2018

Τμήμα Μηχανικών Πληροφορικής
ΤΕΙ ΑΜΘ

Ελευθέριος Μωυσιάδης

[ΕΙΣΑΓΩΓΗ ΣΤΗΝ C++]

Σημειώσεις για το ομώνυμο μάθημα α’ εξαμήνου του τμήματος Μηχανικών Πληροφορικής του ΤΕΙ ΑΜΘ

Περιεχόμενα

1 Βασικές έννοιες

Περίληψη: Στην ενότητα αυτή ορίζονται οι ακόλουθες **θεμελιώδεις** και συχνά παρεξηγημένες έννοιες : Εμβέλεια (scope), Χρόνος ζωής (lifetime), τοπικές (local), καθολικές (global) και στατικές (static) μεταβλητές.

Λέξεις κλειδιά**:** *Εμβέλεια (scope), Χρόνος ζωής (lifetime), τοπικές (local) μεταβλητές, καθολικές (global) μεταβλητές, στατικές (static) μεταβλητές.*

Ας ξεκινήσουμε την συζήτησή μας αποσαφηνίζοντας την απλή έννοια του **μπλοκ**.

Ορισμός 1: Μπλοκ κώδικα (Code Block) ή απλά μπλοκ ονομάζεται ένα τμήμα πηγαίου κώδικα που αρχίζει με μια αριστερή αγκύλη "{" και τελειώνει με μια δεξιά αγκύλη "}".

```
1 #include <iostream>
2
3 using namespace std ;
4
5 int main () {
6     cout << "Hello world!" << endl ;
7     return 0 ;
8 }
```

Κώδικας 1: Η main συνιστά ένα μπλοκ.

Ένα παράδειγμα μπλοκ δίνεται στον κώδικα 1 όπου η `main` συνιστά ένα μπλοκ. Είναι φανερό πως ο ορισμός (definition)[1] κάθε συνάρτησης συνιστά ένα μπλοκ.

Ένα μπλοκ που περιέχει άλλα μπλοκ ονομάζεται **σύνθετο** μπλοκ (**nested** block ή **compound** block). Το μπλοκ είναι επίσης γνωστό και ως σύνθετη **εντολή** (compound **statement**). Στην συνέχεια, παρουσιάζεται ένα σύνθετο μπλοκ.

```
1 void f(bool isMorning) {
2
3   if (isMorning) {
4      cout << "GoodMorning!" ;
5      prepareTea ();
6      openWindows ();
7   }
8   else cout << "GoodNight!" ;
9  }
```

Κώδικας 2 : Η συνάρτηση f συνιστά ένα σύνθετο μπλοκ.

[1] Ο ορισμός (definition) μιας συνάρτησης διαφέρει από την δήλωση (declaration) της συνάρτησης. Η δήλωση της συνάρτησης περιλαμβάνει τιμή επιστροφής, όνομα και λίστα παραμέτρων. Ο ορισμός της συνάρτησης περιλαμβάνει την δήλωση της μαζί με το σώμα της, δηλ. τον πραγματικό κώδικα της συνάρτησης. Η δήλωση της συνάρτησης μπορεί να επαναλαμβάνεται σε διαφορετικά μέρη ενός κώδικα ενώ ο ορισμός της είναι μόνον ένας.

Η συνάρτηση f() στον κώδικα 2 περιέχει το μπλοκ που αρχίζει στην γραμμή 3 και τελειώνει στην γραμμή 7. Επομένως, η f() συνιστά ένα σύνθετο μπλοκ.

Ορισμός 2: **Τοπική** ονομάζεται μια μεταβλητή (**local** variable) όταν δηλώνεται (declare) στο εσωτερικό ενός μπλοκ.

```
1 #include <iostream>
2 using namespace std ;
3 int step = 5 ;
4
5 void f () {
6   int k = 0 ;
7   for ( int i = 0 ; i < 5 ; i ++) {
8     k += step ;
9     cout << k << endl ;
10  }
11 }
12
13 int main () {
14  f ();
15  return 0 ;
16 }
```

Κώδικας 3: Τοπικές και καθολικές μεταβλητές.

Σύμφωνα με τον ορισμό 2 είναι φανερό πως στον κώδικα 3, η `k` είναι τοπική στην `f()` και η `i` είναι τοπική στην **`for`**. Επίσης, η `step` δεν είναι τοπική μεταβλητή. Μια τέτοια μεταβλητή ονομάζεται **καθολική** σύμφωνα με τον ορισμό 3.

Ορισμός 3: Καθολική ονομάζεται μια μεταβλητή (**global** variable) που δηλώνεται έξω από κάθε μπλοκ.

Ορισμός 4: Εμβέλεια (**scope**) μιας μεταβλητής ονομάζεται η περιοχή του πηγαίου κώδικα μέσα στην οποία η μεταβλητή αναγνωρίζεται από τον μεταγλωττιστή.

Στον κώδικα 3, η **εμβέλεια** της `k` είναι η περιοχή από την δήλωσή της μέχρι το τέλος του μπλοκ μέσα στο οποίο αυτή δηλώνεται, δηλ. μέχρι το τέλος της συνάρτησης `f()`. Πράγματι, αν προσπαθήσετε να αναφερθείτε στην `k` εκτός της `f()`, π.χ. στην `main`, ο μεταγλωττιστής θα αποκριθεί με ένα μήνυμα του τύπου : ***'k' was not declared in this scope***. Αυτό ισχύει γενικότερα για τις τοπικές μεταβλητές, δηλ. η εμβέλεια των τοπικών μεταβλητών περιορίζεται στο μπλοκ στο οποίο δηλώθηκαν. Αντίθετα, η καθολική μεταβλητή `step` έχει εμβέλεια την περιοχή από την δήλωσή της μέχρι το τέλος του προγράμματος. Γενικότερα, η εμβέλεια μιας καθολικής μεταβλητής εκτείνεται από την δήλωσή της μέχρι το τέλος του προγράμματος. Ωστόσο, μια καθολική μεταβλητή δεν είναι άμεσα **ορατή (visible)** σε μπλοκ στο οποίο έχει ορισθεί τοπική μεταβλητή με ίδιο **αναγνωριστικό (identifier)** **[1]**. Για παράδειγμα, στον κώδικα 4 έχει δηλωθεί μια καθολική ακέραιη μεταβλητή `g`. Επίσης, με το ίδιο αναγνωριστικό (`g`) έχει δηλωθεί μια τοπική ακέραιη στην `f()`. Ως αποτέλεσμα οι αναφορές στην `g` μέσα στην `f()` αφορούν την

τοπική και όχι την καθολική μεταβλητή. Για αυτόν τον λόγο λέμε πως η τοπική μεταβλητή **αποκρύπτει (hides)** την καθολική. Αν παρόλα αυτά χρειαζόμαστε μέσα στην f να αναφερθούμε στην καθολική `g` τότε θα πρέπει να την χαρακτηρίσουμε με χρήση του **τελεστή καθολικής εμβέλειας (global scope operator)** που αναπαρίσταται με το σύμβολο ::. Για παράδειγμα, η αναφορά `::g` (γραμμή 7) στην `f()` αφορά στην καθολική `g` και όχι στην τοπική. Επιπλέον θα πρέπει να σημειωθεί πως η συζήτηση για την ορατότητα των μεταβλητών ολοκληρώνεται στην ενότητα 5.10, όπου παρουσιάζονται οι **ονοματοχώροι (namespaces)**[2]. Τέλος, θα πρέπει να πούμε πως η απόκρυψη μεταβλητών διευκολύνει την ανάπτυξη των προγραμμάτων καθώς μας επιτρέπει να εργαζόμαστε χωρίς να ενδιαφερόμαστε για τις καθολικές μεταβλητές που δεν χρησιμοποιούνται στην τοπική εμβέλεια.

```
1  #include <iostream>
2  using namespace std ;
3  int g = 2 ;
4
5  void f() {
6   int g = 10 ;
7   cout << g << " " << ::g << endl ;
8  }
9
10 int main () {
11  cout << g << endl ;
12  f();
13  cout << g << endl ;
14 }
```

Κώδικας 4: Η καθολική μεταβλητή g κρύβεται στην f από την τοπική g

Απόκρυψη πληροφορίας (Information hiding): Η απόκρυψη μιας καθολικής μεταβλητής από μια τοπική αποτελεί έναν μόνο από τους μηχανισμούς μιας γενικότερης διαδικασίας γνωστής ως απόκρυψης πληροφορίας. Η απόκρυψη πληροφορίας συνιστά βασική αρχή του σύγχρονου λογισμικού και αφορά στην απόκρυψη κυρίως λεπτομερειών της υλοποίησης ενός τμήματος κώδικα το οποίο όμως αναγκαστικά διαθέτει μια **διεπαφή (interface)** ώστε να μπορεί να χρησιμοποιηθεί. Χαρακτηριστικό παράδειγμα συνιστά η οργάνωση μιας βιβλιοθήκης συναρτήσεων σε header και source αρχεία κώδικα. Στα αρχεία header παρέχονται οι διεπαφές των συναρτήσεων και στα αρχεία source οι ορισμοί των συναρτήσεων. Έτσι, στον χρήστη της εν λόγω βιβλιοθήκης είναι δυνατόν να δοθούν τα αρχεία header και τα αντικειμενικά (.obj) αρχεία των sources. Με αυτόν τον τρόπο, ο χρήστης δεν απαιτείται να ασχοληθεί με τις λεπτομέρειες της εσωτερικής οργάνωσης της βιβλιοθήκης παρά μόνο με τις διεπαφές των συναρτήσεων ώστε να είναι σε θέση να κάνει σωστές κλήσεις. Επιπλέον, ο κατασκευαστής της βιβλιοθήκης δεν υποχρεώνεται να παραδώσει στον χρήστη τους ορισμούς των συναρτήσεων που συνήθως αποτελούν περιουσιακό του στοιχείο και επιθυμεί να το προστατεύσει.

[2] Οι ονοματοχώροι συνιστούν ένα μηχανισμό που χρησιμοποιείται για την αποφυγή συγκρούσεων “ονομάτων”, δηλ. αναγνωριστικών μεταβλητών ή υπογραφών (signature) συναρτήσεων.

Σημείωση 1: Οι παράμετροι μιας συνάρτησης λειτουργούν ως τοπικές μεταβλητές. Επομένως, η εμβέλεια των παραμέτρων μιας συνάρτησης εκτείνεται σε όλη την συνάρτηση.

Σημείωση 2: Οι **μεταβλητές ελέγχου (control variables)** της **`for`** που δηλώνονται στην **`for`** έχουν εμβέλεια το μπλοκ ή την απλή εντολή που επαναλαμβάνεται από την **`for`**. Για παράδειγμα, στον κώδικα 3, η εμβέλεια της i εκτείνεται μέχρι την γραμμή 10.

Μια έννοια σχετική με την εμβέλεια που επίσης μας ενδιαφέρει και είναι πολύ σημαντική είναι ο **χρόνος ζωής** μιας μεταβλητής.

Ορισμός 5: Ο χρόνος ζωής (**lifetime**) μιας μεταβλητής είναι το διάστημα που εκτείνεται από την δημιουργία της μέχρι την καταστροφή της.

Στον κώδικα 3, ο χρόνος ζωής της `step` εκτείνεται από την δημιουργία της κατά την εκτέλεση της γραμμής 3 μέχρι την καταστροφή της κατά τον τερματισμό του προγράμματος. Αντίστοιχα, ο χρόνος ζωής της k εκτείνεται από την δημιουργία της κατά την εκτέλεση της γραμμής 6 μέχρι την καταστροφή της που συμβαίνει κατά την επιστροφή της `f()`. Παρομοίως, ο χρόνος ζωής της i εκτείνεται από την δημιουργία της που συμβαίνει με την είσοδο στην επαναληπτική διαδικασία (γραμμή 7) έως την καταστροφή της κατά την έξοδο της επαναληπτικής διαδικασίας (γραμμή 10).

