ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΡΧΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

1ο ΕΞΑΜΗΝΟ

Ένα Απλό Πρόγραμμα σε C++

Τα προγράμματα της C++ έχουν την επέκταση ***CPP*** και μπορούμε να τα δημιουργήσουμε με τον Editor του DOS. Ένα πολύ απλό πρόγραμμα θα μπορούσε να είναι το παρακάτω :

```
#include <iostream.h>

void main(void)

{

            cout << "Η C++ κάνει θαύματα!";

}
```

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.h. Τα αρχεία που έχουν επέκταση h ονομάζονται *αρχεία επικεφαλίδων (header files)* και είναι αρχεία ASCII. Το κάθε αρχείο επικεφαλίδας περιέχει ορισμένους ορισμούς που χρησιμοποιεί ο μεταγλωττιστής της C++ για διάφορες λειτουργίες. Η πρόταση *void main(void)* προσδιορίζει τη θέση εκκίνησης του προγράμματος, δηλ. το κύριο μέρος (υπορουτίνα) του προγράμματος που θα εκτελείται πρώτο. Τα προγράμματά μας πρέπει πάντα να περιέχουν μία και μόνο μία πρόταση που να περιέχει το όνομα *main*. Η λέξη *void* κανονικά χρησιμοποιείται για να υποδηλώσει ότι μια συνάρτηση (ή πρόγραμμα) δεν επιστρέφει τιμή, ενώ η λέξη void μέσα στις παρενθέσεις δηλώνει ότι δεν μεταβιβάζονται τιμές στη συνάρτηση ή στο πρόγραμμα.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.

Η Εντολή Cout

Με την εντολή cout μπορούμε να εμφανίσουμε ακόμη και αριθμούς ή και να χρησιμοποιήσουμε τον *τελεστή εισαγωγής* << περισσότερες από μία φορές στην ίδια πρόταση, όπως φαίνεται στο παρακάτω πρόγραμμα :

```
#include <iostream.h>

void main(void)

{

            cout << 1001;

            cout << 0.12345;

            cout << 1 << 0 << 0 << 1;

            cout << "Ο αριθμός " << 1001 << " είναι ο τυχερός μου";

}
```

Κάθε φορά που η C++ συναντάει τον τελεστή εισαγωγής, απλώς προσαρτεί όσους χαρακτήρες ακολουθούν σ' εκείνους που βρίσκονται ήδη στο ρεύμα εξόδου. Για να αλλάξουμε γραμμή, πρέπει να τοποθετήσουμε στο ρεύμα εξόδου τον *χαρακτήρα αλλαγής γραμμής (\n)* μέσα σε εισαγωγικά ή τη συμβολική λέξη *endl*.

Δείτε το παρακάτω πρόγραμμα :

```
#include <iostream.h>

void main(void)

{

            cout << "Γραμμή Νο 1 \n Γραμμή Νο 2";

            cout << 1 << '\n' <<0 << '\n';

            cout << "Η C++ " << endl << "κάνει θαύματα!";

}
```

Ο Χειριστής Setw

Με τον χειριστή setw μπορούμε να καθορίσουμε το πλάτος εξόδου, δηλ. τον ελάχιστο αριθμό χαρακτήρων, αλλά για μία μόνο τιμή μέσα στην εντολή cout. Για να μπορέσουμε να χρησιμοποιήσουμε τον χειριστή setw, το πρόγραμμα πρέπει να περιέχει το αρχείο επικεφαλίδας iomanip.h. Αν θέλουμε να καθορίσουμε πλάτη για πολλές τιμές εξόδου, θα πρέπει να χρησιμοποιήσουμε το setw πολλές φορές.

```
#include <iostream.h>

#include <iomanip.h>

void main(void)

{

     cout << "Ο αριθμός μου είναι ο " << setw(6) << 1001 <<endl;

}
```

Οι Ειδικοί Χαρακτήρες

Οι ειδικοί χαρακτήρες της C++ έχουν σαν πρόθεμα τον χαρακτήρα \ και είναι οι εξής :

\a	ήχος	*\ooo*	8δική τιμή, π.χ. \007
\b	οπισθοχώρηση δρομέα	*\xhhh*	16δική τιμή, \xFFFF
\f	αλλαγή σελίδας		
\n	αλλαγή γραμμής		

\r	επαναφορά στην αρχή γραμμής
\t	οριζόντιος στηλοθέτης (tab)
\v	κατακόρυφος στηλοθέτης
\\	ο χαρακτήρας \
\?	ο χαρακτήρας ?
\’	ο χαρακτήρας ‘
\"	ο χαρακτήρας "
\0	ο μηδενικός χαρακτήρας

Εμφάνιση 8-δικών και 16-δικών Αριθμών

Για να εμφανίσουμε τα δεδομένα εξόδου σε 8δική, 16δική ή 10δική μορφή, μπορούμε να χρησιμοποιήσουμε τους χειριστές εξόδου oct, hex και dec αντίστοιχα. Όταν χρησιμοποιήσουμε έναν απ’ αυτούς τους χειριστές, η επιλογή μας ισχύει μέχρι το τέλος του προγράμματος ή μέχρι να χρησιμοποιήσουμε έναν άλλον χειριστή.

```
#include <iostream.h>

void main(void)

{

        cout << "Οκταδικό σύστημα : " << oct << 10 << 20 << endl;

        cout << "Δεκαεξαδικό σύστημα : " << hex << 10 << 20 << endl;

        cout << "Δεκαδικό σύστημα : " << dec << 10 << 20 << endl;

}
```

Δήλωση Μεταβλητών

Οι τύποι μεταβλητών που χρησιμοποιούνται κυρίως στη C++ είναι οι εξής :

char	ακέραιες τιμές από -128 έως 127
int	ακέραιες τιμές από -32768 έως 32767
unsigned	ακέραιες τιμές από 0 έως 65535
long	μεγάλες ακέραιες τιμές
float	μεγάλες πραγματικές τιμές
double	πολύ μεγάλες πραγματικές τιμές

```
#include <iostream.h>

void main(void)

{

            // δήλωση μεταβλητών

            int ilikia = 32;

            float misthos = 350000.00;



        // κύριο σώμα του προγράμματος

        cout << "Είμαι " << ilikia << " ετών" << endl;

       cout << " και έχω εισόδημα " << misthos << " δραχμές" << endl;

}
```

Η Εντολή Cin

Με την εντολή cin μπορούμε να διαβάσουμε δεδομένα από το πληκτρολόγιο και να καθορίσουμε μία ή περισσότερες μεταβλητές στις οποίες θα αποδοθούν τα δεδομένα αυτά. Με την εντολή cin χρησιμοποιούμε τον τελεστή εξαγωγής >>.

