όταν η διαδικασία εξέλθει από το φυσιολογικό τέλος τότε ανασύρονται όλες οι τιμές από τη στοίβα και συνεχίζεται η λειτουργία της από το σημείο που κλήθηκε ο εαυτός της και κάτω. Η διαδικασία τερματίζεται οριστικά όταν η στοίβα είναι άδεια

  Τα πλεονεκτήματα της αναδρομής είναι ότι πολλά προβλήματα στην επιστήμη των Η/Υ προσδιορίζονται με όρους επαναλαμβανόμενων σχέσεων, τα οποία συνήθως μπορούν να επιλυθούν με απλές και «κομψές» αναδρομικές λύσεις. Το μειονέκτημά της είναι ότι απαιτεί περισσότερη μνήμη από μια μη αναδρομική λύση του ίδιου προβλήματος.

Ασκήσεις αξιολόγησης

Άσκηση 1

Να γραφτεί πρόγραμμα που να καθαρίζει μια στοίβα από τα περιεχόμενά της.

Άσκηση 2

Να δημιουργήσετε μια στοίβα που να εμφανίζει τους αριθμούς από το1-11.Στη συνέχεια να αφαιρέσετε το στοιχείο 11 και στη στοίβα να φαίνεται από το 0-5.Ποιά θα είναι η έξοδος;

Κεφ.12 Ουρές

Σκοπός κεφαλαίου

Στο κεφάλαιο αυτό θα ασχοληθούμε με τις ουρές και τις ουρές προτεραιότητας. Επίσης θα μελετήσουμε τις διαδικασίες εισαγωγής και εξαγωγής στοιχείων, καθώς και τη χρήση των ουρών.

Λέξεις κλειδιά

FIFO(first in, first out)
Ουρά προτεραιότητας
Εισαγωγής (push)
Εξαγωγή (pop)
Εξομοίωση

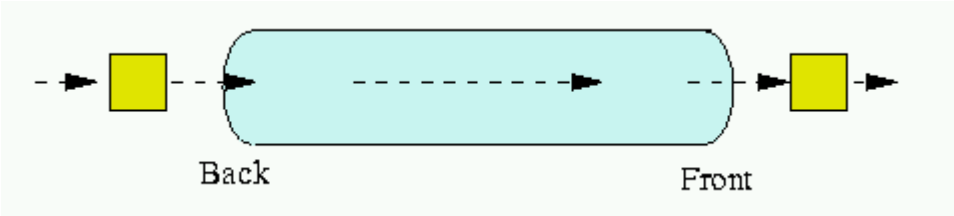
Ορισμός

Μια άλλη ενδιαφέρουσα υποπερίπτωση λίστας επιτρέπει την εισαγωγή στοιχείων από το ένα άκρο και την εξαγωγή των στοιχείων από το άλλο. Η επεξεργασία των πληροφοριών σε μια τέτοια δομή, γίνεται με την ακριβώς σειρά με την οποία μπήκαν οι πληροφορίες στη λίστα. Για αυτό το λόγο ονομάζεται ουρά.

Μια ουρά είναι μια άλλη συχνά χρησιμοποιημένη δομή δεδομένων στον προγραμματισμό. Τα στοιχεία προστίθενται στην ουρά από τη μια πλευρά και

αφαιρούνται από την άλλη. Η ουρά είναι μια δομή **FIFO** (first in, first out – το πρώτο που μπαίνει είναι το πρώτο που βγαίνει) ενώ μια στοίβα είναι LIFO (last in, first out- το τελευταίο που μπαίνει είναι το πρώτο που βγαίνει).

Όπως το stack, έτσι και το queue υλοποιείται ως μια γενικότερη κλάση ενός κοντέινερ. Το κοντέινερ πρέπει να υποστηρίζει τις λειτουργίες front(), back(), push_back() και pop_ front ().



Το παραπάνω γράφημα αποδεικνύει ότι μια σειρά αναμονής έχει ένα σημείο όπου τα στοιχεία μπορούν να προστεθούν στη σειρά αναμονής, και ένα σημείο όπου τα στοιχεία μπορούν να αφαιρεθούν από τη σειρά αναμονής.