Εύκολα δημιουργείται η εντύπωση ότι η εμβέλεια μιας μεταβλητής και ο χρόνος ζωής της είναι ταυτόσημες έννοιες. Ωστόσο, κάτι τέτοιο δεν είναι αληθές. Καταρχάς, ο χρόνος ζωής ορίζεται στο πλαίσιο του **χρόνου εκτέλεσης (run time)**· αντίθετα, η εμβέλεια ορίζεται στο πλαίσιο του **χρόνου μεταγλώττισης (compile time)**. Παρόλα αυτά με βάση τα όσα έχουμε πει μέχρι αυτό το σημείο φαίνεται ότι μια μεταβλητή έχει ζωή μόνον κατά την εκτέλεση του κώδικα που προσδιορίζει την εμβέλειά της. Κάτι τέτοιο αληθεύει για τις μεταβλητές που έχουμε δει μέχρι εδώ και ονομάζονται **μη στατικές (non-statics)** μεταβλητές. Όχι όμως και για τις **στατικές (statics)** μεταβλητές. Μια στατική μεταβλητή βρίσκεται “εν ζωή” παρότι αθέατη και κατά την εκτέλεση κώδικα εκτός εμβέλειας της. Πιο συγκεκριμένα, η εμβέλεια των στατικών τοπικών μεταβλητών ορίζεται όμοια με την εμβέλεια των μη στατικών τοπικών μεταβλητών από το μπλοκ στο οποίο αυτές δηλώνονται. Ο χρόνος ζωής όμως των στατικών τοπικών μεταβλητών ορίζεται διαφορετικά από τον χρόνο ζωής των μη στατικών τοπικών. Πιο συγκεκριμένα, ο χρόνος ζωής των στατικών τοπικών μεταβλητών εκτείνεται από την δημιουργία τους ως το τέλος της εκτέλεσης του προγράμματος. Επομένως, η τιμή των τοπικών στατικών μεταβλητών διατηρείται και όταν αυτές βγαίνουν εκτός εμβέλειας. Ας σημειωθεί πως η συζήτηση μέχρι εδώ αφορά μόνο τοπικές στατικές μεταβλητές. Στην τέλος αυτής της ενότητας παρουσιάζονται οι καθολικές στατικές μεταβλητές και **συναρτήσεις**.

Από συντακτική άποψη μια μεταβλητή ή συνάρτηση δηλώνεται ως στατική με χρήση της **λέξης-κλειδί (keyword)** **`static`**.

```
1 #include <iostream>
2 using namespace std ;
3
4 void increment () {
5     static int k = 0 ;
```

```
6       k ++;
7       cout << k << endl ;
8  }
9
10 int main () {
11      increment ();
12      //cout << k (out of scope)
13      int i=0;
14      increment ();
15      increment ();
16      return 0 ;
17 }
```

Κώδικας 5: Ο χρόνος ζωής της στατικής μεταβλητής.

Στον κώδικα 5, ορίζεται η τοπική στατική μεταβλητή `k` μέσα στην `increment`. Αν επιχειρήσουμε να προσπελάσουμε την `k` εκτός εμβέλειας, π.χ. αν εισάγουμε στην γραμμή 12 μια αναφορά στην `k` ο μεταγλωττιστής θα διαμαρτυρηθεί με το γνωστό μήνυμα : ***'k' was not declared in this scope***. Ωστόσο, κατά την εκτέλεση της γραμμής 13 η `k` υφίσταται, καταλαμβάνοντας χώρο στην μνήμη ίσο με ένα ακέραιο. Αυτό έχει ως αποτέλεσμα κατά την δεύτερη, τρίτη, κλπ κλήση της `increment`() να μην απαιτείται να ξαναδημιουργηθεί η `k`. Με άλλα λόγια **η γραμμή 5 εκτελείται μόνο κατά την πρώτη κλήση της increment()**. Ως αποτέλεσμα, η έξοδος του κώδικα 4 είναι :

```
1
2
3
```

Γενικότερα, η **αρχικοποίηση** που ενδεχομένως λαμβάνει χώρα κατά την **δημιουργία** (construction) μιας τοπικής **στατικής** μεταβλητής **δεν επαναλαμβάνεται σε διαδοχικές κλήσεις του περιβάλλοντος μπλοκ**. Αντίθετα, **αρχικοποίηση στατικής** τοπικής μεταβλητής μέσω **εκχώρησης επαναλαμβάνεται** με κάθε κλήση του περιβάλλοντος μπλοκ [2].

```
1  #include <iostream>
2  using namespace std;
3
4  void increment () {
5       static int k;
6       k = 0;
7       k ++;
8       cout << k << endl ;
9  }
10
11 int main () {
12      increment ();
13      increment ();
14      increment ();
15      return 0 ;
16 }
```

Κώδικας 6: Αρχικοποίηση στατικής με εκχώρηση.

Σύμφωνα με τα παραπάνω, η έξοδος του κώδικα 6 είναι:

1
1
1

Επομένως, η δημιουργία της στατικής μεταβλητής συμβαίνει 1 φορά κατά την πρώτη κλήση της γραμμής στην οποία αυτή δηλώνεται ενώ η εκχώρηση επαναλαμβάνεται σε κάθε κλήση.

Καθολικές στατικές μεταβλητές και συναρτήσεις : Η λέξη-κλειδί static έχει και 2 άλλες χρήσεις. Η μια αφορά τα στατικά μέλη κλάσης και είναι η σημαντικότερη χρήση της static. Η χρήση αυτή θα συζητηθεί εκτενώς στα πλαίσια του Αντικειμενοστραφούς προγραμματισμού. Η δεύτερη χρήση αφορά στον χαρακτηρισμό ως στατικής μιας καθολικής μεταβλητής. Μια μη στατική καθολική μεταβλητή δεν διαφέρει από μια στατική καθολική ως προς τον χρόνο ζωής, διαφέρει όμως ως προς την εμβέλεια. Η εμβέλεια της στατικής περιορίζεται σε επίπεδο αρχείου. Πιο συγκεκριμένα, στην περίπτωση προγραμμάτων που αποτελούνται από πολλά αρχεία πηγαίου κώδικα, η εμβέλεια της στατικής καθολικής περιορίζεται στο αρχείο που δηλώθηκε. Με άλλα λόγια, η στατική καθολική μεταβλητή δεν είναι ορατή (visible) σε άλλα αρχεία από αυτό που δηλώθηκε. Με αυτήν την έννοια μπορούν να δηλωθούν και στατικές συναρτήσεις. Οι στατικές συναρτήσεις, επίσης, δεν είναι ορατές σε άλλο αρχείο από αυτό που δηλώθηκαν.

1.1 Ερωτήσεις πολλαπλής επιλογής

1. Ένα μπλοκ κώδικα που περιλαμβάνει άλλα μπλοκ ονομάζεται
    - πολλαπλή εντολή
    - πολύπλοκη εντολή
    - σύνθετο μπλοκ
2. Μια τοπική μεταβλητή δηλώνεται
    - στο εσωτερικό της main
    - έξω από κάθε μπλοκ
    - στο εσωτερικό ενός οποιουδήποτε μπλοκ
3. Μια καθολική μεταβλητή δηλώνεται
    - στο εσωτερικό της main
    - έξω από κάθε μπλοκ
    - στο εσωτερικό ενός οποιουδήποτε μπλοκ
4. Μια τοπική μεταβλητή είναι αναγνωρίσιμη από τον μεταγλωττιστή
    - καθόλη την διάρκεια του χρόνου ζωής της
    - μέσα στο μπλοκ στο οποίο δηλώνεται
    - από την δήλωσή της μέχρι το τέλος του μπλοκ στο οποίο δηλώνεται
    - από την αρχή του μπλοκ στο οποίο δηλώνεται μέχρι την δήλωσή της
5. Μια καθολική μεταβλητή αποκρύπτεται σε τοπικό επίπεδο από μια τοπική μεταβλητή όταν αυτές έχουν
    - το ίδιο όνομα
    - την ίδια εμβέλεια
    - τον ίδιο χρόνο ζωής
6. Προκειμένου να προσπελάσουμε μια καθολική μεταβλητή που αποκρύπτεται από μια τοπική χρησιμοποιούμε
    - τον τελεστή προσπέλασης
    - τον τελεστή καθολικής εμβέλειας
7. Το διάστημα από την δημιουργία μέχρι την καταστροφή μιας μεταβλητής ονομάζεται
    - εμβέλεια
    - χρόνος ζωής
8. Οι στατικές συναρτήσεις
    - δεν είναι ορατές σε άλλο αρχείο από αυτό που δηλώνονται
    - δεν είναι ποτέ καθολικές

1.2 Ερωτήσεις Σωστού-Λάθους

	Σ	Λ
Μια μεταβλητή που δηλώνεται μέσα σε περισσότερα από ένα μπλοκ ονομάζεται καθολική		

Η εμβέλεια μιας μεταβλητής ορίζεται στο πλαίσιο του χρόνου εκτέλεσης (run time)		
Οι μη στατικές μεταβλητές χαρακτηρίζονται από την λέξη-κλειδί non-static		
Η αρχικοποίηση που ενδεχομένως λαμβάνει χώρα κατά την δημιουργία (construction) μιας τοπικής στατικής μεταβλητής δεν επαναλαμβάνεται σε διαδοχικές κλήσεις του περιβάλλοντος μπλοκ		
Μια μη στατική καθολική μεταβλητή δεν διαφέρει από μια στατική καθολική ως προς τον χρόνο ζωής.		
Μια μη στατική καθολική μεταβλητή δεν διαφέρει από μια στατική καθολική ως προς την εμβέλεια.		

1.3 Ασκήσεις

1. Ο κώδικας που ακολουθεί παρουσιάζει σχεδόν όλα τα θέματα που αναπτύχθηκαν σε αυτήν την ενότητα. Εντοπίστε και σχολιάστε κάθε θέμα. Συγκρίνετε την καθολική non static με την καθολική static μεταβλητή. Ποια είναι τα πλεονεκτήματα της μιας και ποια της άλλης;

```
#include <iostream>
using namespace std;
int gInt=0;
static int sGInt=0;

void tst() {
  int lInt=0;
  static int sLInt=10;
  static int sLInt2;
  sLInt2=10;
  lInt++;
  sLInt++;
  gInt++;
  sGInt++;
  sLInt2++;

  cout << "local int          " << lInt << endl
       << "static local int   " << sLInt << endl
       << "static local int2  " << sLInt2 << endl
       << "global int         " << gInt << endl
       << "static global int  " << sGInt << endl
       << endl;
```

```
}

void tst2 () {
  // lInt++;  compiler error
  // sLInt++; compiler error
  gInt++;
  sGInt++;
}

void tst3() {
  int lInt=0;//Αυτή η lInt δεν σχετίζεται με την
             // lInt της tst
  for (int k=0; k<3; k++) {
    lInt=lInt+1;
    std::cout << lInt;
  }
}

int main() {
     for (int i=0; i<3; i++) tst();
}
```

2. Να αναπτυχθεί συνάρτηση **autoCount** που τυπώνει τον αύξοντα αριθμό κλήσης της, δηλ. κατά την πρώτη κλήση της τυπώνει το 1, κατά την δεύτερη κλήση της το 2, κοκ. Η συνάρτηση να βασισθεί σε καθολικό ακέραιο μετρητή. Να επιδειχθεί γιατί η μέθοδος αυτή είναι προβληματική. Να δοθεί άλλη λύση (**autoCount2**) που να υπερβαίνει το πρόβλημα της **autoCount**.
   Λύση σε Βίντεο

3. Να αναπτυχθεί συνάρτηση bool isTheLargerUntilNow(int k) που επιστρέφει true εφόσον η τιμή της παραμέτρου της είναι μεγαλύτερη από την τιμή που είχε η παράμετρός της σε όλες τις προηγούμενες κλήσεις. Θεωρείστε ότι κατά την πρώτη κλήση, κάθε τιμή της k μεγαλύτερη από τον ελάχιστο ακέραιο έχει σαν αποτέλεσμα να επιστρέφει η συνάρτηση true.