Δείτε τα παρακάτω προγράμματα :

```
#include <iostream.h>

void main(void)

{

            int number;



        cout << "Γράψτε έναν αριθμό και πατήστε enter : ";

        cin >> number;

        cout << "Ο αριθμός σας είναι : " << number << endl;

}
```

```
#include <iostream.h>

void main(void)

{

            int first, second;


 cout << "Γράψτε δύο αριθμούς και πατήστε enter : ";

 cin >> first >> second;

 cout << "Οι αριθμοί είναι : " << first << " και " << second << endl;

}
```

Όταν χρησιμοποιούμε την εντολή cin για να διαβάζουμε χαρακτήρες από το πληκτρολόγιο, αυτή ψάχνει για τον πρώτο χαρακτήρα λευκού διαστήματος, όπως κενό διάστημα, στηλοθέτη (tab), πλήκτρο enter, για να ξέρει πού τελειώνει η μια τιμή και πού αρχίζει η δεύτερη.

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.

Δείτε τα παρακάτω προγράμματα :

```
#include <iostream.h>

void main(void)

{

            int test = 95;


            if (test >= 90)

                  cout << "Μπράβο, πέρασες !" << endl;

}
```

```
 #include <iostream.h>

void main(void)

{

            int test;
```

```
            cout << "Γράψτε έναν βαθμό και πατήστε enter : ";
            cin >> test;

            if (test >= 90)
      {
            cout << "Μπράβο, πέρασες !" << endl;
            cout << "Ο βαθμός σου είναι : " << test << endl;
      }
       else
       {
             cout << "Δυστυχώς, κόπηκες!" << endl;
             cout << "Έχασες " << 100-test << " μονάδες" << endl;
       }
}
```

Η Εντολή For

Η εντολή αυτή είναι παρόμοια με την αντίστοιχη εντολή της C. Δείτε το παρακάτω παράδειγμα :

```
#include <iostream.h>

void main(void)

{
            int counter;
            int total = 0;

            for (counter=1; counter<=10; counter++)
       {
            cout << counter << ' ';
```

```
            total = total + counter;

        }

}
```

Η Εντολή While

Η εντολή αυτή είναι παρόμοια με την αντίστοιχη εντολή της C. Δείτε το παρακάτω παράδειγμα :

```
#include <iostream.h>

void main(void)

{

            int done = 0;

            char letter;


            while (!done)

            {

cin >> letter;

if ((letter == 'N') || (letter == 'O'))

            done = 1;

else

            cout << '\a';

            }

}
```

Η Εντολή Do While

Η εντολή αυτή είναι παρόμοια με την αντίστοιχη εντολή της C. Δείτε το παρακάτω παράδειγμα :

```
#include <iostream.h>

void main(void)

{

            int done = 0;
```

```
            char letter;

            do

            {

cin >> letter;

if ((letter == 'N') || (letter == 'O'))

            done = 1;

else

            cout << '\a';

            } while (!done);

}
```

Ο Τελεστής Διάκρισης Καθολικής Εμβέλειας

Όταν σ' ένα πρόγραμμα υπάρχουν μια τοπική και μια καθολική μεταβλητή με το ίδιο όνομα, μέσα στη συνάρτηση υπερισχύει η τοπική μεταβλητή, όπως ξέρουμε από την απλή C. Για να έχουμε, όμως, πρόσβαση στην καθολική μεταβλητή μέσα στη συνάρτηση, πρέπει να χρησιμοποιήσουμε τον τελεστή διάκρισης καθολικής εμβέλειας (global resolution operator) της C++, που είναι ο ::.

Δείτε το παρακάτω παράδειγμα :

```
#include <iostream.h>

int number = 1001;     // καθολική μεταβλητή

void show_numbers(int number)

{

      cout << number << endl;    // εμφάνιση  τοπικής μεταβλητής

      cout << ::number << endl;  // εμφάνιση καθολικής μεταβλητής

}

void main(void)

{

            int value = 2002;

             show_numbers(value);

}
```

Η Υπέρβαση Συναρτήσεων (Function Overloading)

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.

Το επόμενο παράδειγμα 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.

```
#include <iostream.h>

int add_values(int a, int b)

{

            return(a+b);

}

int add_values(int a, int b, int c)

{

            return(a+b+c);

}

void main(void)

{

      cout << add_values(200, 800) << endl;  // καλείται η 1η

      cout << add_values(100, 200, 300) << endl;// καλείται η 2η

}
```

Οι Αναφορές (References)

Η αναφορά (reference) είναι ένα ψευδώνυμο (alias), δηλ. ένα δεύτερο όνομα, που μπορούμε να χρησιμοποιούμε για να αναφερόμαστε σε μια μεταβλητή. Η χρήση των αναφορών κάνει την αλλαγή τιμής των παραμέτρων μιας συνάρτησης πολύ εύκολη.

Για να δηλώσουμε μια αναφορά, τοποθετούμε το σύμβολο & αμέσως μετά από τον τύπο της μεταβλητής, μετά το όνομα της αναφοράς, το σύμβολο = και το όνομα της μεταβλητής της οποίας ψευδώνυμη είναι η αναφορά, ως εξής :

```
int& synonymo = metavliti;

float& salary_alias = salary;
```

Αφού δηλώσουμε μια αναφορά, μπορούμε να χρησιμοποιήσουμε είτε τη μεταβλητή είτε την αναφορά, ως εξής :

```
synonymo = 1001;

metavliti = 1001;
```

Το επόμενο παράδειγμα δημιουργεί μια αναφορά με το όνομα alias_ name, που συνδέεται με τη μεταβλητή number, και μετά χρησιμοποιεί και το ψευδώνυμο και τη μεταβλητή.

```
#include <iostream.h>

void main(void)

{

        int number = 501;

        int& alias_name = number; // δημιουργία της αναφοράς

        alias_name = alias_name + 500;// αλλάζει και η μεταβλητή
        και η  αναφορά

}
```

Ο βασικός σκοπός μιας αναφοράς είναι να κάνει απλούστερη τη διαδικασία αλλαγής των τιμών των παραμέτρων μέσα σε μια συνάρτηση. Το επόμενο πρόγραμμα αποδίδει την αναφορά number_alias στη μεταβλητή number και μεταβιβάζει την αναφορά στη συνάρτηση change_value(), η οποία αλλάζει την τιμή της μεταβλητής.

```
#include <iostream.h>

void change_value(int& alias)

{

        alias = 1001;

}

void main(void)

{

        int number;

        int& number_alias = number;

        change_value(number_alias);

}
```

Το πρόγραμμα μεταβιβάζει την αναφορά στη συνάρτηση change_value() και στη δήλωση της συνάρτησης η παράμετρος alias δηλώνεται σαν μια αναφορά σε τιμή τύπου int. Η συνάρτηση

change_value() μπορεί να αλλάξει την τιμή της παραμέτρου χωρίς να χρησιμοποιηθεί δείκτης (pointer).