Δηλώνοντας ουρές

Οι ουρές χρησιμοποιούν σαν εργαλείο για να τις αρχικοποιήσουν το std::queue. Μπορείς να χρησιμοποιήσεις το std::queue κοντέινερ για να δημιουργήσεις ουρές από std::deque ή std::list. Η κατασκευή του std::queue αντικειμένου φαίνεται παρακάτω:

std::queue <type, container > name;

Ο **type** παράμετρος είναι ένας τύπος δεδομένων που χρειάζεται η ουρά και ο **container** είναι ένας τύπος κοντέινερ που χρησιμοποιεί η ουρά σαν εργαλείο, είναι ακριβώς η ίδια λογική με τη στοίβα. Για παράδειγμα, αν θέλετε να δημιουργήσετε ένα std:: queue αντικείμενο για ακέραιους βασισμένο στο std::list αντικείμενο μπορείτε να γράψετε :

std:: queue <int, std::list<int> > intQueue;

Δείτε το παρακάτω:

1. # include <iostream>
2. # include <list>
3. # include <queue>

```
4.
5.  int main()
6.  {
7.      std::queue<int, std::list<int> > intQueue;
8.      std::cout <<"Values pushed onto queue: " <<std::endl;
9.      for (int x=1; x<11; ++x)
10.     {
11.         intQueue.push(x*100);
12.         std::cout <<x*100 <<std::endl;
13.     }
14.     std::cout<<"Values removed from queue: " <<std::endl;
15.     int size = intQueue.size();
16.     for (int x=0; x<size; ++x)
17.     {
18.         std::cout<<intQueue.front() <<std::endl;
19.         intQueue.pop();
20.     }
21.      char response;
22. std::cin >> response;
23. return 0;
24. }
```

Έξοδος :

```
Values pushed onto queue:
100
200
300
400
500
600
700
800
900
1000
```

Values removed from queue:
100
200
300
400
500
600
700
800
900
1000

Ανάλυση :

Παρατηρούμε ότι οι διαδικασίες είναι σχεδόν ίδιες και με τη στοίβα.

Στη γραμμή 7 δηλώνουμε την ουρά σαν ακέραιο βασισμένο σε αντικείμενο της λίστας. Στη γραμμή 9 έχουμε σαν περιορισμό ότι ο ακέραιος αρχίζει από το 1, είναι μικρότερο από το 11 και ότι ο ακέραιος αυξάνεται. Στη γραμμή 11 γίνεται η διαδικασία της εισαγωγής (push) και τον ακέραιο τον πολλαπλασιάζουμε με 100.Στη συνέχεια εφαρμόζουμε τα ίδια, όμως με τη διαδικασία της αφαίρεσης(pop).

Σημείωση: Επειδή όπως η στοίβα, έτσι και οι ουρές είναι μια πολύ απλή δομή δεδομένων, απαιτεί μόνο βασικές διαδικασίες. Γι'αυτό το λόγο η std::queue κοντέινερ καθορίζει μόνο το empty(). size(), front(), back(), push() και pop() συναρτήσεις.

Ουρά προτεραιότητας (std::priority_queue)

Η ουρά προτεραιότητας είναι μια δομή δεδομένων η οποία αφαιρεί (pop) τα στοιχεία από την ακολουθία σε λειτουργία προτεραιότητας. Η προτεραιότητα είναι βασισμένη στην παρεχόμεμενη σύγκριση συναρτήσεων (ονομάζεται predicate). Για παράδειγμα αν θέλεις να χρησιμοποιήσεις την προκαθορισμένη std::less<> βεβαιώστε κάθε φορά που προσθέτετε ή αφαιρείτε την αξία από τη ουρά προτεραιότητας, το περιεχόμενο τακτοποιείται με την φθίνουσα σειρά. Αυτό δίνει στην αξία με τη μεγαλύτερη τιμή την υψηλότερη προτεραιότητα. Για να δηλώσετε μια ουρά προτεραιότητας μπορείτε να γράψετε std::priority_queue.

Μπορείς να δημιουργήσεις ένα std:: queue αντικείμενο όπως φαίνεται παρακάτω:

std:: queue <type, container, predicate > name;

Η παράμετρος **type** είναι ένας τύπος δεδομένων που χειρίζεται η ουρά προτεραιότητας και ο **container** είναι ένας τύπος κοντέινερ που χρησιμοποιεί η ουρά προτεραιότητας σαν εργαλείο. Η παράμετρος **predicate** είναι μια συνάρτηση σύγκρισης που χρησιμοποιείται για να καθορίσει την προτεραιότητα των στοιχείων στην ουρά προτεραιότητας.