2 Παράμετροι

Περίληψη: Στην ενότητα αυτή συζητούνται θεωρητικά και αποσαφηνίζονται με ασκήσεις οι ακόλουθες βασικές και συχνά παρεννοημένες έννοιες :Τυπικές (formal) και πραγματικές (actual) παράμετροι, παράμετροι τιμής (value) και αναφοράς (reference), σταθερές (const) παράμετροι, επιστροφή με τιμή (return by value), επιστροφή με αναφορά (return by reference), παράμετροι με προκαθορισμένες τιμές (parameters with default values)

Λέξεις κλειδιά: *δήλωση (declaration), ορισμός (definition), τυπικές παράμετροι (formal parameters), πραγματικές παράμετροι (actual parameters), παράμετροι τιμής (value parameters), παράμετροι αναφοράς (reference parameters), παράμετρος μόνο για ανάγνωση* ***(read-only parameter)****, σταθερή παράμετρος αναφοράς* ***(read-only reference parameter)***

Στην αρχή του κώδικα που ακολουθεί δίνεται η **δήλωση (declaration)** της gElement. Στο τέλος του ίδιου κώδικα δίνεται ο **ορισμός της (definition)**. Εάν σχολιάσετε την δήλωση, θα διαπιστώσετε ότι ο κώδικας δεν μεταγλωττίζεται. Αυτό οφείλεται στο ότι ο μεταγλωττιστής συναντά την κλήση της (στην tst1) πριν τον ορισμό της. Αν αντίθετα, σχολιάσετε τον ορισμό της, θα διαπιστώσετε πως ο κώδικας περνάει από τον μεταγλωττιστή όχι όμως και από τον **Linker**. Βεβαίως στον Linker μπορεί να δοθεί ο ορισμός της gElement υπό μορφή **object αρχείου**[3] (.obj). Με αυτόν τον τρόπο επιτυγχάνεται η διανομή βιβλιοθηκών συναρτήσεων χωρίς την διανομή του πηγαίου κώδικα. Όταν δεν υπάρχει διαφορετική ανάγκη, δήλωση και ορισμός δίνονται ταυτόχρονα, π.χ. getElement.

Οι παράμετροι στην δήλωση και ορισμό μιας συνάρτησης ονομάζονται **τυπικές παράμετροι (formal parameters)**. Έτσι, η λίστα των τυπικών παραμέτρων της gElement είναι η (int tbl[], int idx). Οι παράμετροι που περνάμε σε μια συνάρτηση κατά την κλήση της ονομάζονται **πραγματικές παράμετροι (actual parameters),** π.χ. οι πραγματικές παράμετροι και των δύο κλήσεων της gElement στην tst1 είναι η tbl και το 2.

```
#include <iostream>
using namespace std;

int gElement(int tbl[], int);
int& getElement(int tbl[], int idx) {return tbl[idx];}

void tst1() {
  int tbl[]={1,2,3,4,5,6,7,8,9};

  cout << gElement(tbl,2)    << endl
       << getElement(tbl,2) << endl;
  //gElement(tbl,2)=13; compile time error, lvalue required
```

[3] Είναι τα αρχεία που παράγει ο μεταγλωττιστής. Τα object αρχεία αποτελούν είσοδο για τον Linker, ο οποίος " συνδέει" μια σειρά από object αρχεία για να παράγει το εκτελεστέο πρόγραμμα (executable, .exe)

```
  getElement(tbl,2)=13;
  cout << endl
       << gElement(tbl,2)    << endl
       << getElement(tbl,2) << endl;
}
void inc(int i) {i++;}
void inc1(int& i) {i++;}

void tst2() {
  int i=0;
  cout << i << endl;
  inc(i);
  cout << i << endl;
  inc1(i);
  cout << i << endl;
}

//int gElement(int tbl[], int idx) {return tbl[idx];}
```

2.1 Τύποι Παραμέτρων

Στην C++ υποστηρίζονται δύο κύριοι τύποι παραμέτρων: οι **παράμετροι τιμής (value parameters)** και οι **παράμετροι αναφοράς (reference parameters)**. Αντίστοιχη ορολογία αναφέρεται στο πέρασμα παραμέτρου με την τιμή της **(pass by value)** και στο πέρασμα παραμέτρου με την αναφορά της **(pass by reference)**. Συντακτικά, οι παράμετροι αναφοράς καθορίζονται από την παρουσία στην λίστα τυπικών παραμέτρων του τελεστή αναφοράς **(reference ή address of operator) &**. Οι παράμετροι τιμής προσδιορίζονται αντίστοιχα από την απουσία του τελεστή αναφοράς, π.χ. η παράμετρος της inc που παρουσιάζεται στον κώδικα προηγουμένως είναι τιμής ενώ η παράμετρος της inc1 είναι αναφοράς. Στην **συνάρτηση κλήσης (calling function)**, περνάει η διεύθυνση (ένα αντίγραφο της διεύθυνσης) της πραγματικής παραμέτρου όταν αυτή είναι δηλωμένη ως παράμετρος αναφοράς. Αντίθετα, στην παράμετρο τιμής δημιουργείται ένα αντίγραφο της πραγματικής παραμέτρου το οποίο έχει εμβέλεια την συνάρτηση κλήσης (τοπικό αντίγραφο).

Έτσι, η συνάρτηση inc2 για παράδειγμα, προσομοιάζει την συνάρτηση inc.

```
void inc2(int& i) {int j=i; j++;}
```

Η inc2 δημιουργεί σαφώς (explicitly) ένα τοπικό αντίγραφο της πραγματικής παραμέτρου της, επί του οποίου στην συνέχεια επιδρά. Η inc δημιουργεί αυτομάτως (implicitly) ένα αντίγραφο της παραμέτρου της, επί του οποίου στην συνέχεια επιδρά. Ας σημειωθεί ότι στην προκειμένη περίπτωση οι inc και inc2 είναι “ακριβότερες” σε σχέση με την inc1 διότι δημιουργούν επιπλέον (αυτομάτως ή μη) ένα αντίγραφο της παραμέτρου των.

Η inc1 περιγράφεται στην συνέχεια με χρήση ψευδοκώδικα

```
void inc3(addressOfInt atMem) { evaluate(atMem)++;}
```

Προσέξτε ο μηχανισμός παραμέτρων τιμής εγγυάται ότι η συνάρτηση κλήσης δεν θα μεταβάλλει την τιμή της πραγματικής παραμέτρου. Με αυτήν την έννοια, οι παράμετροι τιμής είναι στην ουσία παράμετροι εισόδου στην συνάρτηση. Οι παράμετροι τιμής

παρέχουν ασφάλεια απέναντι σε λογικά σφάλματα τα οποία αν δεν αποφευχθούν καταλήγουν σε σφάλματα χρόνου εκτέλεσης (run time errors). Το κόστος των παραμέτρων τιμής είναι η δημιουργία και διατήρηση του αντιγράφου.

2.2 Επιστροφή με τιμή ή αναφορά

Αντίστοιχα με την παράμετρο τιμής ορίζεται και η **επιστροφή με τιμή (return by value)**. Η gElement επιστρέφει την τιμή ενός int. Αντίστοιχα με την παράμετρο αναφοράς ορίζεται και η επιστροφή με αναφορά (return by reference). Η getElement επιστρέφει την διεύθυνση ενός int. Η διεύθυνση αυτή είναι μία lvalue[4]. Όπως θα συζητήσουμε στην ενότητα 3.1 (τελεστής εκχώρησης), η εκχώρηση απαιτεί στο αριστερό της μέρος μία lvalue. Για τον λόγο αυτόν είναι σωστή η γραμμή

```
getElement(tbl,2)=13;
```

και λάθος η γραμμή

```
gElement(tbl,2)=13;    //compile time error, lvalue required
```

2.3 Η δεσμευμένη λέξη const

Η δεσμευμένη λέξη const μπορεί να χρησιμοποιηθεί για να χαρακτηρίσει μια ή περισσότερες τυπικές παραμέτρους, π.χ. tstConst1, tstConst3.

```
void tstConst(int i) {
  i=5;
  cout << i;
}

void tstConst1(const int i) {
  //i=5; compile time error
  cout << i;
}

void tstConst2 (int& i) {
  i=5;
  cout << i;
}

void tstConst3 (const int& i) {
  //i=5; compile time error
  cout << i;
}
```

Οι παράμετροι που χαρακτηρίζονται με την λέξη const ονομάζονται **παράμετροι μόνον για ανάγνωση (read-only parameters)**. Πράγματι, όπως φαίνεται στις tstConst1 & tstConst3, τυχόν προσπάθεια εγγραφής σε const παράμετρο καταλήγει σε compile time error.

[4] Location value. Επίσης, άλλοι συγγραφείς το ερμηνεύουν ως left-hand value

Η χρήση της λέξης const με παραμέτρους τιμής, π.χ. tstConst1, δεν είναι συνηθισμένη. Πράγματι, καθώς οι παράμετροι τιμής δεν επιδρούν στην πραγματική παράμετρο σπάνια υπάρχει η ανάγκη να χαρακτηρισθούν ως const. Αντίθετα, η χρήση της const είναι πολύ συνηθισμένη με τις παραμέτρους αναφοράς. Δημιουργείται έτσι ένας οικονομικός μηχανισμός read-only παραμέτρων που ονομάζεται **σταθερή παράμετρος αναφοράς (const reference ή read-only reference parameter).** Η σταθερή παράμετρος αναφοράς βρίσκει μεγάλη εφαρμογή στις περιπτώσεις που η παράμετρος είναι σύνθετος τύπος δεδομένων[5] με σημαντικό μέγεθος. Στην περίπτωση αυτή, η σταθερή παράμετρος αναφοράς πλεονεκτεί έναντι της παραμέτρου τιμής διότι αποφεύγει την δημιουργία τοπικού αντιγράφου το οποίο μπορεί να καταλαμβάνει αρκετά bytes ή kbytes και ταυτοχρόνως εγγυάται πως δεν θα μετατραπεί η πραγματική παράμετρος.

Ας σημειωθεί πως τόσο η παράμετρος τιμής όσο και η σταθερή παράμετρος αναφοράς επιτρέπουν την χρήση σταθερών στην θέση πραγματικών παραμέτρων κάτι που προφανώς δεν υποστηρίζει η παράμετρος αναφοράς. Σχετικό παράδειγμα δίνεται ακολούθως:

```
int i=1;
const int j=2;
tstConst(i);
tstConst(j);
tstConst1(i);
tstConst1(j);
tstConst2(i);
//tstConst2(j); compile time error
tstConst3(i);
tstConst3(j);
```

2.4 Ασκήσεις

Μελετήστε τον ακόλουθο κώδικα.

```
1  #include <iostream>
2
3  using namespace std;
4
5  bool positive(double n1, double n2, double& rslt) {
6      rslt=n1-n2;
7      return rslt>0;
8  }
9
10 void assign (double& l, double r) { l=r;}
11 double assign1 (double& l, double r) { l=r; return l;}
12 double& assign2 (double& l, double r) { l=r; return l;}
13
14 int main() {
15     double n1=32, n2=30, rslt;
16     if (positive(n1,n2,rslt)) {
17         if (rslt>10) { cout << rslt << endl;}
18     }
19     assign2(n1,n2)=2;
```

[5] Οι σύνθετοι τύποι δεδομένων συζητούνται στην ενότητα 18.1

```
20      cout << n1 << ' ' << n2 << endl;
21
22      n1=32; n2=30;
23      assign2(assign2(n1,n2),assign2(n2,n1));
24      cout << n1 << ' ' << n2 << endl;
25
26      n1=32; n2=30;
27      assign2(n1,n2)=assign2(n2,n1);
28      cout << n1 << ' ' << n2 << endl;
29      return 0;
30  }
```

Απαντήστε και δικαιολογήστε τις απαντήσεις σας στις παρακάτω ερωτήσεις:

- Θα μπορούσε το rslt στην συνάρτηση positive να είναι παράμετρος τιμής;
- Θα μπορούσε το l στην assign να είναι const
- Ο κώδικας assign1(n1,n2)=2; είναι σωστός ή λάθος;
- Εξηγήστε το Output της main

3 Ενσωματωμένοι Τελεστές (built-in operators)

Περίληψη : Στην ενότητα αυτή παρουσιάζονται οι προ-ορισμένοι (predefined) τελεστές που λειτουργούν με τους ενσωματωμένους τύπους (build in types). Η βαθύτερη γνώση των χαρακτηριστικών αυτών των τελεστών αποτελεί προαπαιτούμενη γνώση για να μπορέσει κανείς στη συνέχεια να προγραμματίσει, δηλ. να υπερφορτώσει τους τελεστές ώστε να λειτουργούν με τύπους οριζόμενους από τον χρήστη (user defined types). Η συζήτηση κατηγοριοποιεί τους τελεστές ως εξής: Αριθμητικοί (arithmetic), π.χ. +,-,*,/,%, Τελεστές σύγκρισης (comparison operators), π.χ. ==, >, <, Λογικοί(logical), δηλ. !, &&, ||, Bitwise operators(πράξεις σε bitmaps), π.χ. ~, &, |, <<, >>, Εκχώρησης (assignment), =, +=, -=, Τελεστές μελών και δεικτών (member and pointer operators), π.χ. *, &, Λοιποί τελεστές, π.χ. ::, ?,.