Πρέπει να έχουμε υπόψη μας ότι μια αναφορά δεν είναι μεταβλητή και από τη στιγμή που θα αποδώσουμε μια μεταβλητή σε μια αναφορά, δεν μπορούμε να την αλλάξουμε. Η υπερβολική χρήση αναφορών μπορεί να οδηγήσει σε πολύ δυσνόητο κώδικα προγράμματος.

Προεπιλεγμένες Τιμές σε Παραμέτρους Συναρτήσεων

Οι προεπιλεγμένες τιμές χρησιμοποιούνται από τη συνάρτηση όταν το πρόγραμμα που την καλεί παραλείπει τιμές για κάποιες παραμέτρους. Η απόδοση προεπιλεγμένων τιμών σε παραμέτρους συναρτήσεων μέσα στον ορισμό της συνάρτησης είναι πολύ εύκολη. Απλά χρησιμοποιούμε τον τελεστή απόδοσης τιμής, ως εξής :

```
void sinartisi(int size=12, float cost=19.95)
{
        ...
}
```

Στο επόμενο παράδειγμα αποδίδονται προεπιλεγμένες τιμές στις παραμέτρους μιας συνάρτησης και μετά το πρόγραμμα καλεί τη συνάρτηση αρχικά χωρίς να καθορίζει τιμές για καμία παράμετρο και μετά καθορίζοντας τιμές σταδιακά και για τις τρεις παραμέτρους. Ç όδíάρτηση χρησιμοποιεί τις προεπιλεγμένες τιμές των παραμέτρων όπου χρειάζεται.

```
#include <iostream.h>

void show(int a=1, int b=2, int c=3)

{

        cout << a << b << c << endl;

}



void main(void)

{

        show();   // a=1, b=2, c=3

        show(1001);  // a=1001, b=2, c=3

        show(1001, 2002);  // a=1001, b=2002, c=3

        show(1001, 2002, 3003); // a=1001, b=2002, c=3003

}
```

Οι Ενώσεις (Unions)

Μια ένωση (union) της C++ μοιάζει πολύ με μια δομή (struct) και όπως και στις δομές, ο ορισμός της ένωσης δεν δεσμεύει μνήμη, αλλά παρέχει ένα πρότυπο για μελλοντικές δηλώσεις μεταβλητών.

Δείτε τα παρακάτω παραδείγματα :

```
union distance {
            int miles;
            long meters;
            } japan, germany, france;


union distance {
            int miles;
            long meters;
            };
distance japan germany, france;
```

Το πρόγραμμα μπορεί να δώσει τιμή σε οποιαδήποτε από τα μέλη μιας ένωσης, αλλά, αντίθετα με ό,τι ισχύει σε μια δομή, μπορεί να δώσει τιμή μόνο σ' ένα μέλος της ένωσης κάθε φορά. Η C++ κατανέμει όση μνήμη χρειάζεται για την αποθήκευση του μεγαλύτερου μέλους της ένωσης. Δηλαδή στην περίπτωση της παραπάνω ένωσης, η C++ κατανέμει 4 bytes. Για να προσπελάσουμε τα μέλη μιας ένωσης χρησιμοποιούμε τον τελεστή της τελείας και όταν δίνουμε τιμή σ' ένα μέλος, χάνεται η τιμή του άλλου μέλους. Δείτε το παρακάτω παράδειγμα :

```
#include <iostream.h>

void main(void)

{

union distance {
                int miles;
                long meters;
              } walk;
           walk.miles = 5;
          cout << walk.miles << endl;
```

```
            walk.meters = 10;

            cout << walk.meters << endl;

}
```

Οι Τάξεις (Classes)

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 πράγματα (πληροφορίες, δεδομένα, μέλη, πεδία) που αποτελούν ένα σύστημα καθώς με τις πράξεις (συναρτήσεις, μέθοδοι) που μπορούμε να κάνουμε μ' αυτά τα πράγματα. 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 :

```
class class_name   // όνομα της τάξης

{

        int data_member;   // μέλος δεδομένων

        void show_member(int);// συνάρτηση μέλος ή μέθοδος

 };
```

Αφού ορίσουμε μια τάξη, μπορούμε να δηλώσουμε μεταβλητές του συγκεκριμένου τύπου τάξης, που ονομάζονται αντικείμενα (objects), ως εξής:

```
            class_name class_1, class_2, class_3;
```

Δείτε το παρακάτω παράδειγμα που δημιουργεί μια τάξη με το όνομα employee, η οποία περιέχει μεταβλητές δεδομένων και ορισμούς μεθόδων :

```
class employee {

    public:

            char name[64];

            long employee_id;

            float salary;

            void show_employee(void)

            {

                        cout << name << endl;

                        cout << employee_id << endl;
```

```
                                cout << salary << endl;

                }

        };
```

Η παραπάνω τάξη περιέχει τρία μέλη που είναι μεταβλητές και ένα μέλος που είναι συνάρτηση. Όπως θα δούμε αργότερα, τα μέλη των τάξεων μπορεί να είναι δημόσια (public) ή ιδιωτικά (private). Εδώ όλα τα μέλη είναι δημόσια, που σημαίνει ότι τα προγράμματα μπορούν να προσπελάζουν όλα τα μέλη της τάξης με τον τελεστή τελείας.

Το παρακάτω παράδειγμα ορίζει μια τάξη employee, όπως την παραπάνω, και δημιουργεί δύο αντικείμενα του τύπου employee. Το πρόγραμμα δίνει τιμές στα μέλη δεδομένων και χρησιμοποιεί τη συνάρτηση μέλος show_ member για να εμφανίσει τα στοιχεία των υπαλλήλων.

```
#include <iostream.h>

#include <string.h>

class employee {

    public:

            char name[64];

            long employee_id;

            float salary;

            void show_employee(void)

            {

                        cout << name << endl;

                        cout << employee_id << endl;

                        cout << salary << endl;

            }

};



void main(void)

{

        employee worker, boss; // αντικείμενα της τάξης employee

        strcpy(worker.name, "Παπαδόπουλος Αντώνιος");

        worker.employee_id = 100;
```

```
        worker.salary = 250000;

        strcpy(worker.name, "Ιωαννίδης Δημήτριος");

        worker.employee_id = 101;

        worker.salary = 300000;

        worker.show_employee();

        boss.show_employee();

}
```

Στο προηγούμενο παράδειγμα, η συνάρτηση έχει οριστεί μέσα στην ίδια την τάξη και έτσι ονομάζεται συνάρτηση inline. Για ευκολία, μπορούμε να τοποθετήσουμε ένα πρωτότυπο της συνάρτησης μέσα στην τάξη και να ορίσουμε τη συνάρτηση έξω από την τάξη.