Για παράδειγμα αν θέλεις να δημιουργήσεις ένα std::queue αντικείμενο για ακέραιους βασισμένο σε ένα std::vector αντικείμενο και χρησιμοποιώντας την προκαθορισμένη προτεραιότητα, μπορείτε να γράψετε:

std::queue<int, std::vector<int> >intQueue;

Δείτε το παρακάτω παράδειγμα

```
1.  # include <iostream>
2.  # include <list>
3.  # include <queue>
4.
5.  int main()
6.  {
7.      std::priority_queue<int, std::vector<int> >intPQueue;
8.      intPQueue.push(400);
9.      intPQueue.push(100);
10.     intPQueue.push(500);
11.     intPQueue.push(300);
12.     intPQueue.push(200);
13.
14.     std::cout << "Value removed from priority queue: " <<std::endl;
15.     int size = intPQueue.size();
16.     for (int x=0; x<size; ++x)
17.     {
18.         std::cout << intPQueue.top() <<std::endl;
19.         intPQueue.pop();
20.     }
```

```
21.     char response;
22. std::cin >> response;
23. return 0;
24. }
```

Έξοδος :

Value removed from priority queue:
500
400
300
200
100

Ανάλυση :

Επειδή το πρόγραμμα καθορίζει την ουρά προτεραιότητας χρησιμοποιώντας την εξ' ορισμού std::less<> παρεχόμενη σύγκριση, οι τιμές αφαιρούνται από την ακολουθία σε λειτουργία από το μεγαλύτερο στο μικρότερο. Μπορείς να δημιουργήσεις μια ουρά προτεραιότητας η οποία θα οργανώνει το περιεχόμενό του βασισμένο σε άλλες προτεραιότητες. Για παράδειγμα αν στο πρόγραμμά σου χρησιμοποιήσεις το std::greater<>, όταν καθορίζεις την ουρά προτεραιότητας, η αξία της πιο υψηλής προτεραιότητας είναι η μικρότερη επειδή οι τιμές τοποθετούνται με αύξουσα σειρά. Για να αλλάξεις τις τιμές από φθίνουσα σε αύξουσα θα πρέπει να αντικατασταθεί η γραμμή :

std::priority_queue<int, std::vector<int> >intPQueue;

και να γίνει

std::priority_queue<int, std::vector <int> std::greater <int> >intPQueue;

Τότε η έξοδος θα είναι ως εξής :

Value removed from priority queue:
100
200
300
400
500

Ουρές και ουρές προτεραιότητας

Μια αφαίρεση σειρών αναμονής εκθέτει μια πολιτική αποθήκευσης FIFO και ανάκτησης. Το επόμενο αντικείμενο που ανακτάται λαμβάνεται από το μέτωπο της σειράς αναμονής

Υπάρχουν δύο γεύσεις των σειρών αναμονής που παρέχονται από τη standard library, την ουρά και την ουρά προτεραιότητας.

Μια ουρά προτεραιότητας επιτρέπει στο χρήστη για να καθιερώσει μια προτεραιότητα μεταξύ των στοιχείων που κρατούνται με στη ουρά. Παρά να τοποθετηθεί ένα πρόσφατα εισαγμένο στοιχείο στο πίσω μέρος της ουράς, το στοιχείο τοποθετείται μπροστά από όλα εκείνα τα στοιχεία με μια χαμηλότερη προτεραιότητα.

Ο χρήστης που καθορίζει την ουρά προτεραιότητας καθορίζει πώς εκείνη η προτεραιότητα πρόκειται να αποφασιστεί.

Ένα παράδειγμα ουρά προτεραιότητας είναι ο έλεγχος γραμμών στις αποσκευές. Εκείνοι των οποίων η πτήση πρόκειται να φύγει σε 15 λεπτά κινούνται γενικά προς το μέτωπο της γραμμής έτσι ώστε αυτοί να κάνουν τσεκ προτού φύγει το αεροπλάνο.

Ο παρακάτω πίνακας δείχνει ένα πλήρες σύνολο διαδικασιών που υποστηρίζονται από τις ουρές και τις ουρές προτεραιότητας.

Συναρτήσεις	**Ανάλυση**
Empty()	Επιστρέφει αληθές αν η ουρά είναι άδεια, διαφορετικά ψευδής.
Size()	Επιστρέφει την αρίθμηση του αριθμού των στοιχείων της ουράς.
Pop()	Διαγράφει, αλλά δεν επιστρέφει, το μπροστινό στοιχείο της ουράς. Στην περίπτωση της ουράς προτεραιότητας, το μπροστινό στοιχείο αντικαθίσταται από το στοιχείο με την μεγαλύτερη προτεραιότητα
Front()	Επιστρέφει, αλλά δεν διαγράφει, το μπροστινό στοιχείο της ουράς. Αυτό μπορεί να εφαρμοστεί μόνο στην ουρά.
Back()	Επιστρέφει, αλλά δεν διαγράφει, το τελευταίο στοιχείο της ουράς. Αυτό μπορεί να εφαρμοστεί μόνο στην ουρά.
Top()	Επιστρέφει, αλλά δεν διαγράφει, το στοιχείο με την μεγαλύτερη προτεραιότητα. Αυτό μπορεί να εφαρμοστεί μόνο στην ουρά προτεραιότητας.
Push(item)	Τοποθετεί ένα νέο στοιχείο στο τέλος της ουράς. Στην περίπτωση της ουράς προτεραιότητας, το τοποθετεί ανάλογα με την προτεραιότητα