Λέξεις κλειδιά : *τελεστές (operators), τιμή θέσης (location value), εκχώρηση (assignment), προσαύξηση (increment), μείωση (degrement)*

Η C++ παρέχει μια σειρά από τελεστές, π.χ. +, =, <, κλπ., για χρήση με τους ενσωματωμένους τύπους. Οι τελεστές δεν είναι παρά συναρτήσεις με παραμέτρους και τιμή επιστροφής. Ωστόσο, υποστηρίζονται από ειδικό συντακτικό που συχνά βασίζεται σε μη αλφαβητικούς χαρακτήρες. Για παράδειγμα, ας εξετάσουμε την περίπτωση της πρόσθεσης. Έστω η συνάρτηση int add(const int& a, const int& b) που προσθέτει τα a και b και επιστρέφει το αποτέλεσμα. Αντίστοιχα, το interface του τελεστή πρόσθεσης είναι[6] int **operator**+(const int& a, const int& b), δηλ. προκύπτει αν αντικαταστήσουμε το όνομα της συνάρτησης με την δεσμευμένη-λέξη (**reserved word**) **operator** ακολουθούμενη από το σύμβολο του τελεστή. Στην περίπτωση που χρησιμοποιούμε την λέξη **operator**, τότε η κλήση δεν γίνεται με τον συμβατικό τρόπο, π.χ. operator+(b,c) αλλά εξειδικεύεται ανά τελεστή. Όπως είναι γνωστό στην περίπτωση του τελεστή πρόσθεσης η κλήση είναι b+c.

Οι τελεστές κατηγοριοποιούνται ως ακολούθως;

- Αριθμητικοί (arithmetic), π.χ. +,-,*,/,%, κλπ.
- Τελεστές σύγκρισης (comparison operators), π.χ. ==, >, < κλπ.
- Λογικοί (logical), δηλ. !, &&, ||, κλπ.
- Bitwise operators (πράξεις σε bitmaps), π.χ. ~, &, |, <<, >> κλπ.
- Σύνθετης εκχώρησης (compound assignment), +=, -=, κλπ.
- Τελεστές μελών και δεικτών (member and pointer operators), π.χ. *, &, κτλ.
- και λοιποί τελεστές, π.χ. ::, ?, κτλ.

Δείτε την πλήρη λίστα των τελεστών της C++ ταξινομημένων σύμφωνα με την παραπάνω κατηγοριοποίηση στην Wikipedia.

[6] Πιο συγκεκριμένα, το interface του τελεστή πρόσθεσης είναι T **operator**+(const T& a, const T& b), όπου T οποιοδήποτε data type για το οποίο ορίζεται η πρόσθεση.

Οι τελεστές μπορεί να είναι μοναδιαίοι (unary), δυαδικοί (binary) ή τριαδικοί (ternary), δηλ. να λαμβάνουν μία, δύο ή τρείς παραμέτρους, αντίστοιχα.

3.1 Ο τελεστής εκχώρησης (assignment operator)

Ο τελεστής εκχώρησης είναι δυαδικός. Το συντακτικό κλήσης του είναι a=b. Η κατά σειρά πρώτη παράμετρός του οφείλει να είναι **lvalue (location value)**. Η τιμή της δεύτερης παραμέτρου αντιγράφεται στην πρώτη. Η τιμή επιστροφής της εκχώρησης είναι η διεύθυνση της πρώτης παραμέτρου.

Πιο συγκεκριμένα, το interface της εκχώρησης δίνεται ακολούθως

`T& operator=(T& a, const`[7] `K& b),` όπου Τ, Κ ενσωματωμένοι τύποι

Προσοχή, στην C++, όπως και στην C δεν γίνεται έλεγχος ώστε οι δύο παράμετροι της εκχώρησης να είναι ίδιου τύπου.

Η πολλαπλή εκχώρηση είναι δυνατή, π.χ.

```
int a,b,c;
a=b=c=0;
```

Στην περίπτωση της εκχώρησης ο υπολογισμός (evaluation) της έκφρασης γίνεται από τα δεξιά προς τα αριστερά. Πιο συγκεκριμένα, η παραπάνω έκφραση είναι ισοδύναμη με την έκφραση a=(b=(c=0));

3.2 Τελεστές προσαύξησης και μείωσης (Increment, decrement operators), ++, --

Ο τελεστής προσαύξησης είναι **μοναδιαίος**. Υπάρχουν δύο τελεστές προσαύξησης, ο **προθεματικός** (prefix) με συντακτικό κλήσης ++a και ο **επιθεματικός** (suffix) με συντακτικό κλήσης a++.

Ο επιθεματικός τελεστής αυξάνει την τιμή της παραμέτρου του και επιστρέφει την τιμή που είχε η παράμετρος πριν την αύξηση. Αντίθετα, ο προθεματικός αυξάνει την τιμή της παραμέτρου του και στην συνέχεια την επιστρέφει.

Το interface του προθεματικού τελεστή είναι `T& operator++(T& a)`

Το interface του επιθεματικού τελεστή είναι `T operator++(T& a, int)`

Προσέξτε πως ο επιθεματικός τελεστής έχει δύο παραμέτρους. Η δεύτερη είναι ανώνυμη. Πράγματι η C++, μας δίνει την δυνατότητα να ορίσουμε μια η περισσότερες ανώνυμες παραμέτρους. Κάτι τέτοιο βέβαια έχει νόημα μόνον εφόσον δεν αναφερόμαστε στην ανώνυμη παράμετρο, διαφορετικά είναι προφανές πως το όνομα είναι απαραίτητο. Επίσης, τις παραμέτρους μπορούμε να μην τις **ονομάζουμε** στις δηλώσεις των συναρτήσεων, π.χ. δείτε την δήλωση της gElement. Όσον αφορά την ανώνυμη παράμετρο του επιθεματικού

[7] Ο μεταγλωττιστής αποτρέπει την μεταβολή της τιμής μιας παραμέτρου που έχει χαρακτηριστεί με το keyword `const`. Περισσότερες λεπτομέρειες για το θέμα αυτό έχουν δοθεί στην ενότητα 1.2.3.

τελεστή προσαύξησης οφείλεται σε σύμβαση της C++ με σκοπό την διαφοροποίηση μεταξύ προθεματικού και επιθεματικού τελεστή.

Ποια είναι η έξοδος του κώδικα που ακολουθεί;

```
int i=0;
 std:: cout << i++;
 std::cout << " " << ++i << std::endl;
```

Αντίστοιχα με τον τελεστή προσαύξησης και ο τελεστής μείωσης (--) διακρίνεται σε προθεματικό και επιθεματικό.

3.3 Τελεστές διολίσθησης σε ρεύμα, <<, >>

Για τους τελεστές **διολίσθησης** θα μιλήσουμε αναλυτικά στην ενότητα 20 (αναπαράσταση ακεραίων). Εδώ θα αναφερθούμε στην διολίσθηση σε ρεύμα. Στην περίπτωση αυτή, η αριστερή διολίσθηση (left shift), <<, ονομάζεται και εισαγωγή σε ρεύμα (stream insertion) και η δεξιά διολίσθηση (right shift), >>, ονομάζεται και εξαγωγή από ρεύμα (stream extraction). Η εισαγωγή σε ρεύμα συντάσσεται με ρεύμα εξόδου, π.χ. ofstream και χρησιμοποιείται για εγγραφή σε αρχεία και η εξαγωγή από ρεύμα συντάσσεται με ρεύμα εισόδου, π.χ. ifstream και χρησιμοποιείται για ανάγνωση από αρχείο. Ακολούθως δίνεται παραδειγματικός κώδικας.

```
#include <fstream>
  std::ofstream tOut( "fName.txt" );
  int x=5;
  tOut << "Hello! I'm " << x << " years old";
  tOut.close();
  char in[25];
  std::ifstream tIn("fName.txt");
  while (!tIn.eof()) {
    tIn >> in;
    std::cout << in << " ";
  }
  std::cout << std::endl;
```

Ας σημειωθεί ότι το ρεύμα cout αφορά την **τυπική έξοδο (standard output)**, δηλ. την οθόνη. Αντίστοιχα, το ρεύμα cin μπορεί να χρησιμοποιηθεί για την τυπική είσοδο (standard input), δηλ. το πληκτρολόγιο.

Οι εν λόγω τελεστές είναι δυαδικοί. Η πρώτη παράμετρος είναι η διεύθυνση ενός ρεύματος και η δεύτερη η διεύθυνση της μεταβλητής που θέλουμε να διολισθήσει στο ρεύμα. Ας σημειωθεί ότι ένα ρεύμα είναι μια σύνθετη μεταβλητή[8] που περιλαμβάνει μια περιοχή μνήμης, δηλ. έναν **buffer**. Στον buffer τοποθετούνται οι τιμές που διολισθαίνουν από όπου στην συνέχεια εγγράφονται στο αρχείο. Τιμή επιστροφής είναι η τιμή του ρεύματος όπως

[8] Στην ουσία πρόκειται για μια κλάση. Περισσότερα επάνω σε αυτό στον Αντικειμενοστραφή Προγραμματισμό.

αυτό διαμορφώνεται μετά την διολίσθηση. Πιο συγκεκριμένα, το συντακτικό των τελεστών διολίσθησης μπορεί να περιγραφεί σε ψευδοκώδικα ως ακολούθως :

```
oS& operator <<(const oS& a, const T& b);

iS& operator >>(const iS& a, const T& b);
```

Όπου oS είναι ένα output Stream, iS είναι input Stream και T οποιοσδήποτε τύπος. Ας σημειωθεί ότι στην πραγματικότητα το συντακτικό των τελεστών διολίσθησης είναι ελαφρώς διαφορετικό καθώς η διολίσθηση υποστηρίζεται προς όλους τους τύπους δεδομένων και όχι μόνο προς τα ρεύματα.

Τέλος, ο υπολογισμός παραστάσεων με πολλαπλούς τελεστές διολίσθησης γίνεται από αριστερά προς τα δεξιά, δηλ. η έκφραση

```
tOut << "Hello! I'm " << x << " years old"
```

είναι ισοδύναμη με την έκφραση

```
((tOut << "Hello! I'm ") << x) << " years old"
```

3.4 Λοιποί τελεστές

Ενημερωθείτε για τον τελεστή παράσταση υπό συνθήκη (conditional expression), τους τελεστές σύνθετης εκχώρησης και άλλους τελεστές.

3.5 Ασκήσεις

1. Να αναπτυχθεί συνάρτηση **assign** με ακέραιες παραμέτρους που προσομοιάζει την λειτουργία του τελεστή εκχώρησης. Σε κατάλληλο τεστ κώδικα να δηλωθεί int a=0, b=0, c=1; Στην συνέχεια να εκτελεσθεί η πολλαπλή εκχώρηση a=b=c μέσω της **assign**. Να ελεγχθούν οι τιμές των a,b,c. Να δοθεί εναλλακτική λίστα παραμέτρων για την **assign**.
2. Να αναπτυχθούν οι συναρτήσεις **prefixInc** και **suffixInc** που προσομοιάζουν την λειτουργία του προθεματικού και επιθεματικού τελεστή προσαύξησης, αντίστοιχα. Για τις συναρτήσεις αυτές να χρησιμοποιηθούν ο προθεματικός και ο επιθεματικός τελεστής προσαύξησης, αντίστοιχα. Να αναπτυχθούν οι συναρτήσεις **prefixInc2** και **suffixInc2** με τις προφανείς προδιαγραφές. Για την ανάπτυξη των συναρτήσεων αυτών να μην χρησιμοποιηθούν ο προθεματικός και ο επιθεματικός τελεστής προσαύξησης. Εξηγήστε γιατί ο επιθεματικός τελεστής προσαύξησης επιστρέφει την τιμή του **by value**. Είναι δυνατόν να χρησιμοποιηθεί για τον συγκεκριμένο τελεστή **return by reference**;
3. Να αναπτυχθεί συνάρτηση **swap** που λαμβάνει δύο ακέραιες παραμέτρους και αντιμεταθέτει τα περιεχόμενά τους. Να αιτιολογηθεί η επιλογή των τυπικών παραμέτρων.
4. Να αναπτυχθούν οι προφανείς συναρτήσεις

```
int max(const int& a, const int& b);
```

```
const int& max1(const int& a, const int& b);
```

Για την ανάπτυξη της max να χρησιμοποιηθεί ο τελεστής παράσταση υπό συνθήκη. Για την max1 να χρησιμοποιηθεί η if.