Επειδή διαφορετικές τάξεις μπορούν να χρησιμοποιούν συναρτήσεις με το ίδιο όνομα, μπροστά από τα ονόματα των συναρτήσεων που δηλώνονται έξω από μια τάξη, πρέπει να τοποθετούμε το όνομα της τάξης και τον τελεστή διάκρισης εμβέλειας (::). Έτσι, ο προηγούμενος ορισμός της τάξης employee θα μπορούσε να γίνει και ως εξής :

```
class employee {

    public:

            char name[64];

            long employee_id;

            float salary;

            void show_employee(void);

            };


            void employee::show_employee(void);

            {

                        cout << name << endl;

                        cout << employee_id << endl;

                        cout << salary << endl;

            };
```

Τα Δημόσια και τα Ιδιωτικά Μέλη Δεδομένων

Όταν δημιουργούμε μια τάξη, μπορεί να υπάρχουν μέλη που τις τιμές τους τις χρησιμοποιεί εσωτερικά η τάξη, αλλά το ίδιο το πρόγραμμα δεν χρειάζεται να τις προσπελάζει. Αυτά τα μέλη είναι ιδιωτικά μέλη και πρέπει να κρύβονται από το πρόγραμμα. Εξ ορισμού, αν δεν χρησιμοποιήσουμε την ετικέτα public (δημόσια), η C++ θεωρεί όλα τα μέλη των τάξεων σαν private (ιδιωτικά).

Τα προγράμματα δεν μπορούν να προσπελάζουν τα ιδιωτικά μέλη των τάξεων με τον τελεστή τελείας, αλλά μόνο με τις μεθόδους της τάξης. Δείτε το παρακάτω παράδειγμα ορισμού δημόσιων και ιδιωτικών μελών :

```
class some_class {

    public:

            int some_variable;

            void init_private(int, float);

            void show_data(void);


    private:

            int key_value;

            float key_number;

};
```

Οι μέθοδοι των τάξεων μέσω των οποίων γίνεται η πρόσβαση στα ιδιωτικά μέλη δεδομένων της τάξης ονομάζονται συναρτήσεις διασύνδεσης (interface functions). Θα μπορούμε να χρησιμοποιούμε τις συναρτήσεις διασύνδεσης για να προστατεύουμε τα δεδομένα μιας τάξης.

Οι Συναρτήσεις Εγκατάστασης και Αποσύνδεσης

Όταν δημιουργούμε ένα αντικείμενο, για να αποδώσουμε αρχικές τιμές στα μέλη δεδομένων του, χρησιμοποιούμε τη συνάρτηση εγκατάστασης (constructor), που εκτελείται αυτόματα για κάθε αντικείμενο που δημιουργούμε. Η C++ διαθέτει ακόμη και τη συνάρτηση αποσύνδεσης (destructor), που εκτελείται αυτόματα όταν διαγράφουμε ένα αντικείμενο. Οι συναρτήσεις αποσύνδεσης είναι χρήσιμες όταν έχει κατανεμηθεί χώρος μνήμης για ένα αντικείμενο και θέλουμε να αποδεσμευτεί αυτή η μνήμη πριν διαγραφεί το αντικείμενο.

Η συνάρτησης εγκατάστασης είναι στην ουσία μια μέθοδος μιας τάξης που έχει το ίδιο όνομα με την τάξη. Αν έχουμε ορίσει μια συνάρτηση εγκατάστασης, η C++ την καλεί αυτόματα κάθε φορά που δημιουργούμε ένα αντικείμενο. Μια συνάρτηση εγκατάστασης δεν μπορεί να επιστρέφει τιμή και δεν προσδιορίζουμε γι' αυτήν κανέναν επιστρεφόμενο τύπο.

Δείτε το παρακάτω παράδειγμα :

```
class employee {

    public:

            employee(char *, long, float); // συνάρτηση εγκατάστασης

            void show_employee(void);

            int change_salary(float);

            long get_id(void);

    private:

            char name[64];

            long employee_id;

            float salary;

};
```

Μέσα στο πρόγραμμα, η συνάρτηση εγκατάστασης ορίζεται όπως οποιαδήποτε μέθοδος τάξης :

```
employee::employee(char *name, long employee_id, float salary)

{

            strcpy(employee::name, name);

            employee::employee_id = employee_id;

            employee::salary = salary;

}

void main(void)

{

            employee worker("Δημητριάδης Γεώργιος", 101, 350000.0);


            worker.show_employee();

}
```

Όπως βλέπουμε από το παραπάνω παράδειγμα, η δήλωση ενός αντικειμένου ακολουθείται από παρενθέσεις και τις αρχικές τιμές του αντικειμένου, που μεταβιβάζονται σαν παράμετροι στη συνάρτηση εγκατάστασης.

Μια συνάρτηση αποσύνδεσης εκτελείται αυτόματα κάθε φορά που διαγράφεται ένα αντικείμενο. Οι συναρτήσεις αποσύνδεσης έχουν το ίδιο όνομα με την τάξη του αντικειμένου, αλλά με τον χαρακτήρα (~) μπροστά από το όνομά τους. Σε μια συνάρτηση αποσύνδεσης δεν μπορούμε να μεταβιβάσουμε παραμέτρους.

Το επόμενο παράδειγμα ορίζει μια συνάρτηση αποσύνδεσης για την τάξη employee :

```
void employee::~employee(void)

{

      cout << "Έγινε αποσύνδεση του αντικειμένου : " << name << endl;

}
```

Η C++ καλεί αυτόματα τη συνάρτηση αποσύνδεσης για κάθε αντικείμενο όταν τερματίζεται το πρόγραμμα και μάλιστα χωρίς να χρησιμοποιήσει κλήση συνάρτησης.