Χρήση των ουρών

Η χρήση των ουρών απαντάται σε ποικίλες περιοχές ενδιαφέροντος. Ένα ειδικό παράδειγμα χρησιμοποίησης των ουρών συναντάμε στα συστήματα Η/Υ κατανεμημένου χρόνου, όπου πολλοί χρήστες μοιράζονται ταυτόχρονα ένα σύστημα Η/Υ. Τα προγράμματα των χρηστών που αναμένουν να υποστούν επεξεργασία, σχηματίζουν μια ουρά αναμονής. Αυτή η ουρά πιθανόν να μη λειτουργεί επάνω στο first in, first out, αλλά σε ένα πιο πολύπλοκο σύστημα προτεραιότητας, λεγόμενο ουρά προτεραιότητας.

Μια από τις πλέον κλασικές περιοχές στην οποία γίνεται εκτεταμένη χρήση των ουρών είναι η **εξομοίωση**. Εξομοίωση είναι η διαδικασία σχηματισμού ενός γενικευμένου μοντέλου από μια πραγματική κατάσταση, το οποίο αποτελεί αντικείμενο διερεύνησης του αποτελέσματος που θα έχει η παραδοχή διαφόρων υποθέσεων ή η υιοθέτηση διαφόρων στρατηγικών επί της καταστάσεως. Ο κύριος σκοπός ενός προγράμματος εξομοίωσης είναι να βοηθήσει το χρήστη, να διαμορφώσει γνώμη περί του τι θα συμβεί σε μια πραγματική κατάσταση κάτω από ορισμένες προϋποθέσεις. Το κύριο πλεονέκτημα είναι ότι επιτρέπει τον πειραματισμό χωρίς την ανάγκη τροποποίησης της πραγματικής καταστάσεως. Το μειονέκτημα είναι ότι μεγάλες και λεπτομερείς εξομοιώσεις είναι δαπανηρές.

Πολλά παραδείγματα εφαρμογής της εξομοίωσης είναι ο έλεγχος της τροχαίας κυκλοφορίας, τον παραγραμματισμό και έλεγχο παραγωγής βιομηχανικών προϊόντων, Management Information System,Command and Control Information System κτλ.

Ασκήσεις αξιολόγησης

Άσκηση 1

Γράψτε μια ουρά που να εμφανίζει ακέραιους βασισμένο σε λίστα και να εμφανίζονται οι αριθμοί από το 1-20.

Άσκηση 2

Να δημιουργήσετε μια ουρά που να εμφανίζει τους αριθμούς από το1-14.Στη συνέχεια να αφαιρέσετε το στοιχείο 1 και στην ουρά να φαίνεται από το 0-8. (εισαγωγή-εξαγωγή)

Απαντήσεις ασκήσεις αξιολόγησης

Κεφ.1 «Ανατομία προγράμματος c++»

Άσκηση 1

Το παρακάτω είναι μια πιθανή απάντηση:

```
10. #include <iostream>
11.
12. int main()
13. {
14.     std::cout<< "Run C++ \n";
15.     char response;
16. std::cin >> response;
17. return 0;
18. }
```

Άσκηση 2

Το h δεν χρειάζεται να το προσθέσουμε, εκτός αν δουλεύουμε με παλιό μεταγλωττιστή όπως έχουμε αναφέρει στο κεφάλαιο «Το πρώτο σας πρόγραμμα σε C++».

Δεν γράφουμε Main αλλά int main.

Πρέπει να προσθέσουμε το std:: πριν από το cout ή using std::cout;

using std::endl; μετά από την int main και αντί για std::cout να γράψετε cout ή το using namespace std; μετά από τη main().

Κεφ.2 «Μεταβλητές και σταθερές»

Άσκηση 1

1. unsigned short int
2. unsigned long int ή unsigned float
3. unsigned double
4. unsigned short int

Άσκηση 2

Το παρακάτω είναι μια δήλωση για το PI

```
const float PI = 3.14159;
```

Άσκηση 3

Το παρακάτω δηλώνει και αρχικοποιεί τη μεταβλητή:

```
float myPi = PI;
```

Κεφ.3 «Τελεστές»

Άσκηση 1

Οι τιμές τους είναι myAge: 41, a: 39, b: 41.