Η συνάρτηση

```
int& max1(const int& a, const int& b);
```

περνάει από μεταγλώττιση; Εξηγήστε την απάντησή σας.

4 Ελάχιστη αξιολόγηση

Η **ελάχιστη αξιολόγηση (short circuit evaluation, minimal evaluation ή McCarthy evaluation)** στην C++ αφορά τους λογικούς τελεστές && (and) και || (or). Ελάχιστη αξιολόγηση σημαίνει ότι διακόπτεται η αξιολόγηση των λογικών εκφράσεων (**expressions**) αμέσως με τον υπολογισμό της τιμής τους. Ας σημειωθεί ότι η αξιολόγηση των λογικών εκφράσεων γίνεται από αριστερά προς τα δεξιά. Ας υποθέσουμε ότι e1 και e2 είναι δύο λογικές εκφράσεις, τότε αν e1 είναι false συνεπάγεται πως e1 && e2 είναι επίσης false, ανεξαρτήτως της τιμής της e2. Στην περίπτωση αυτή η e2 δεν αξιολογείται. Αντίθετα, αν e1 είναι true τότε η e2 αξιολογείται υποχρεωτικά. Αντίστοιχα, αν e1 είναι true συνεπάγεται πως e1 || e2 είναι επίσης true, ανεξαρτήτως της τιμής της e2. Στην περίπτωση αυτή η e2 δεν αξιολογείται. Αντίθετα, αν e1 είναι false τότε η e2 αξιολογείται υποχρεωτικά. Εκτελέστε και μελετήστε τον παρακάτω κώδικα.

```
bool negative(int i) {
      cout << "check short circuit evaluation: negative " << i
      << endl ;
      return i<0;
}

bool positive (int i) {
      cout << "check short circuit evaluation: positive " << i
      << endl ;
      return i>0;
}

int main () {
      int i=-11;
      if (negative(i) || (!positive (i))) cout << " i is
      negative" ;
}
```

Ας σημειωθεί ότι οι επενέργειες μιας συνάρτησης εκτός της τιμής επιστροφής της είναι γνωστές ως παράπλευρα αποτελέσματα (side effects), π.χ. οι cout στις συναρτήσεις negative και positive. Προσέξτε, ότι η ελάχιστη αξιολόγηση θέτει περιορισμούς στην χρήση συναρτήσεων με παράπλευρα αποτελέσματα καθώς δεν είναι βέβαιο ότι αυτές θα εκτελεστούν ως μέρος μιας boolean έκφρασης.

5 Οργάνωση προγράμματος

Περίληψη : Η συζήτηση σε αυτήν την ενότητα έχει ως σκοπό να δώσει στους σπουδαστές την δυνατότητα να οργανώνουν τα προγράμματα τους έτσι ώστε να είναι δυνατή η διάθεσή τους χωρίς να απαιτείται η διάθεση του πηγαίου κώδικα. Περιλαμβάνονται οι έννοιες : Μεταγλωττιστής (Compiler), Διασυνδέτης (Linker), Πηγαίος κώδικας (source code), ενδιάμεσος κώδικας (object code), εκτελεστέος κώδικας (executable code), Ονοματοχώροι (namespaces), η οδηγία (directive) include, αρχεία .h και αρχεία .cpp.

Λέξεις κλειδιά : *Μεταγλωττιστής (Compiler), Διασυνδέτης (Linker), Πηγαίος κώδικας (source code), ενδιάμεσος κώδικας (object code), εκτελεστέος κώδικας (executable code), Ονοματοχώροι (namespaces), η οδηγία (directive) include, αρχεία .h και αρχεία .cpp.*

5.1 Γλώσσα Μηχανής (Machine Code)

Η θεμελιώδης γλώσσα προγραμματισμού είναι η λεγόμενη Γλώσσα Μηχανής. Το "αλφάβητο" αυτής της γλώσσας είναι οι χαρακτήρες 0 και 1, δηλ. τα δυαδικά ψηφία (binary digits ή bits). Επομένως, τα προγράμματα που είναι γραμμένα σε γλώσσα μηχανής δεν είναι παρά ακολουθίες από bits. Η γλώσσα μηχανής είναι η μοναδική γλώσσα στην οποία ανταποκρίνεται η μηχανή, δηλ. το υλικό μέρος του υπολογιστή. Στην εικόνα 1 δίνεται ένα παράδειγμα γλώσσας μηχανής.

Assembly Language	Machine Code
add $t1, t2, $t3	04CB: 0000 0100 1100 1011
addi $t2, $t3, 60	16BC: 0001 0110 1011 1100
and $t3, $t1, $t2	0299: 0000 0010 1001 1001
andi $t3, $t1, 5	22C5: 0010 0010 1100 0101
beq $t1, $t2, 4	3444: 0011 0100 0100 0100
bne $t1, $t2, 4	4444: 0100 0100 0100 0100
j 0x50	F032: 1111 0000 0011 0010
lw $t1, 16($s1)	5A50: 0101 1010 0101 0000
nop	0005: 0000 0000 0000 0101
nor $t3, $t1, $t2	029E: 0000 0010 1001 1110
or $t3, $t1, $t2	029A: 0000 0010 1001 1010
ori $t3, $t1, 10	62CA: 0110 0010 1100 1010
ssl $t2, $t1, 2	0455: 0000 0100 0101 0101
srl $t2, $t1, 1	0457: 0000 0100 0101 0111
sw $t1, 16($t0)	7050: 0111 0000 0101 0000
sub $t2, $t1, $t0	0214: 0000 0010 0001 0100

Εικόνα 1: Γλώσσα Μηχανής

5.2 Κωδικοποιημένη Γλώσσα Μηχανής(Assembly)

Η Assembly είναι η γλώσσα μηχανής κωδικοποιημένη. Πρόκειται επομένως για μια συμβολική χαμηλού επιπέδου γλώσσα προγραμματισμού, δηλαδή μια γλώσσα πολύ κοντά στη γλώσσα μηχανής και στο υλικό του υπολογιστή. Κάθε συγκεκριμένη αρχιτεκτονική συνόλου εντολών, δηλαδή κάθε οικογένεια επεξεργαστών, έχει τη δική της συμβολική γλώσσα, η οποία δίνεται συνήθως από τον κατασκευαστή της. Η Assembly είναι πιο ευανάγνωστη από την γλώσσα μηχανής και διευκολύνει τον άνθρωπο να κατανοεί την γλώσσα μηχανής.

Για παράδειγμα ένας επεξεργαστής της αρχιτεκτονικής x86/IA-32 θα καταλάβει την εντολή σε γλώσσα μηχανής:

```
10110000 01100001
```

Ένας προγραμματιστής όμως είναι πιο εύκολο να θυμάται την ισοδύναμη συμβολική αναπαράσταση, για παράδειγμα μια τυπική εντολή σε συμβολική γλώσσα είναι η εξής:

```
mov  al, 061h
```

που είναι συντομογραφία της αγγλικής λέξης move ("μετακίνησε"). Η εντολή αυτή μετακινεί τη δεκαεξαδική τιμή 61 (97 στο δεκαδικό σύστημα) στον καταχωρητή με το όνομα "al".

Η μετατροπή ενός προγράμματος από συμβολική γλώσσα σε γλώσσα μηχανής γίνεται από ένα συμβολομεταφραστή (assembler) και το αντίστροφο γίνεται από έναν αντισυμβολομεταφραστή (disassembler).

5.3 Γλώσσες υψηλού επιπέδου (High level languages)

Ως υψηλού επιπέδου γλώσσα προγραμματισμού (high-level programming language) ορίζεται αυτή που επιτρέπει τη μεταφερσιμότητα ενός προγράμματος από έναν υπολογιστή σε έναν άλλο. Αποτελείται από εντολές εύκολα κατανοητές στον προγραμματιστή, καθώς μοιάζουν με -περιορισμένη- φυσική γλώσσα. Για την εκτέλεση του προγράμματος από τον υπολογιστή, απαιτείται η χρήση μεταγλωττιστή για την παραγωγή του προγράμματος σε γλώσσα μηχανής. Παραδείγματα γλωσσών υψηλού επιπέδου : C++, Cobol, Pascal, κα.

5.4 Πηγαίος κώδικας (Source code)

Πηγαίος κώδικας είναι οποιαδήποτε σειρά από εντολές ή δηλώσεις σε κάποια γλώσσα υψηλού επιπέδου. Ο όρος πηγαίος κώδικας αναφέρεται συνήθως σε εντολές που

γράφονται από κάποιον προγραμματιστή σε μια γλώσσα προγραμματισμού, και όχι σε εντολές που παράγονται αυτόματα από λογισμικό.

Ο πηγαίος κώδικας μπορεί να μεταγλωττιστεί σε εκτελέσιμο κώδικα μηχανής ή να εκτελεστεί ως έχει από κάποιον διερμηνευτή.

5.5 Διερμηνευτής (Interpreter)

Ο Διερμηνευτής δέχεται ως είσοδο έναν κώδικα υψηλού επιπέδου και μεταφράζει και εκτελεί μια-μια τις εντολές του κώδικα.

5.6 Μεταγλωττιστής (compiler)

Ο Μεταγλωττιστής δέχεται ως είσοδο έναν κώδικα υψηλού επιπέδου, τον μεταφράζει παράγοντας έναν κώδικα που ονομάζεται Object code (Αντικειμενικός).

5.7 Linker (Διασυνδέτης)

Ο διασυνδέτης διασυνδέει ένα σύνολο από αντικειμενικούς κώδικες και παράγει τον εκτελέσιμο (executable) κώδικα, δηλ. τον κώδικα μηχανής

5.8 Προεπεξεργαστής (Pre-Processor)

Ο προεπεξεργαστής κάνει κάποια επεξεργασία στον πηγαίο κώδικα, όπως για παράδειγμα αντικαθιστά το όνομα ενός αρχείου με το περιεχόμενό του, π.χ. οδηγία include. Όπως φανερώνει το όνομά του, η επεξεργασία αυτή εφαρμόζεται πριν την μεταγλώττιση.

5.9 Απολαθοποιητής (Debugger)

Ο debugger διευκολύνει στην εύρεση και διόρθωση λαθών του πηγαίου κώδικα. Ας σημειωθεί πως υπάρχουν 2 τύποι λαθών: Τα λάθη μεταγλώττισης (compile time errors) που εντοπίζει ο μεταγλωττιστής και επομένως είναι αντιμετωπίσιμα και τα λάθη χρόνου εκτέλεσης (run time errors) που δεν εντοπίζει ο μεταγλωττιστής και άρα εκδηλώνονται κατά τον χρόνο εκτέλεσης. Τα λάθη του χρόνου εκτέλεσης είναι πάρα πολύ σοβαρά καθώς περνάνε στον εκτελέσιμο κώδικα, φτάνουν στον χρήστη και είναι δυνατόν να προκαλέσουν σημαντικές καταστροφές, π.χ. λάθη χρόνου εκτέλεσης σε λογισμικό πλοήγησης αεροσκάφους.

5.10 Ονοματοχώρος (Namespace)

Η **ταυτότητα (signature)** μιας συνάρτησης συνθέτεται από το όνομά της και την λίστα των **πραγματικών παραμέτρων (actual parameters)** της. Τα ονόματα των πραγματικών παραμέτρων δεν συμμετέχουν στην ταυτότητα· αντίθετα, συμμετέχουν οι τύποι των πραγματικών παραμέτρων και η σειρά με την οποία αυτές δηλώνονται.

Για παράδειγμα:

```
int f(int x, bool y);
void f(int z , bool y);
void f(bool y, int z );
```

Η πρώτη συνάρτηση έχει ίδια ταυτότητα με την δεύτερη, ενώ η Τρίτη έχει διαφορετική ταυτότητα τόσο από την δεύτερη όσο και από την πρώτη.

Ο **ονοματοχώρος** είναι ένας μηχανισμός που μας επιτρέπει να διαφοροποιούμε μεταξύ συναρτήσεων με κοινή **ταυτότητα (signature)**.