Η Υπέρβαση Τελεστών

Όταν ορίζουμε μια νέα τάξη, στην ουσία ορίζουμε έναν νέο τύπο δεδομένων και η C++ μάς επιτρέπει να καθορίζουμε πώς θα συμπεριφέρονται οι τελεστές σ' αυτόν τον νέο τύπο. Η υπέρβαση τελεστών (operator overloading) είναι η διαδικασία αλλαγής της σημασίας ενός τελεστή για μια συγκεκριμένη τάξη. Όταν χρησιμοποιούμε υπέρβαση σ' έναν τελεστή για κάποια τάξη, η λειτουργία του τελεστή δεν αλλάζει για τους άλλους τύπους μεταβλητών. Στο παρακάτω παράδειγμα θα ορίσουμε μια τάξη αλφαριθμητικών με το όνομα string και θα χρησιμοποιήσουμε υπέρβαση στους τελεστές + και -, όπου στα αντικείμενα της τάξης string, ο τελεστής + θα προσαρτεί τους χαρακτήρες και ο τελεστής - θα αφαιρεί κάθε εμφάνιση ενός γράμματος από το αλφαριθμητικό. Όταν θέλουμε να εφαρμόσουμε υπέρβαση σ' έναν τελεστή, χρησιμοποιούμε τη λέξη κλειδί operator της C++ για να προσδιορίσουμε τον τελεστή και την αντίστοιχη μέθοδο. Στη συνέχεια, το πρόγραμμα πρέπει να ορίσει τις συναρτήσεις που υλοποιούν τους τελεστές υπέρβασης.

```
#include <iostream.h>

#include <string.h>

class string {

    public:

         string(char *); // συνάρτηση εγκατάστασης

         void operator +(char *);// συνάρτηση υπέρβασης του τελεστή +

         void operator -(char);// συνάρτηση υπέρβασης του τελεστή -

    private:

            char data[256];

            };

string::string(char *str)
```

```
{
            strcpy(data, str);
}
void string::operator +(char *str)
{
            strcat(data, str);
}
void string::operator -(char letter)
{
            char temp[256];
            int i, j;

            for (i=0, j=0; data[i]; i++)
                  if (data[i] != letter)
                              temp[j++] = data[i];
            temp[j] = NULL;
      strcpy(data, temp);
}
void string::show_string(void)
{
            cout << data << endl;
}
void main(void)
{
    string title("Φλώρινα");
    string lesson("Καλαμάτα");
    title.show_string();
    title + " - Τόπος Ολυμπιονικών";// προσθέτει τα δύο αλφαριθμητικά
```

```
    title.show_string();

    lesson.show_string();

    lesson - 'α'; // αφαιρεί το γράμμα α

    lesson.show_string();

}
```

Στατικές Συναρτήσεις και Μέλη Δεδομένων

Υπάρχουν περιπτώσεις που θα θέλουμε τα διαφορετικά αντικείμενα μιας τάξης να κάνουν κοινή χρήση μίας ή περισσότερων μεταβλητών δεδομένων. Για να κάνουμε κοινόχρηστο ένα μέλος δεδομένων, πρέπει να το δηλώσουμε σαν static. Μπορούμε να προσπελάσουμε ένα δημόσιο στατικό μέλος ή και μια δημόσια στατική μέθοδο μιας τάξης μέσα από το πρόγραμμα, ακόμη κι αν δεν έχει δηλωθεί κανένα αντικείμενο αυτής της τάξης.

Αφού ορίσουμε την τάξη που θα περιέχει το κοινόχρηστο μέλος, πρέπει να δηλώσουμε το συγκεκριμένο μέλος και σαν καθολική μεταβλητή έξω από τον ορισμό της τάξης.

Στο παρακάτω παράδειγμα ορίζουμε την τάξη book_series, η οποία περιέχει την κοινόχρηστη μεταβλητή page_count. Αν το πρόγραμμα αλλάξει την τιμή αυτού του μέλους, η αλλαγή γίνεται αμέσως ορατή σ' όλα τα αντικείμενα της τάξης.

```
#include <iostream.h>

#include <string.h>

class book_series {

    public:

            book_series(char *, char *, float);

            void show_book(void);

            void set_pages(int);

    private:

            static int page_count;

            char title[64];

            char author[64];

            float price;

};
```

```
int book_series::page_count; //δήλωση κοινόχρηστου μέλους σαν καθολικής μεταβλητής


void book_series::set_pages(int pages)

{

            page_count = pages;

}

book_series::book_series(char *title, char *author, float price)

{

            strcpy(book_series::title, title);

            strcpy(book_series::author, author);

            book_series::price = price;

}

void book_series::show_book(void)

{

            cout << title << endl;

            cout << author << endl;

            cout << price << endl;

            cout << page_count << endl;

}

void main(void)

{

          book_series programming("Programming in C", "James", 19.2);

          book_series word("Programming in Pascal", "Brown", 20.1);

          word.set_pages(256);

          programming.set_pages(512);

}
```

Η Κληρονομικότητα

Καθώς ορίζουμε τις τάξεις μας, μπορεί να υπάρχουν περιπτώσεις όπου μια νέα τάξη να χρησιμοποιεί πολλά ή και όλα τα χαρακτηριστικά μιας ήδη υπάρχουσας τάξης και μετά να προσθέτει ένα ή περισσότερα μέλη, είτε δεδομένα είτε συναρτήσεις.

Λέμε ότι το νέο αντικείμενο κληρονομεί τα μέλη της υπάρχουσας τάξης, που ονομάζεται βασική τάξη (base class) και η νέα τάξη ονομάζεται παράγωγη τάξη (derived class). Η κληρονομικότητα (inheritance) μπορεί να μας απαλλάξει από προγραμματιστική εργασία και να μας βοηθήσει να κάνουμε σημαντική οικονομία χρόνου.

Κληρονομικότητα λοιπόν λέγεται η δυνατότητα μιας παράγωγης τάξης να κληρονομεί τα χαρακτηριστικά μιας ήδη υπάρχουσας βασικής τάξης. Δηλαδή, αν θέλουμε τα μέλη μιας τάξης (δεδομένα ή συναρτήσεις) να μπορούν να χρησιμοποιούνται από μια νέα τάξη, μπορούμε να ορίσουμε τη νέα τάξη με βάση την προϋπάρχουσα (βασική) τάξη.

Για παράδειγμα, έστω ότι έχουμε μια βασική τάξη employee με τα εξής στοιχεία :

```
class employee {

    public:

            employee(char *, char *, float);

            void show_employee(void);

    private:

            char name[64];

            char position[64];

            float salary;

};
```

Για να δημιουργήσουμε την παράγωγη τάξη manager από τη βασική τάξη employee, γράφουμε τα εξής :

```
class manager : public employee {

    public:

            manager (char *, char *, char *, float, float, int);

            void show_manager(void);

    private:

            float annual_bonus;

            char company_car[64];
```

```
            int stock_options;

};
```

Η λέξη κλειδί public μπροστά από το όνομα της βασικής τάξης employee καθορίζει ότι τα δημόσια μέλη της τάξης employee είναι επίσης δημόσια μέλη στην τάξη manager. Πρέπει να έχουμε υπόψη μας ότι τα ιδιωτικά μέλη μιας βασικής τάξης είναι προσπελάσιμα από την παράγωγη τάξη μόνο μέσω των συναρτήσεων διασύνδεσης και όχι με τον τελεστή τελείας.