Άσκηση 2

1. 32
2. 42
3. 24

Κεφ.4 «Εντολές»

Άσκηση 1

Το παρακάτω είναι μια πιθανή απάντηση:

```
if (x > y)
    x = y;
else      // y > x || y == x
    y = x;
```

Άσκηση 2

Επειδή η γραμμή 6 εκχωρεί την τιμή a-b στο c,η τιμή εκχώρησης είναι a(2) μείον b(2), ή 0.Επειδή το 0 δίνει ψευδές, η if αποτυγχάνει και τίποτα δεν τυπώνεται.

Κεφ.5 «Δείκτες»

Άσκηση 1

1. Το int * pOne; δηλώνει ένα δείκτη σε ένα ακέραιο.
2. Το int vTwo; δηλώνει μια ακέραιη μεταβλητή.
3. Το int * pThree = &vTwo; δηλώνει ένα δείκτη σε ένα ακέραιο και τον αρχικοποιεί με τη διεύθυνση μιας άλλης μεταβλητής, της vTwo.

Άσκηση2

Το παρακάτω είναι μια πιθανή απάντηση:

```
1. # include <iostream>
2.
3. int main()
4. {
5.     int theInteger;
6.     int *pInteger = &theInteger;
7.     *pInteger = 5;
8.     std::cout << "the integer: " <<*pInteger<<std::endl;
9.     char response;
10. std::cin >> response;
11. return 0;
12. }
```

Άσκηση3

Το pint πρέπει να έχει αρχικοποιηθεί. Το πιο σημαντικό, επειδή δεν αρχικοποιήθηκε και δεν του εκχωρήθηκε η διεύθυνση κάποιας μνήμης, δείχνει σε μια τυχαία θέση στη μνήμη. Η εκχώρηση μιας κυριολεκτικής σταθεράς (9) σε αυτή την τυχαία θέση είναι ένα επικίνδυνο λάθος.

Κεφ.6 «Αντικειμενοστραφής προγραμματισμός»

Άσκηση 1

Το παρακάτω είναι μια πιθανή απάντηση:

```
1. class Employee
2. {
3. int Age;
4. int YearsOfService;
5. intSalary;
6. };
```

Άσκηση2

Το παρακάτω είναι μια πιθανή απάντηση:

```
1. //Employee.cpp
2. # include <iostream>
3. # include "Employee.hpp"
4.
5. int Employee::GetAge() const
6. {
7.     return itsAge;
8. }
9. void Employee::SetAge(int age)
10. {
11.     itsAge = age;
12. }
13. int Employee::GetYearsOfService() const
14. {
15.    return itsYearsOfSernice;
16. }
17. void Employee::SetYearsOfService (int years)
18. {
19.     itsYearsOfService = years;
20. }
21. int Employee::GetSalary() const
```

```
22. {
23.     return itsSalary;
24. }
25. void Employee::SetSalary(int salary)
26. {
27.      itsSalary = salary;
28. }
29. int main()
30. {
31.     using namespace std;
32.     Employee John;
33.     Employee Sally;
34.
35.     John.SetAge(30);
36.     John.SetYearsOfService (5);
37.     John.SetSalary(2000);
38.
39.     Sally.SetAge(28);
40.     Sally.SetYearsOfService (3);
41.     Sally.SetSalary(1000);
42.
43.     cout << "At company, John and Sally ";
44.     cout <<"the same job.\n\n";
45.     cout <<"John is "<<John.GetAge()<< "years old."<<endl;
46.     cout <<"John has been with the firm for";
47.     cout <<John.GetYearsOfService()<<" years."<<endl;
48.     cout <<"John earns $ "<<John.GetSalary();
49.     cout <<" per years.\n\n";
50.
51.     cout <<"Sally is " <<Sally.GetAge();
52.     cout <<" years old and has been with the company ";
53.     cout <<Sally.GetYearsOfService ();
54.     cout <<" years.Sally takes " << Sally.GetSalary();
55.     cout <<" per years.\n\n";
```

```
56.
57.     char response;
58. std::cin >> response;
59. return 0;
60. }
```

Άσκηση3

Η μέθοδος πρόσβασης GetAge() είναι ιδιωτική. Να θυμάστε ότι όλα τα μέλη της κλάσης είναι ιδιωτικά, εκτός και αν ορίσετε κάτι άλλο.