Όπως είναι γνωστό, η ανάπτυξη προγραμμάτων συχνά περιλαμβάνει την χρήση **βιβλιοθηκών (libraries)**, π.χ. η βιβλιοθήκη iostream. Δύο ή περισσότερες, βιβλιοθήκες, πιθανώς προερχόμενες από διαφορετικούς κατασκευαστές, ενδέχεται να δηλώνουν συναρτήσεις με κοινή ταυτότητα. Στην περίπτωση αυτή, η ταυτόχρονη χρήση τους οδηγεί σε **σύγκρουση (conflict)** που εκδηλώνεται κατά την μεταγλώττιση. Η χρήση του μηχανισμού των ονοματοχώρων μας επιτρέπει να χωρίσουμε τον **καθολικό χώρο (global space)** σε υποχώρους, ο κάθε ένας εκ των οποίων διαθέτει το όνομά του. Έτσι, η ταυτότητα των συναρτήσεων **εξειδικεύεται (qualified)** από το όνομα του ονοματοχώρου στον οποίο δηλώνονται.

Ας σημειωθεί ότι ο μηχανισμός ονοματοχώρων χρησιμοποιείται και για την διαφοροποίηση μεταβλητών που ορίζονται με το ίδιο **αναγνωριστικό (identifier)**.

Στην συνέχεια δίνεται παράδειγμα χρήσης των ονοματοχώρων

```
//file nS1.h
#ifndef NS1_H_INCLUDED
#define NS1_H_INCLUDED
namespace nS1 {
  void prt ();
}
#endif // NS1_H_INCLUDED

//file nS2.h
#ifndef NS2_H_INCLUDED
#define NS2_H_INCLUDED
namespace nS2 {
  void prt ();
}
#endif // NS2_H_INCLUDED

//file nS1.cpp
#include <iostream>
#include "nS1.h"
void nS1::prt () {
  std::cout << "NameSpace nS1" << std ::endl ;
}

//file nS2.cpp
#include <iostream>
#include "nS2.h"
void nS2::prt () {
  std::cout << "nameSpace nS2" << std ::endl ;
}

//file main.cpp
#include <iostream>
#include "nS1.h"
#include "nS2.h"
using namespace std ;
```

```
int main () {
  nS1 ::prt ();
  nS2 ::prt ();
}
```

Προσέξτε στην main πως οι συναρτήσεις prt εξειδικεύονται κατάλληλα με το όνομα του αντίστοιχου ονοματοχώρου. Το ίδιο συμβαίνει με το ρεύμα cout και την endl στα nS1.cpp και nS2.cpp. Σημειώστε πως ο τελεστής :: ονομάζεται **τελεστής επίλυσης εμβέλειας (scope resolution operator)**.

Ακολούθως δίνεται ένα ακόμη παράδειγμα χρήσης ονοματοχώρου.

```
int cout=9;
#include <iostream>
int main () {
  std ::cout << cout << std::endl;
}
```

Στις περιπτώσεις που χρειαζόμαστε συνεχή πρόσβαση στα μέλη ενός ονοματοχώρου, μπορούμε αντί να επαναλαμβάνουμε το τροποποιητικό του ονοματοχώρου, να χρησιμοποιήσουμε την δεσμευμένη λέξη **using**, π.χ.

```
int i=9;
#include <iostream>
using namespace std;
int main () {
  cout << i << endl;
}
```

6 Δομές ελέγχου (Control structures)

Διαβάστε τις επαναληπτικές δομές for, while, do-while από το Πράξεις και Εντολές (Αλεβίζος). Επίσης, διαβάστε το πολύ καλό control structures. Επιπλέον των επαναληπτικών δομών μελετήστε τις **break**, **continue**, **exit** & **switch**. Σημειώστε, πως η χρήση της **goto** δεν συνίσταται.
Μελετήστε το ακόλουθο demo σχετικά με τις δομές ελέγχου.

```
#include <iostream>
#include <cstdlib>
using namespace std;

void prtEven() {
      cout << "prtEven ";
      for (int i=0; i<=20; i+=2) cout << i << " ";
      cout << endl ;
}

void prtAlphabetFor() {
      cout << "prtAlphabetFor ";
      for (char c='a'; c<='z'; c++) cout << c << " ";
```

```
	cout << endl ;
}

void prtAlphabetWhile() {
	cout << "prtAlphabetWhile ";
	char c='a';
	while (c<='z') cout << c++ << " ";
	cout << endl ;
}

void prtAlphabetDo() {
	cout << "prtAlphabetDo ";
	char c='a';
	do
		cout << c++ << " ";
	while (c<='z');
	cout << endl ;
}

void prtAlphabetBreak() {
	cout << "prtAlphabetBreak ";
	char c='a';
	while (true) {
		cout << c++ << " ";
		if (c>'z') break ;
	}
	cout << endl ;
}

void continueX () {
	for (int n=10; n>0; n--) {
		if (n==5) continue;
		cout << n << ", ";
	}
	cout << "FIRE!\n";
}

void mChoiceIf (int i) {
	if (i<0) cout << "Negative" << endl ;
	else if (i<10) cout << "One digit value";
	else if (i<100 ) cout << "Two digits value" ;
	else if (i<1000 ) cout << "Three digits value" ;
	else cout << "Over three digits value" ;
	cout << endl ;
}

void mChoiceSwitch (int i) {
	if (i<=0) {
		cout << "unrecovered error value in switch test" <<
	endl ;
		exit (1);
	}
	switch (2*i) {
		case 2 :
			cout << "first even" << endl ;
			break ;
```

```
            case 4 :
                  cout << "second even" << endl ;
                  break ;
            default :
            cout << "high order" << endl ;
      }
}

int main () {
//prtEven();
//prtAlphabetFor();
//prtAlphabetWhile();
//prtAlphabetDo();
//prtAlphabetBreak();
//mChoiceIf(35);
mChoiceSwitch(1);
continueX();
}
```

6.1 Ασκήσεις

1. Να ορίσετε δύο ξεχωριστούς ονοματοχώρους (namespaces) με ονόματα first και second, να ορίσετε και να αρχικοποιήσετε τις μεταβλητές var (ακέραιος) και var (δεκαδικός) στους αντίστοιχους ονοματοχώρους. Στη συνέχεια να γίνει κλήση των αντίστοιχων μεταβλητών και να εμφανιστούν στην οθόνη.

2. Να ορίσετε δύο ξεχωριστούς ονοματοχώρους (namespace) με ονόματα first και second, να ορίσετε και να αρχικοποιήσετε τις μεταβλητές x,y (ακέραιος) και x,y (δεκαδικός) στους αντίστοιχους ονοματοχώρους. Στην συνέχεια να γίνει κλήση και εμφάνιση και των τεσσάρων μεταβλητών με την βοήθεια της δεσμευμένης λέξης using.

3. Να γραφτεί πρόγραμμα το οποίο θα ζητάει από τον χρήστη να εισάγει 10 ακέραιους αριθμούς που αντιστοιχούν στου βαθμούς των αντίστοιχων μαθημάτων ενός φοιτητή. Οι βαθμοί είναι ακέραιοι αριθμοί από 0 ως 100. Με την ολοκλήρωση της εισαγωγής των βαθμών από το χρήστη το πρόγραμμα θα υπολογίζει τον μέσο όρο και θα τον εμφανίζει στην οθόνη συνοδευόμενο από το αντίστοιχο μήνυμα.

4. Να γραφτεί πρόγραμμα το οποίο θα ζητάει από τον χρήστη να εισάγει τις επιδόσεις των φοιτητών (έναν από τους χαρακτήρες A,B,C,D,E,F). Η εισαγωγή θα τερματίζει δίνοντας ως είσοδο το κεφαλαίο γράμμα Q (χρησιμοποιήστε την συνάρτηση cin.get() για να μετατρέπετε τους χαρακτήρες σε ακέραιους. Ο χαρακτήρας Q αντιστοιχεί στο 81 σε ASCII). Στο τέλος, το πρόγραμμα θα υπολογίζει και θα εμφανίζει το πλήθος των ξεχωριστών χαρακτήρων A,B,C,D,E,F που έχουν εισαχθεί.

Να προβλεφτεί η περίπτωση όπου ο χρήστης δίνει μικρά γράμματα αντί κεφαλαίων. Επίσης να προβλεφτεί και η περίπτωση της εισαγωγής από τον χρήστη ενός χαρακτήρα ο οποίος δεν είναι ένας από τους ζητούμενους(A,B,C,D,E,F).

7 Πολυδιάστατοι πίνακες

Περίληψη : Η ενότητα αυτή παρουσιάζει τους πολυδιάστατους πίνακες, δίνει τα ιδιαίτερα χαρακτηριστικά τους και τους περιορισμούς των μη δυναμικών πολυδιάστατων πινάκων της C++ και προετοιμάζει τους σπουδαστές ώστε να κατανοήσουν πώς να υπερβούν τους υφιστάμενους περιορισμούς χρησιμοποιώντας δυναμικούς πίνακες

Οι πολυδιάστατοι πίνακες μπορούν να θεωρηθούν ως πίνακες πινάκων, π.χ. ο παρακάτω κώδικας

```
int tbl[2][3]={ {11,12,13},
                {21,22,23}
              };
```

δημιουργεί ένα πίνακα ακεραίων(διαστάσεων) 2x3 ή αλλιώς έναν πίνακα 2 γραμμών και 3 στηλών[9].

Τα στοιχεία ενός πολυδιάστατου πίνακα προσπελαύνονται με κατάλληλη δεικτοδότηση του αναγνωριστικού του πίνακα, π.χ.

```
for (int i=0; i<2; i++) {
  for (int j=0; j<3; j++) {
     cout.width (3);
     cout << tbl [i][j];
  }
  cout << endl ;
}
```

Όπως είναι προφανές από τον παραπάνω κώδικα ο δείκτης κάθε διάστασης ενός πίνακα εκτείνεται από το 0 έως το μέγεθος της διάστασης -1.

Οι πολυδιάστατοι πίνακες παρέχουν μια διεπαφή (interface) στον προγραμματιστή και διευκολύνουν την λογική οργάνωση μιας σειράς ομοειδών πληροφοριών, π.χ. ο ακόλουθος πίνακας δηλώνει έναν χαρακτήρα για κάθε ώρα ενός αιώνα.

```
char century [100][365][24];
```

Ωστόσο, οι ίδιες πληροφορίες μπορεί να αποθηκευθούν σε ένα μονοδιάστατο πίνακα.

```
char century [100*365*24];
```

Έτσι, ότι μπορούμε να επιτύχουμε χρησιμοποιώντας έναν πολυδιάστατο πίνακα μπορούμε επίσης να το επιτύχουμε με την χρήση ενός μονοδιάστατου πίνακα.