Στο επόμενο παράδειγμα, υπάρχει μια βασική τάξη book και ορίζουμε μια παράγωγη τάξη με όνομα library_card :

```
class book {

    public:

            book(char *, char *, int);

            void show_book(void);

    private:

            char title[64];

            char author[64];

            int pages;

};

class library_card : public book {

    public:

            library_card(char *, char *, int, char *, int);

            void show_card(void);

    private:

            char catalog[64];

            int checked_out;

};
```

Ένα προστατευμένο (protected) μέλος μιας βασικής τάξης είναι κάτι μεταξύ ιδιωτικού και δημόσιου μέλους, δηλ. τα αντικείμενα των παράγωγων τάξεων μπορούν να το προσπελάζουν σαν να ήταν δημόσιο, αλλά σ' όλο το υπόλοιπο πρόγραμμα φαίνεται σαν ιδιωτικό.

Στο παρακάτω παράδειγμα, τα μέλη title, author και pages είναι προστατευμένα (protected) και μπορούν να προσπελάζονται από τις παράγωγες τάξεις της τάξης book σαν δημόσια μέλη, αλλά όχι και από το πρόγραμμα.

```
class book {

    public:

            book(char *, char *, int);

            void show_book(void);

    protected:

            char title[64];

            char author[64];

            int pages;

};
```

Στην περίπτωση που ένα όνομα μέλους μιας παράγωγης τάξης είναι ίδιο μ' ένα όνομα μέλους της βασικής τάξης, η C++ χρησιμοποιεί το μέλος της παράγωγης τάξης μέσα στις συναρτήσεις της παράγωγης τάξης και για να προσπελάσουμε το μέλος της βασικής τάξης, πρέπει να χρησιμοποιήσουμε το όνομα της τάξης και τον τελεστή διάκρισης εμβέλειας.

Η Πολλαπλή Κληρονομικότητα

Πολλαπλή κληρονομικότητα (multiple inheritance) έχουμε όταν μια τάξη κληρονομεί χαρακτηριστικά από περισσότερες από μία βασικές τάξεις. Στα επόμενα παραδείγματα βλέπουμε πώς μπορούμε να παράγουμε την τάξη computer από τις βασικές τάξεις computer_screen και mother_board.

```
class computer_screen {

    public:

            computer_screen(char *, long, int, int);

            void show_screen(void);

    private:

            char type[32];

            long colors;

            int x_resolution;

            int y_resolution;

};

class mother_board {

    public:
```

```
            mother_board(int, int, int);

            void show_mother_board(void);

    private:

            int processor;

            int speed;

            int RAM;

};

class computer : public computer_screen, public mother_board {

    public:

 computer(char *, int, float, char *, long, int, int, int, int);

            void show_computer(void);

    private:

            char name[64];

            int hard_disk;

            float floppy;

};
```

Όταν μια παράγωγη τάξη γίνεται βασική τάξη σε μια άλλη τάξη, τότε σχηματίζεται μια αλυσίδα τάξεων (class chain), όπου η τελευταία παράγωγη τάξη κληρονομεί τα χαρακτηριστικά όλων των βασικών τάξεων από τις οποίες παράχθηκε.

Οι Φίλες Τάξεις

Μπορούμε να χαρακτηρίσουμε μια τάξη σαν φίλη (friend) μιας άλλης τάξης, ώστε να μπορεί η φίλη τάξη να προσπελάζει τα ιδιωτικά μέλη της άλλης τάξης. Όπως ξέρουμε ήδη, τα ιδιωτικά μέλη μάς επιτρέπουν να προστατεύουμε τις τάξεις μας και μειώνουν την πιθανότητα λαθών. Γι' αυτό το λόγο, καλό είναι να περιορίζουμε όσο γίνεται τη χρήση των φίλων τάξεων.

Στη C++ μπορούμε να δώσουμε σε μια φίλη τάξη το δικαίωμα να προσπελάζει τα ιδιωτικά μέλη της τάξης στην οποία είναι φίλη και για να δηλώσουμε ότι μια τάξη είναι φίλη μιας άλλης, συμπεριλαμβάνουμε τη λέξη κλειδί friend και το όνομα της φίλης τάξης στον ορισμό της τάξης.

Στο παρακάτω παράδειγμα, μέσα στην τάξη book δηλώνεται ότι η τάξη librarian είναι φίλη και έτσι τα αντικείμενα της τάξης librarian μπορούν να προσπελάζουν άμεσα τα ιδιωτικά μέλη της τάξης book :

```
class book {

    public:

            book(char *, char *, char *);

            void show_book(void);

            friend librarian;

    private:

            char title[64];

            char author[64];

            char catalog[64];

};

class librarian {

    public:

            void change_catalog(book *, char *);

            char *get_catalog(book);};
```

Όταν χρησιμοποιούμε φίλες τάξεις για να προσπελάζουμε ιδιωτικά μέλη άλλων τάξεων, μπορούμε να περιορίσουμε τα μέλη (συναρτήσεις) της φίλης τάξης που θα έχουν πρόσβαση στα ιδιωτικά δεδομένα, χρησιμοποιώντας φίλες συναρτήσεις (friend functions), δηλ. τη λέξη κλειδί friend ακολουθούμενη από ένα πλήρες πρωτότυπο της συνάρτησης, ως εξής :

```
class book {

    public:

            book(char *, char *, char *);

            void show_book(void);

            friend char *librarian::get_catalog(book);

            friend void librarian::change_catalog(book *, char *);

    private:

            char title[64];

            char author[64];

            char catalog[64];

};
```

Χρήση Προτύπων Συναρτήσεων

Στη C++ μπορούμε πολύ εύκολα να χρησιμοποιούμε πρότυπα συνάρτησης (function templates) για να δημιουργούμε γρήγορα συναρτήσεις που έχουν την ίδια δομή αλλά επιστρέφουν διαφορετικούς τύπους τιμών και έχουν διαφορετικούς τύπους παραμέτρων. Ένα πρότυπο συνάρτησης ορίζει μια συνάρτηση χωρίς τύπο δεδομένων, ο οποίος θα καθοριστεί αργότερα από το πρόγραμμα όπου θα χρησιμοποιηθεί η συνάρτηση.

Στο παρακάτω παράδειγμα ορίζουμε ένα πρότυπο για μια συνάρτηση που ονομάζεται max και επιστρέφει τη μεγαλύτερη από δύο τιμές :

```
template<class T> T max(T a, T b)

{

            if (a>b)

                        return(a);

            else

                        return(b);

}
```

Το T παριστάνει τον γενικό τύπο του προτύπου και αφού ορίσουμε το πρότυπο μέσα στο πρόγραμμα, μπορούμε να δηλώσουμε πρωτότυπα συναρτήσεων για κάθε τύπο δεδομένων που θέλουμε. Τα παρακάτω πρωτότυπα δηλώνουν δύο συναρτήσεις τύπου float και int με βάση το πρότυπο συνάρτησης max

```
float max(float, float);

int max(int, int);
```

Όταν η C++ συναντήσει αυτά τα πρωτότυπα συναρτήσεων, αντικαθιστά το T με τον κατάλληλο τύπο και δημιουργεί την αντίστοιχη συνάρτηση με το όνομα του προτύπου. Με τα πρότυπα συναρτήσεων μπορούμε να μειώσουμε τον αριθμό των συναρτήσεων που πρέπει να γράψουμε και μπορούμε να χρησιμοποιούμε το ίδιο όνομα για συναρτήσεις που κάνουν μια συγκεκριμένη λειτουργία, ανεξάρτητα από την επιστρεφόμενη τιμή και τους τύπους των παραμέτρων που χρησιμοποιούν.