Κεφ.7 «Συναρτήσεις»

Άσκηση 1

unsigned long int Perimeter(unsigned short int, unsigned short int);

Άσκηση2

```
1. unsigned long int Perimeter (unsigned short int length, unsigned short int
   width)
2. {
3.         return(2*length) + (2*width);
4. }
```

Άσκηση3

Το παρακάτω είναι μια πιθανή απάντηση:

```
short int Divider(unsigned short int valOne, unsigned short int valTwo)
1. {
2. if valTwo== 0)
3. return -1;
4. else
5. return valOne / valTwo
6. }
```

Κεφ.8 «Ροή προγραμμάτων»

Άσκηση 1

Το παρακάτω είναι μια πιθανή απάντηση:

```
1. for (int i = 0; i< 10; i++)
2. {
3. for (int j = 0; j<10; j++)
4. cout << "0";
5. cout<< endl;
6. }
```

Άσκηση2

Το παρακάτω είναι μια πιθανή απάντηση:

```
int x = 100;
while (x<= 200)
x+=2;
```

Άσκηση3

Το παρακάτω είναι μια πιθανή απάντηση:

```
1. int x = 100;
2. do
3. {
4. x+=2;
5. while (x<=200);
6. }
```

Άσκηση4

Ο counter αρχικοποιείται στο 100, αλλά η συνθήκη ελέγχου εξετάζει το εάν είναι μικρότερος από 10, οπότε ο έλεγχος θα αποτύχει και το σώμα του βρόγχου δεν θα εκτελεστεί ποτέ. Εάν η γραμμή 1 άλλαζε σε int counter = 5;, ο βρόγχος δεν θα τέλειωνε μέχρι να είχε μετρήσει έως τον πιο μικρό πιθανό int.Επειδή εξ' ορισμού, ένας int είναι signed, αυτό δεν θα ήταν σωστό.

Κεφ.9 «Πίνακες»

Άσκηση1

Ο πίνακας έχει πέντε στοιχεία επί τέσσερα στοιχεία, αλλά ο κώδικας αρχικοποιείται 4X5 στοιχεία.

Άσκηση2

Το παρακάτω είναι μια πιθανή απάντηση. Ο πίνακας σας μπορεί να έχει διαφορετικό όνομα, αλλά πρέπει να ακολουθείται από το [3][3], προκειμένου να περιέχει ένα πίνακα 3X3.

int GameBoard[3][3];

Άσκηση3

Θέλατε να γράψετε i<5, αλλά γράψατε i<=5. Ο κώδικας θα τρέξει όταν το ι = = 5 και το j = = 4, αλλά δεν υπάρχει ένα στοιχείο SomeArray[5][4].

Κεφ.10 «Λίστες»

Άσκηση1

std::list <type> name (String);

std::list <type> name (Cat);

std::list <type> name (int);

Άσκηση 2

Το παρακάτω είναι μια πιθανή απάντηση:

```
1. # include <iostream>
2. # include <list>
3. int main()
4. {
5.       std::list<char> charList;
6.         for (int x=0; x<6; ++x)
7.       charList.push_back(65 + x);
```

```
8.       std::cout <<"original list: ";
9.       std::list<char>::iterator iter;
10.        for (iter = charList.begin();
11.       iter !=charList.end(); iter++)
12.    {
13.        std::cout <<*iter;
14.     }
15.     std::cout << std::endl;
16.     charList.remove('E');
17.     std::cout <<"resultant list: ";
18.      for (iter = charList.begin();
19.       iter !=charList.end(); iter++)
20.     {
21.          std::cout <<*iter;
22.      }
23.       std::cout << std::endl;
24.  char response;
25. std::cin >> response;
26. return 0;
27. }
```

Κεφ.11 «Στοίβα»

Άσκηση 1

Το παρακάτω είναι μια πιθανή απάντηση:

```
1.  # include <iostream>
2.  # include <list>
3.  # include <stack>
4.
5.  int main()
6.    {
7.      std::stack<int, std::list<int> > intStack;
8.      std::cout <<"Values pushed onto stack: " <<std::endl;
```

```
9.      for (int x=1; x<5; ++x)
10.     {
11.       intStack.push(x*1);
12.       std::cout <<x*1 <<std::endl;
13.     }
14.    std::cout<<"Values popped from stack: " <<std::endl;
15.    int size = intStack.size();
16.    for (int x=0; x<size; ++x)
17.     {
18.       std::cout<<intStack.top() <<std::endl;
19.       intStack.pop();
20.     }
21. char response;
22. std::cin >> response;
23. return 0;
24. }
```