Παράδειγμα: Στην συνάρτηση `twoDimTbl` χρησιμοποιούμε ένα πίνακα διαστάσεων 3x4.

```
void twoDimTbl() {
  const int noLns=3, noClmns =4;
```

[9] Συχνά, όταν αναφερόμαστε σε δυσδιάστατους πίνακες αποκαλούμαι την μια διάσταση γραμμές και την άλλη στήλες.

```
  int tbl [noLns][noClmns];
  for (int i=0; i<noLns; i++)
    for (int j=0; j<noClmns; j++) tbl[i][j]=i*noClmns +j;

  for (int i=0; i<noLns; i++) {
    for (int j=0; j<noClmns ; j++) {
     cout .width (3);
     cout << tbl [i][j];
    }
    cout << endl;
  }
}
```

Προσέξτε ότι στο στοιχείο `[i][j]` τοποθετούμαι την τιμή `i*noClmns +j`. Σαν αποτέλεσμα κάθε στοιχείο του δυσδιάστατου πίνακα λαμβάνει μια διαδοχική τιμή. Έτσι, ο τύπος αυτός μπορεί να χρησιμοποιηθεί για την μετατροπή των τιμών των δεικτών ενός δυσδιάστατου πίνακα στην τιμή του δείκτη ενός αντίστοιχου μονοδιάστατου. Αυτή η μετατροπή χρησιμοποιείται στην συνάρτηση `twoDimTbl2` στην οποία επιτυγχάνεται ίδιο αποτέλεσμα με την `twoDimTbl`, μόνο που εδώ χρησιμοποιείται μονοδιάστατος πίνακας.

```
void twoDimTbl2() {
    const int noLns=3, noClmns=4;
    int tbl[noLns*noClmns];
    for (int i=0; i<noLns; i++)
       for        (int        j=0;        j<noClmns;        j++)
         tbl[i*noClmns+j]=i*noClmns+j;

    for (int i=0; i<noLns; i++) {
      for (int j=0; j<noClmns; j++) {
           cout.width(3);
           cout << tbl[i*noClmns+j];
      }
      cout << endl;
    }
}
```

Στην πράξη, η εσωτερική οργάνωση των πολυδιάστατων πινάκων βασίζεται στους μονοδιάστατους. Για παράδειγμα, ένας δυσδιάστατος πίνακας 2X3 αποτελείται από 6 στοιχεία τα οποία είναι κατανεμημένα σειριακά, όπως επιδεικνύεται από το ακόλουθο παράδειγμα.

```
void twoDimTbl3() {
  int tbl [][3]={{1,2,3}, {4,5,6}};
  int *p=&tbl [0][0];
  for (int i=0; i<2*3; i++) cout << p[i] << " " ;
  cout << endl ;
}
```

Οι πολυδιάστατοι πίνακες μπορούν να περνάν ως παράμετροι σε συναρτήσεις. Στην λίστα τυπικών παραμέτρων, ένας πίνακας μπορεί να έχει απροσδιόριστη μόνον μια διάσταση (την πρώτη), οι υπόλοιπες απαιτείται να είναι σταθερές. Σχετικό παράδειγμα δίνεται με την prt.

```
void prt(int array[][4], int length) {
```

```
  for (int i=0; i<length; i++) {
    for (int j=0; j<4; j++) {
     cout.width(8);
     cout << array[i][j];
    }
    cout << endl;
  }
}
```

Η συνάρτηση prt λαμβάνει ως παράμετρο έναν δυσδιάστατο πίνακα και έναν ακέραιο. Προσέξτε ότι η πρώτη διάσταση του πίνακα (ο αριθμός γραμμών) δεν προσδιορίζεται για τον λόγο αυτό η prt "ενημερώνεται" για το μέγεθος της πρώτης διάστασης διαμέσου της δεύτερης παραμέτρου της, δηλ. της length. Αντίθετα, η δεύτερη διάσταση (ο αριθμός των στηλών) είναι σταθερή με τιμή 4. Έτσι, η prt μπορεί να κληθεί μόνο για δυσδιάστατους πίνακες που η δεύτερη διάστασή τους είναι 4. Το γεγονός αυτό αποτελεί σοβαρό περιορισμό των πολυδιάστατων πινάκων στην C++. Στην συνέχεια, αφού πρώτα παρουσιάσουμε τους δείκτες (pointers), θα δούμε πως με την βοήθεια τους μπορούμε να ξεπεράσουμε τον προαναφερθέντα περιορισμό των πολυδιάστατων πινάκων της C++.

8 Δείκτες[10] (pointers)

Περίληψη : Στην ενότητα αυτή γίνεται διεξοδική συζήτηση για τους δείκτες. Πιο συγκεκριμένα περιλαμβάνονται : γενικά περί δεικτών , αναφορές (references), αριθμητική δεικτών (pointer arithmetic), δείκτες και πίνακες, δείκτες και συναρτήσεις, συνάρτηση παράμετρος, δυναμική παραχώρηση μνήμης (dynamic memory allocation).

Λέξεις κλειδιά : δείκτες (pointers), αναφορές (references), πολυδιάστατοι πίνακες (multidimensional arrays)

Συχνά στα προγράμματά μας η άμεση προσπέλαση, για διάβασμα ή/και γράψιμο, θέσεων μνήμης αποτελεί χρήσιμη δυνατότητα. Για την υλοποίηση της δυνατότητας αυτής απαιτείται ένας τύπος δεδομένων τέτοιος ώστε οι τιμές που εκχωρούνται στις μεταβλητές του να αναπαριστούν διευθύνσεις μνήμης. Ο τύπος αυτός ονομάζεται δείκτης (pointer). Προσέξτε πως ο pointer είναι ακέραιος τύπος. Οι pointers μπορεί να εξειδικευτούν ανάλογα με τα περιεχόμενα της μνήμης στην οποία "δείχνουν". Έτσι, έχουμε pointers ακεραίων, pointers χαρακτήρων, κλπ. Ακολουθούν παραδείγματα δήλωσης pointers.

```
int *pi, *pj;
char *pc, *ps;
```

[10] Άλλοι συγγραφείς μεταφράζουν το όρο pointer ως βέλος. Εμείς αποδίδουμε τον όρο pointer ως δείκτη. Επίσης, αποδίδουμε ως δείκτη τον όρο index, ο οποίος χρησιμοποιείται για μια ακέραια μεταβλητή που "δείχνει" θέση σε ένα πίνακα.

Ο pointer `pi` δείχνει (ή θα δείξει) σε μια διεύθυνση μνήμης. Το περιεχόμενο της διεύθυνσης αυτής μεταφράζεται μέσω του pi σε ακέραιη τιμή. Το ίδιο συμβαίνει και με τον pj. Αντίθετα, οι pc και ps "βλέπουν" το περιεχόμενο της μνήμης στην οποία δείχνουν ως χαρακτήρα.

Ένας τρόπος για να λάβει τιμή ένας pointer είναι να του εκχωρηθεί το αποτέλεσμα του **τελεστή αναφοράς ή διεύθυνσης (reference operator ή address of operator), &**. Ο τελεστής αναφοράς επιστρέφει την διεύθυνση της πραγματικής παραμέτρου του, δηλ. την θέση μνήμης στην οποία αυτή (η πραγματική παράμετρος) έχει κατανεμηθεί. Πιο συγκεκριμένα, ο τελεστής αναφοράς έχει το ακόλουθο interface: `T* operator&(T& a)`, δηλ. η διεύθυνση ενός τύπου T είναι ένας pointer σε T, π.χ.

```
int i=1;
int *pi=&i;
```

ο pointer pi "δείχνει" στην θέση μνήμης που έχει κατανεμηθεί για τον ακέραιο i. Η επόμενη γραμμή

```
cout << "adress at " << pi << " contains " << *pi << endl ;
```

τυπώνει το περιεχόμενο του pi που είναι η διεύθυνση του i και το περιεχόμενο του i. Όπως φαίνεται όταν θέλουμε να πάρουμε το περιεχόμενο μνήμης στην οποία δείχνει ένας pointer, τότε χρησιμοποιούμε τον **τελεστή αποπαραπομπής ή ανακατεύθυνσης (dereference operator ή indirection operator)**, * με παράμετρο τον pointer. Επομένως, η ανακατεύθυνση ενός pointer αξιολογείται στο περιεχόμενο της θέσης μνήμης στην οποία ο pointer "δείχνει". Στην συνέχεια το *pi, δηλ. το i αυξάνεται κατά 1 και "τυπώνεται".

```
(*pi)++;
cout << "adress at " << pi << " contains " << i << endl ;
```

8.1 Η αριθμητική των δεικτών (pointer arithmetic)

Η μνήμη του υπολογιστή μπορεί να θεωρηθεί ως μια ακολουθία από στοιχειώδεις θέσεις μνήμης κάθε μια από τις οποίες καταλαμβάνει το ελάχιστο μέγεθος που ένας υπολογιστής μπορεί να διαχειρισθεί (1 byte). Αυτές οι θέσεις είναι αριθμημένες σειριακά, δηλ. σε κάθε θέση αντιστοιχεί ένας ακέραιος από το 0 έως το συνολικό μέγεθος (σε bytes) που συνιστά την διεύθυνση της θέσης. Έτσι, ο pointer στον οποίο αποθηκεύονται τέτοιες διευθύνσεις είναι ένας ακέραιος τύπος. Η αριθμητική των pointers περιλαμβάνει τους διάφορους τύπους πρόσθεσης, +, ++, += και αφαίρεσης, -, --, -=. Οι πράξεις αυτές ορίζονται μεταξύ pointers και non-pointer ακέραιους τύπους, π.χ.

```
int t[]= {1,2,3,4,5,6,7,8,9,10};
int * p=&t[0];
cout << p << " " << p+1 <<
     << *p << " " << *(p+1) << endl;
```

Στον παραπάνω κώδικα, ο pointer p αρχικοποιείται να δείχνει στο μηδενικό στοιχείο του πίνακα t. Στην συνέχεια τυπώνεται η διεύθυνση αυτού του ακεραίου, `&t[0]` και του επόμενου, `&t[1]`, δηλ. η τιμή p+1 είναι αυξημένη κατά **`sizeof(int)`**. Με άλλα λόγια η έκφραση

`((int)p + sizeof(int)==(int)(p+1))` είναι αληθής. Γενικότερα, η πρόσθεση ενός ακέραιου n σε T* αυξάνει το περιεχόμενο του pointer κατά n*sizeOf(T). Αντίστοιχη λογική ισχύει για τις υπόλοιπους τύπους πρόσθεσης και αφαίρεσης.

Όταν θέλουμε να δηλώσουμε ότι ένας pointer δεν αναφέρεται σε θέση μνήμης τότε εκχωρούμαι σε αυτόν την τιμή 0. Ο pointer αυτός ονομάζεται **null** pointer.

8.2 Pointer σε void (void*)

Ένας ειδικός τύπος pointer είναι ο void*. Οι void* δείχνουν μια θέση μνήμης αλλά δεν διαθέτουν πληροφορίες τύπου για αυτήν. Έτσι για να διαβάσουμε τα περιεχόμενα της θέσης που δείχνει ένας void* θα πρέπει να προβούμε σε κατάλληλη μετατροπή τύπου (type casting).

```
void increase (void* data, int psize) {
  if ( psize == sizeof (char ) ) {
      char * pchar ;
      pchar =(char *)data;
      ++(* pchar );
  }
  else if (psize == sizeof (int) ) {
      int* pint ;
      pint =(int*)data;
      ++(* pint );
  }
}

void tstVoid () {
  char a = 'x' ;
  int b = 1602 ;
  increase (&a,sizeof (a));
  increase (&b,sizeof (b));
  cout << a << ", " << b << endl ;
}
```

8.3 Pointers και πίνακες

Ας σημειωθεί ότι το αναγνωριστικό (identifier) ενός πίνακα αξιολογείται στην διεύθυνση της πρώτης θέσης του πίνακα. Με αυτήν την έννοια μπορούμε να εκχωρήσουμε ένα πίνακα σε κατάλληλο pointer. Προσοχή, η αντίστροφη εκχώρηση δεν είναι δυνατή.

```
void arrayAndPointer() {
    int array []={ 1,2,3,4,5};
    int * p=array ;
    array=p;
    for (int i=0; i<5; i++) cout << p[i] << " " ;
    cout << endl ;
}
```

8.4 Pointers σε συναρτήσεις

Η C++ υποστηρίζει pointers σε συναρτήσεις. Κυρίως χρησιμοποιούνται για να περνάμε σε μια συνάρτηση μια άλλη συνάρτηση ως παράμετρο.

```
int addition(int,int);
int addition (int a, int b) { return (a+b); }

int subtraction (int a, int b) { return (a-b); }

int operation (int x, int y, int (*func)(int,int)) {
  int g;
  g = (*func)(x,y);
  return (g);
}

void PointerToFunc () {
  int m,n;
  int (*minus)(int,int) = subtraction;
  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  cout <<n;
}
```

Στο παραπάνω παράδειγμα, η operation δέχεται ως δεύτερη παράμετρο μια συνάρτηση που επιστρέφει int και δέχεται δύο int παραμέτρους τιμής. Στην πρώτη κλήση περνάμε ως πραγματική παράμετρο την addition και στην δεύτερη την minus.

8.5 Ασκήσεις

1. Να γράψετε ένα πρόγραμμα για την αποθήκευση της βαθμολογίας 3 φοιτητών σε 4 διαφορετικά test και να επεξεργαστείτε τα στοιχεία αυτά για την εξαγωγή απλών στατιστικών.