Σ' ένα πρότυπο συνάρτησης μπορούν να δηλωθούν και περισσότεροι από ένας τύποι δεδομένων. Οι παρακάτω προτάσεις δημιουργούν ένα πρότυπο για τη συνάρτηση show_array, η οποία εμφανίζει στην οθόνη τα στοιχεία ενός πίνακα. Το T καθορίζει τον τύπο του πίνακα και το T1 τον τύπο του αριθμού στοιχείων.

```
template<class T, class T1> void show_array(T *array, T1 count)

{

            T1 index;

            for (index = 0; index < count; index++)
```

```
                        cout << array[index] < ' ';

                        cout << endl;

}
```

Μπορούμε μετά να δηλώσουμε πρωτότυπα συναρτήσεων, ως εξής :

```
void show_array(int *, int);

void show_array(float *, unsigned);
```

Οι Τελεστές New και Delete

Σ' ένα πρόγραμμα μπορούμε να κατανέμουμε τον χώρο της μνήμης που χρειαζόμαστε κατά τη διάρκεια της εκτέλεσης του προγράμματος και έτσι να μην περιοριζόμαστε μόνο σε πίνακες σταθερού μεγέθους. Το πρόγραμμα καθορίζει την ποσότητα της μνήμης που χρειάζεται και η C++ επιστρέφει έναν δείκτη στη μνήμη και κατανέμει χώρο μνήμης από μια περιοχή που ονομάζεται σωρός (heap).

Με τον τελεστή new μπορούμε να κατανέμουμε χώρο μνήμης κατά την εκτέλεση ενός προγράμματος και να δηλώσουμε έναν δείκτη προς έναν πίνακα. Για παράδειγμα, αν το πρόγραμμα χρειάζεται έναν πίνακα των 50 bytes, γράφουμε τα εξής :

```
char *bugger = new char[50];
```

Δείτε το παρακάτω πρόγραμμα :

```
#include <iostream.h>

void main(void)

{

            char *pointer = new char[100];


            if (pointer != NULL)

                  cout << "Επιτυχής κατανομή μνήμης" << endl;

            else

                  cout << "Σφάλμα στην κατανομή της μνήμης" << endl;

}
```

Αν δεν υπάρχει αρκετή διαθέσιμη μνήμη στον σωρό και ο τελεστής new αποτύχει να κατανείμει την απαιτούμενη μνήμη, ο δείκτης παίρνει την τιμή NULL. Έτσι, εξετάζοντας την τιμή του δείκτη, μπορούμε να διαπιστώσουμε αν ικανοποιήθηκε η απαίτηση μνήμης.

Μπορούμε, όμως, πρώτα να ρωτήσουμε τον χρήστη πόσα bytes χρειάζεται και μετά να κατανέμουμε τον χώρο μνήμης με τον τελεστή new, όπως στο παρακάτω παράδειγμα :

```
#include <iostream.h>

void main(void)

{

            int size;

            char *pointer;

            cout << "Δώστε μέγεθος πίνακα ( < 3000) : ";

            cin >> size;

            if (size <= 3000)

            {

              pointer = new char[size];

               if (pointer != NULL)

                  cout << " Επιτυχής κατανομή μνήμης" << endl;

               else

                  cout << " Σφάλμα στην κατανομή της μνήμης" << endl;

            }

}
```

Όταν δεν χρειαζόμαστε άλλο τη μνήμη που έχει κατανεμηθεί σ' ένα πρόγραμμα, μπορούμε να την αποδεσμεύσουμε με τον τελεστή delete. Απλά προσδιορίζουμε έναν δείκτη προς την περιοχή μνήμης, ως εξής :

```
delete pointer;
```

Περισσότερα για τις Εντολές Cin και Cout

Το αρχείο επικεφαλίδας iostream.h περιέχει ορισμούς που επιτρέπουν στα προγράμματα να χρησιμοποιούν την εντολή cout για εμφάνιση πληροφοριών στην οθόνη και την εντολή cin για είσοδο δεδομένων από το πληκτρολόγιο. Στην ουσία, το αρχείο αυτό ορίζει τις τάξεις istream (ρεύμα εισόδου) και ostream (ρεύμα εξόδου), των οποίων αντικείμενα-μεταβλητές είναι τα cin και cout.

Το cout είναι ένα αντικείμενο τάξης που περιέχει διάφορες μεθόδους. Με τον χειριστή setw μπορούμε να καθορίσουμε τον ελάχιστο αριθμό κενών διαστημάτων που θα καταλαμβάνει η επόμενη τιμή και η συνάρτηση cout. width καθορίζει τον ελάχιστο αριθμό χαρακτήρων που θα χρησιμοποιεί η επόμενη τιμή εξόδου.

```
cout << "Ο αριθμός είναι ο : " setw(3) << 1001 << endl;

cout.width(6);

cout << "Ο αριθμός είναι ο : " << 1001 << endl;
```

Με τη συνάρτηση cout.fill μπορούμε να καθορίσουμε τον χαρακτήρα που θα χρησιμοποιεί η cout για να συμπληρώσει τον κενό χώρο.

```
cout.fill('.');

cout << "Ο αριθμός είναι ο : " setw(10) << 1001 << endl;
```

Με τον χειριστή setprecision μπορούμε να καθορίσουμε τον αριθμό των δεκαδικών ψηφίων που θέλουμε να εμφανίζονται.

setprecision(3);

Με τη συνάρτηση cout.put μπορούμε να εμφανίζουμε μεμονωμένους χαρακτήρες εξόδου.

```
cout.put(string[i]);
```

Με τη συνάρτηση cin.get μπορούμε να διαβάσουμε μεμονωμένους χαρακτήρες εισόδου. Απλά αποδίδουμε τον χαρακτήρα που επιστρέφει η συνάρτηση σε μια μεταβλητή τύπου χαρακτήρα, ως εξής :

```
letter = cin.get();
```

Με τη συνάρτηση cin.getline μπορούμε να διαβάσουμε μια ολόκληρη γραμμή δεδομένων και να την τοποθετήσουμε σ' ένα αλφαριθμητικό. Καθορίζουμε το αλφαριθμητικό στο οποίο θα τοποθετηθούν οι χαρακτήρες και το μέγεθος του αλφαριθμητικού, ως εξής :

```
cin.getline(string, 64);

cin.getline(string, sizeof(string));
```

Η επόμενη κλήση της συνάρτησης την κάνει να διαβάσει μια γραμμή κειμένου και να τερματιστεί όταν συναντήσει επαναφορά κεφαλής ή όταν διαβάσει 64 χαρακτήρες ή όταν συναντήσει το γράμμα 'α'.

```
cin.getline(string, 64, 'α');
```

Τα Αρχεία στη C++

Στο αρχείο επικεφαλίδας fstream.h ορίζεται η τάξη ofstream με τα αντικείμενα της οποίας μπορούμε να πραγματοποιούμε έξοδο σε αρχεία. Στην αρχή, δηλώνουμε ένα αντικείμενο της τάξης ofstream, δίνοντας το όνομα αρχείου εξόδου σε μορφή αλφαριθμητικού.