Άσκηση 2

Το παρακάτω είναι μια πιθανή απάντηση:

```
1.  # include <iostream>
2.  # include <list>
3.  # include <stack>
4.
5.  int main()
6.   {
7.     std::stack<int, std::list<int> > intStack;
8.     std::cout <<"Values pushed onto stack: " <<std::endl;
9.     for (int x=1; x<12; ++x)
10.    {
11.      intStack.push(x*1);
12.      std::cout <<x*1 <<std::endl;
13.    }
14.   std::cout<<"Values popped from stack: " <<std::endl;
15.      std::cout<<intStack.top() <<std::endl;
```

```
16.         intStack.pop();
17.       std::cout <<"Values pushed onto stack: " <<std::endl;
18.      for (int x=0; x<6; ++x)
19.       {
20.         intStack.push(x*1);
21.         std::cout <<x*1 <<std::endl;
22.       }
23. char response;
24. std::cin >> response;
25. return 0;
26. }
```

Κεφ.12 «Ουρά»

Άσκηση 1

Το παρακάτω είναι μια πιθανή απάντηση:

```
1.  # include <iostream>
2.  # include <list>
3.  # include <queue>
4.
5.  int main()
6.  {
7.       std::queue<int, std::list<int> > intQueue;
8.       std::cout <<"Values pushed onto queue: " <<std::endl;
9.       for (int x=1; x<21; ++x)
10.     {
11.         intQueue.push(x*1);
12.         std::cout <<x*1 <<std::endl;
13.     }
14.
15.      char response;
16. std::cin >> response;
17. return 0;
```

```
18. }
```

Άσκηση 2

Το παρακάτω είναι μια πιθανή απάντηση:

```
1.  # include <iostream>
2.  # include <list>
3.  # include <queue>
4.
5.  int main()
6.  {
7.  std::queue<int, std::list<int> > intQueue;
8.  std::cout <<"Values pushed onto queue: " <<std::endl;
9.   for (int x=1; x<15; ++x)
10.     {
11.       intQueue.push(x*1);
12.       std::cout <<x*1 <<std::endl;
13.      }
14. std::cout<<"Values popped from queue: " <<std::endl;
15. std::cout<<intQueue.front() <<std::endl;
16. intQueue.pop();
17. std::cout <<"Values pushed onto Queue: " <<std::endl;
18. for (int x=0; x<9; ++x)
19.      {
20.         intQueue.push(x*1);
21.         std::cout <<x*1 <<std::endl;
22.      }
23. char response;
24. std::cin >> response;
25. return 0;
26. }
```

Βιβλιογραφία

Ελληνική

1. Ι.Στρούτζος «*Δομές δεδομένων, αρχείων και τεχνικές προγραμματισμού σε γλώσσα Pascal και C++*», Αθήνα, Ίων 1997.
2. Ιωάννης Τσούλος «*Ασκήσεις*», Τμήμα τηλεπληροφορικής & διοίκησης ΤΕΙ Ηπείρου.
3. Βασίλειος Βεσκούκης «*Εισαγωγή στην πληροφορική*», Εθνικό Μετσόβειο Πολυτεχνείο.
4. Διομήδης Σπινέλλης «*Αντικείμενα στη C++*», Οικονομικό Πανεπιστήμιο Αθηνών.
5. Χ.Κοίλιας «*Δομές δεδομένων και οργάνωση αρχείων*», Νέων Τεχνολογιών 1993.
6. Διομήδης Σπινέλλης «*Υλοποίηση με έτοιμες βιβλιοθήκες*», Οικονομικό Πανεπιστήμιο Αθηνών, 2004.
7. Διομήδης Σπινέλλης «*Πρότυπα στη C++*» Οικονομικό Πανεπιστήμιο Αθηνών, 2005
8. Αδαμόπουλος Νίκος «*Ασκήσεις στη γλώσσα προγραμματισμού C++*» 2006.
9. Ίων Ανδρουτσόπουλος «*Προγραμματισμός υπολογιστών με C++*» 2005.
10. Γ.Βλέτσας «*Διαχείριση δεδομένων:Δομές αρχείων, μέθοδοι οργάνωσης αρχείων, δομές δεδομένων, τράπεζες πληροφοριών*», Αθήνα Χ.Ο 1987.
11. Σταματιάδης Σταμάτης, σημειώσεις, 2006.
12. Μαρκίνος Αθανάσιος «*Αντικειμενοστραφής Προγραμματισμός C++*» ΤΕΙ Λάρισας, 2002.
13. «*Αντικειμενοστραφής Προγραμματισμός*» Σημειώσεις, Τμήμα ΜΠΔ.
14. Ζ.Θεοδοσία, «*Αντικειμενοστραφής Προγραμματισμός C++*» Εργαστηριακές Ασκήσεις, ΤΕΙ Ηπείρου.
15. «*C++: Σημειώσεις* », Πανεπιστήμιο Μακεδονίας, 2005.
16. Ιζαμπώ Καράλη, «*Αντικειμενοστραφής Προγραμματισμός*», 2005.
17. ΚΕΝΤΡΟ ΠΛΗ.ΝΕ.Τ. Ν. ΦΛΩΡΙΝΑΣ «*Η Γλώσσα Προγραμματισμού C++*».
18. Β.Βεσκούκης, Ρ.Κορακίτης, «*Εισαγωγή στην Πληροφορική*» Εθνικό Μετσόβιο Πολυτεχνείο.
19. Χ.Βασίλειος «*Κώδικας σε γλώσσα C++*», ΤΕΙ Ηπείρου.