Όταν καθορίζουμε όνομα αρχείου με μια δήλωση αντικειμένου τύπου ofstream, η C++ διαγράφει το τυχόν υπάρχον αρχείο του δίσκου που έχει το ίδιο όνομα. Το επόμενο πρόγραμμα δημιουργεί ένα αντικείμενο τύπου ofstream και μετά χρησιμοποιεί τον τελεστή εισαγωγής << για να γράψει μερικές γραμμές κειμένου στο αρχείο.

```
#include <fstream.h>

void main(void)

{

        ofstream book_file("BOOKINFO.DAT");

        book_file << "Η Επικοινωνία στο Χθες και το Σήμερα" << endl;

       book_file << "Εκδόσεις Πατάκη" << endl;

}
```

Το παραπάνω πρόγραμμα ανοίγει το αρχείο BOOKINFO.DAT και μετά γράφει σ' αυτό δύο γραμμές κειμένου.

Για να κάνουμε λειτουργίες εισόδου από αρχεία, χρησιμοποιούμε αντικείμενα τύπου ifstream. Απλά δημιουργούμε ένα αντικείμενο, μεταβιβάζοντας σαν παράμετρο το όνομα του αρχείου. Το επόμενο πρόγραμμα ανοίγει το αρχείο BOOKINFO.DAT και διαβάζει και εμφανίζει στην οθόνη τις δύο πρώτες καταχωρήσεις του αρχείου.

```
#include <iostream.h>

#include <fstream.h>

void main(void)

{

             ifstream input_file("BOOKINFO.DAT");

             char on[64], two[64];

             input_file >> one;

             input_file >> two;

             cout << one << endl;

             cout << two << endl;

}
```

Τα αντικείμενα τύπου istream μπορούν να χρησιμοποιήσουν τη συνάρτηση getline για να διαβάσουν μια γραμμή εισόδου από ένα αρχείο και τη συνάρτηση get για να διαβάσουν έναν χαρακτήρα από ένα αρχείο.

```
             input_file.getline(one, sizeof(one));

             letter = input_file.get();
```

Για να ελέγχουμε πότε έχουμε φθάσει στο τέλος ενός αρχείου, μπορούμε να χρησιμοποιούμε τη συνάρτηση eof του αντικειμένου, που επιστρέφει την τιμή 0 αν δεν βρέθηκε το τέλος του αρχείου και την τιμή 1 αν βρέθηκε. Το επόμενο πρόγραμμα χρησιμοποιεί τη συνάρτηση eof για να διαβάσει τα περιεχόμενα του αρχείου BOOKINFO.DAT μέχρι να συναντήσει το τέλος του αρχείου.

```
#include <iostream.h>

#include <fstream.h>

void main(void)

{

            ifstream input_file("BOOKINFO.DAT");

            char line[64];

            while (!input_file.eof())

            {

                        input_file.getline(line, sizeof(line));

                        cout << line << endl;

            }

}
```

Για να μπορούμε να ελέγχουμε αν έχουν συμβεί λάθη κατά την ανάγνωση από ή την εγγραφή δεδομένων σε αρχείο, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση μέλος fail ενός αντικειμένου τύπου αρχείου. Αν δεν συμβεί κανένα λάθος κατά τη διάρκεια μιας λειτουργίας αρχείου, η συνάρτηση θα επιστρέψει τιμή 0 (ψευδή), ενώ αν συμβεί κάποιο λάθος, θα επιστρέψει τιμή αληθή.

```
if (input_file.fail())

{

            ...

}
```

Για να κλείσουμε ένα αρχείο, πρέπει να χρησιμοποιήσουμε τη συνάρτηση close, ως εξής :

```
input_file.close();
```

Για να μπορούμε να διαβάζουμε και να γράφουμε πίνακες και δομές (structs) από και προς αρχεία, πρέπει να χρησιμοποιήσουμε τις συναρτήσεις read και write. Με τις συναρτήσεις αυτές καθορίζουμε μια περιοχή προσωρινής αποθήκευσης (buffer), στην οποία τοποθετούνται τα δεδομένα που διαβάζονται καθώς και το μέγεθος της περιοχής προσωρινής αποθήκευσης σε bytes, ως εξής :

```
input_file.read(buffer, sizeof(buffer));

output_file.write(buffer, sizeof(buffer));
```

Το επόμενο πρόγραμμα χρησιμοποιεί τη συνάρτηση write για έξοδο των περιεχομένων μιας δομής στο αρχείο EMPLOYEE.DAT :

```
#include <iostream.h>

#include <fstream.h>

void main(void)

{

            struct employee {

                        char name[64];

                        int age;

                        float salary;

            } worker = {"Παπαδόπουλος Ιωάννης", 33, 300000.0};

            ofstream emp_file("EMPLOYEE.DAT");

            emp_file.write((char *) &worker, sizeof(employee));

}
```

Το επόμενο πρόγραμμα χρησιμοποιεί τη συνάρτηση read για να διαβάσει τις πληροφορίες από το αρχείο EMPLOYEE.DAT :

```
#include <iostream.h>

#include <fstream.h>

void main(void)

{

            struct employee {

                        char name[64];

                        int age;

                        float salary;

            } worker;

            ifstream emp_file("EMPLOYEE.DAT");

            emp_file.read((char *) &worker, sizeof(employee));
```

```
        cout << worker.name << endl;

        cout << worker.age << endl;

        cout << worker.salary << endl;

}
```

Η Λέξη Κλειδί Inline

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.

Μ' αυτόν τον τρόπο, το πρόγραμμα συνεχίζει να είναι ευανάγνωστο επειδή χρησιμοποιεί συναρτήσεις, αλλά έχει και βελτιωμένη απόδοση επειδή έχει απαλλαγεί από τις κλήσεις των συναρτήσεων και εκτελείται ταχύτερα.

```
inline int max(int a, int b)

{

        if (a > b)

                return(a);

        else

                return(b);

}
```