20. Ι.Μανωλόπουλος «*Δομές δεδομένων*», Art of Text, 1998
21. Νικόλαος Ζάχαρης «*Προγραμματισμός Α*» ΤΕΙ Πειραιά.
22. Β.Βεσκούκης, Ρ.Κορακίτης, «*Προγραμματιστικές Τεχνικές*», Εθνικό Μετσόβιο Πολυτεχνείο.
23. Δ.Φωτάκης «*Δομές δεδομένων: εισαγωγικά*», Πανεπιστήμιο Αιγαίου, 2005.
24. Δ.Φωτάκης «*Δέντρα Αναζήτησης*», Πανεπιστήμιο Αιγαίου, 2005.
25. Κ.Μαργαρίτης, Γ.Τσιομπίκας ,«*Εισαγωγή στην C++*», 2004.
26. Ν.Παπασπύπρου «*Τεχνολογία Λογισμικού*» Σημειώσεις.
27. Σταμάτης Σταματιάδης, «*Εισαγωγή στη γλώσσα προγραμματισμού C++*», Πανεπιστήμιο Κρήτης.
28. Παναγιωτόπουλος Ι.Χ «*C++/ Visual C++*», Σταμούλης Α., 2002.

Αγγλική

1. Simon and Schuster Company «*Special Edition Using Visual C++ 5*»1997.
2. Macmillan Computer Publishing «*Special Edition Using Visual C++ 6*».
3. Stroustrup «*Η γλώσσα προγραμματισμού C++*», Κλειδάριθμος 1992.
4.
5. Andrew Koenig and B. E.Moo «*Accelerated C++*», Addison Wesley 2000.
6. Nicolai M. Josuttis «*The C++ Standard Library*», Addison Wesley, 1999.
7. Mickey Williams «*Teach Yourself Visual C++® 5 in 24 Hours*».
8. Herbert Schildt «*C++: The Complete Reference*» (2nd Edition), McGraw-Hill, 1995.
9. Robert Lafore, «*Object-oriented Programming in C++*», SAMS, 2002
10. Bjarne Stroustrup «*Η Γλώσσα Προγραμματισμού C++*» Κλειδάριθμος, 1999.
11. Frank B. Brokken «*C++ Annotations* 6.2.4».
12. Cameron Huqhes, Tracey Huqhes «*Parallel and Distributed Programming Using C++*», Addison Wesley, 2003.
13. Kate Gregory «*Microsoft® Visual C++® .NET 2003 Kick Start*», Sams Publishing, 2003.
14. Bruce Eckel «*Thinking in C++, 2nd ed. Volume 1*»2000.

15. Stroustrup Bjarne «*The C++ Language*» Addison Wesley Longman,1997.
16. Frank B. Brokken «*C++ Annotations Versions 5.1.0.a - center*».
17. S. Qualline «*Practical C++ Programming*», O'Reilly
18. David Levine «*Teach Yourself C++ in 21 Days*».
19. Lyn Robison «*Sams Teach Yourself Database Programming with Visual C++ in 21 Days*», Macmillan Computer, 1999.
20. Stroustrup Bjarne «*C++ Network Programming*»
21. Terry Chapman «*Software Engineering using C++*», 1999.
22. Satya Sai Kolachina « *C++ Builder 6 Developer's Guide*» Wordware Publishing, 2002.

Links

1. www.cplusplus.com
2. www.Cprogramming.com
3. www.mindview.net
4. The C++ Standards Committee
5. C++ Stuff by Neil
6. C++ Language Tutorial
7. The programmer's cafe - Tutorial C++