- Υπολογισμός του μικρότερου βαθμού
- Υπολογισμός του μέγιστου βαθμού
- Υπολογισμός της μέσης βαθμολογίας κάθε φοιτητού

Να χρησιμοποιηθεί ένας δυσδιάστατος πίνακας στον οποίο οι γραμμές θα αναπαριστούν τους φοιτητές και οι στήλες τους βαθμούς. (Λύση : l3.ap1.pdf)

2. Να ορίσετε έναν μονοδιάστατο πίνακα ακεραίων τεσσάρων θέσεων. Να εισάγετε ακέραιες τιμές στον πίνακα. Στη συνέχεια να προσπελάσετε (εμφάνιση με cout) τις τιμές του πίνακα με τέσσερις διαφορετικούς τρόπους. (Λύση : l3.ap2.pdf)

8.6 Δυναμική Παραχώρηση Μνήμης (Dynamic Memory Allocation)

Η μνήμη στην οποία κατανέμονται οι τοπικές μεταβλητές είναι οργανωμένη ως **LIFO (Last In First Out)** δομή και για αυτόν τον λόγο ονομάζεται **stack**. Πράγματι, μια LIFO δομή

εξυπηρετεί την εύκολη και ταχύτατη διαχείριση των τοπικών μεταβλητών. Έστω η συνάρτηση f() καλεί μια συνάρτηση g() και η g() την k(). Όταν κληθεί η g(), οι τοπικές μεταβλητές της f() τοποθετούνται στην κορυφή του stack. Στην συνέχεια, όταν κληθεί η k(), οι τοπικές μεταβλητές της g() επίσης τοποθετούνται στην κορυφή του stack. Με την ολοκλήρωση της k(), ο έλεγχος ροής επανέρχεται στην g() αφού οι τοπικές μεταβλητές της αντληθούν από την κορυφή του stack. Έτσι, στην κορυφή του stack βρίσκονται τώρα οι μεταβλητές της f() οι οποίες αντλούνται από την κορυφή με το πέρας της g() και την συνέχιση εκτέλεσης της f().

Για παράδειγμα θεωρείστε τις ακόλουθες f() και g():

```
void g() { int i=0;}
void f() {
  int j=0;

  g();
  cout << j;
}
```

Όταν καλείται η g() η μεταβλητή j (μαζί με άλλες πληροφορίες) κατανέμεται στην κορυφή του stack. Στην συνέχεια καλείται η g() η οποία εργάζεται με την i. Με το τέλος της g(), η i καταστρέφεται, από την κορυφή του stack αντλείται η j και συνεχίζει η εκτέλεση της f().

Η μνήμη στην οποία κατανέμονται οι καθολικές και οι στατικές μεταβλητές ονομάζεται static. Ας σημειωθεί ότι τόσο για το stack όσο και για την static μνήμη κατανέμεται σε ένα πρόγραμμα (κατά την εκτέλεσή του) ένα τμήμα της μνήμης του H\Y. Το τμήμα αυτό είναι σταθερό καθόλη την διάρκεια εκτέλεσης. Την υπόλοιπη διαθέσιμη μνήμη του υπολογιστή μπορούμε να την προσπελάσουμε διαμέσου της **δυναμικής παραχώρησης της μνήμης**.

Η δυναμική παραχώρηση της μνήμης βασίζεται στους τελεστές new και delete καθώς και στους pointers.

8.7 Ο τελεστής new

Ο τελεστής **new** λαμβάνει ως παράμετρο ένα τύπο δεδομένων και απευθύνει μια αίτηση στο λειτουργικό σύστημα για παραχώρηση δυναμικής μνήμης μεγέθους ίσου με το μέγεθος της παραμέτρου του, π.χ. η κλήση **new int** απευθύνει αίτηση στο λειτουργικό σύστημα για δέσμευση μνήμης ίσης με το μέγεθος ενός ακεραίου. Εάν η μνήμη αυτή δεσμευθεί επιτυχώς, ο τελεστής **new** επιστρέφει κατάλληλο pointer, π.χ. `int* p=new int;`

Ο τελεστής **new** χρησιμοποιείται και για δυναμική κατανομή πινάκων, π.χ. `int* tbl=new int[10];`

8.8 Ο τελεστής delete

Κάθε αντικείμενο που έχει κατανεμηθεί (allocate) με τον τελεστή **new**, όταν δεν χρειάζεται πια, θα πρέπει να αποδεσμευθεί (deallocate) με τον τελεστή **delete**, π.χ. **delete** p και για πίνακες **delete** [] tbl;

Ας σημειωθεί ότι η δυναμική μνήμη είναι γνωστή και ως σωρός (heap).

8.9 Ασκήσεις – Δυναμική παραχώρηση μνήμης

Να αναπτυχθεί βιβλιοθήκη συναρτήσεων διαχείρισης δυσδιάστατων πινάκων ακεραίων κατανεμημένων στην δυναμική μνήμη. Ένας δυσδιάστατος πίνακας m γραμμών και n στηλών θα αναπαρίσταται ως μονοδιάστατος πίνακας μεγέθους mxn. Πιο συγκεκριμένα, η βιβλιοθήκη θα περιλαμβάνει τις ακόλουθες συναρτήσεις :

```
int* create(int noOfLns, int noOfClmns);
void set (int* array ,int noOfClmns, int ln , int clmn ,
        int value );
int get (int* array ,int noOfClmns, int ln , int clmn );
void resize (int*& array , int noOfLns, int noOfClmns,
           int newNoOfLns, int newNoOfClmns );
void prt (int* array , int noOfLns, int noOfClmns, int
width );
```

Να αναπτυχθεί βιβλιοθήκη συναρτήσεων διαχείρισης δυσδιάστατων πινάκων ακεραίων κατανεμημένων στην δυναμική μνήμη. Ένας δυσδιάστατος πίνακας m γραμμών και n στηλών θα αναπαρίσταται ως μονοδιάστατος πίνακας μεγέθους m. Κάθε στοιχείο του μονοδιάστατου πίνακα θα είναι ένας int* που θα δείχνει σε ένα μονοδιάστατο πίνακα ακεραίων μεγέθους n. Πιο συγκεκριμένα, η βιβλιοθήκη θα περιλαμβάνει τις ακόλουθες συναρτήσεις :

```
int** create(int noOfLns, int noOfClmns);
void set(int** array, int ln, int clmn, int value);
int get(int** array, int ln, int clmn);
void resize(int**& array, int noOfLns, int noOfClmns, int
newNoOfLns, int newNoOfClmns);
void prt(int** array, int noOfLns, int noOfClmns, int
width);
void deletePtoP(int** array, int noOfClmns);
```

Να δοθεί κατάλληλος τεστ κώδικας.

8.10 Λύσεις

<u>1η Άσκηση</u>

```
//int2DimArrayPtoInt.h

#ifndef INT2DIMARRAYPTOINT_H_INCLUDED
#define INT2DIMARRAYPTOINT_H_INCLUDED
namespace pToInt {
    int * create ( int noOfLns , int noOfClmns );
    void set ( int * array , int noOfClmns , int ln , int clmn
    , int value );
    int get ( int * array , int noOfClmns , int ln , int clmn
    );
    void resize ( int *& array , int noOfLns , int noOfClmns ,
    int newNoOfLns
    , int newNoOfClmns );
```

```
    void prt ( int * array , int noOfLns , int noOfClmns , int
    width );
}
#endif // INT2DIMARRAYPTOINT_H_INCLUDED
```

```
//int2DimArrayPtoInt.cpp

#include "int2DimArrayPtoInt.h"
#include <iostream>
using namespace std;
int* pToInt::create(int noOfLns, int noOfClmns) {
      return new int[noOfLns*noOfClmns];
}

void pToInt::set(int* array, int noOfClmns, int ln, int clmn,
                 int     value) {
      array[ln*noOfClmns+clmn]=value;
}

int pToInt::get(int* array,int noOfClmns, int ln, int clmn) {
      return array[ln*noOfClmns+clmn];
}

void pToInt::resize(int*& array, int noOfLns, int noOfClmns,
int newNoOfLns, int newNoOfClmns) {
      int* local = new int[newNoOfLns*newNoOfClmns];
      for (int i = 0; i < newNoOfLns && i < noOfLns; i++)
      for (int j = 0; j < newNoOfClmns && j < noOfClmns; j++)
      local[i * newNoOfClmns + j] = array[i * noOfClmns + j];
      delete [] array;
      array = local;
}

void pToInt::prt(int* array, int noOfLns, int noOfClmns, int
width) {
      for (int i = 0; i < noOfLns; i++) {
            for (int j = 0; j < noOfClmns; j++) {
            cout.width(width);
            cout << array[i*noOfClmns+j];
            }
            cout << endl;
      }
      cout << endl;
}
```

Test κώδικας

```
void tst_int2DimArrayPToInt() {
      using namespace pToInt;
      int* array=create(2,3);
      for (int i=0; i<2; i++)
            for (int j=0; j<3; j++) set(array, 3, i,j,i*3+j);
```

```
        prt(array,2,3,4);
        resize(array,2,3,3,3);
        for (int j=0; j<3; j++) set(array,3,2,j,2*3+j);
        prt(array,3,3,4);
        delete [] array;
}
```

Άσκηση 2

```
//int2DimArrayPtoP.h

#ifndef INT2DIMARRAYPTOP_H_INCLUDED
#define INT2DIMARRAYPTOP_H_INCLUDED
namespace pToP {
        int ** create ( int noOfLns , int noOfClmns );
        void set ( int ** array , int ln , int clmn , int value
        );
        int get ( int ** array , int ln , int clmn );
        void resize ( int **& array , int noOfLns , int noOfClmns
        , int
        newNoOfLns , int newNoOfClmns );
        void prt ( int ** array , int noOfLns , int noOfClmns ,
        int width );
        void deletePtoP ( int ** array , int noOfClmns );
}
#endif // INT2DIMARRAYPTOP_H_INCLUDED
```

```
//int2DimArrayPtoP.cpp

#include "int2DimArrayPtoP.h"
#include <iostream>
using namespace std;

int** pToP::create(int noOfLns, int noOfClmns) {
     int** rVal=new int*[noOfLns];
     for (int i=0; i<noOfLns; i++) rVal[i]=new int[noOfClmns];
     return rVal;
}

void pToP::set(int** array, int ln, int clmn, int value) {
     array[ln][clmn]=value;
}

int pToP::get(int** array, int ln, int clmn) {
     return array[ln][clmn];
}

void pToP::resize(int**& array, int noOfLns, int noOfClmns,
                  int newNoOfLns, int newNoOfClmns) {
     int** l=create(newNoOfLns, newNoOfClmns);
     for (int i=0; i<noOfLns && i<newNoOfLns; i++)
        for (int j=0; j<noOfClmns && j<newNoOfClmns; j++)
           l[i][j]=array[i][j];
     deletePtoP(array, noOfClmns);
```

```
    array=l;
}

void pToP::prt(int** array, int noOfLns, int noOfClmns, int
width) {
      for (int i = 0; i < noOfLns; i++) {
            for (int j = 0; j < noOfClmns; j++) {
            cout.width(width);
            cout << array[i][j];
            }
            cout << endl;
      }
      cout << endl;
}

void pToP::deletePtoP(int** array, int noOfClmns) {
      for (int i=0; i<noOfClmns; i++) delete [] array[i];
      delete [] array;
}
```

Test κώδικας

```
void tst_int2DimArrayPToP() {
      using namespace pToP;
      int** array=create(2,3);
      for (int i=0; i<2; i++)
            for (int j=0; j<3; j++) set(array, i,j,i*3+j);
      prt(array,2,3,4);
      resize(array,2,3,3,3);
      for (int j=0; j<3; j++) set(array,2,j,2*3+j);
      prt(array,3,3,4);
      deletePtoP(array,3);
}
```

9 Αλφαριθμητικές σειρές (character strings)

Περίληψη: Η αλφαριθμητική σειρά (character string ή απλούστερα string) είναι μια σειρά από χαρακτήρες τους οποίους διαχειριζόμαστε ως ενιαίο σύνολο. Η αλφαριθμητική σειρά είναι ένας αναγκαίος τύπος δεδομένων που υποστηρίζεται από τις περισσότερες γλώσσες προγραμματισμού. Κυρίως χρησιμοποιείται για την αποθήκευση και διαχείριση λεκτικών, π.χ. ονόματα, περιγραφές αντικειμένων, κλπ. Στην C++ έχουμε δύο διαφορετικούς τύπους για την αναπαράσταση των σειρών, τα παραδοσιακά (conventional) strings που προέρχονται από την C και την κλάση string που έχει αναπτυχθεί στο πλαίσιο του αντικειμενοστραφούς υποδείγματος. Στην ενότητα αυτή η συζήτηση περιστρέφεται αποκλειστικά στις παραδοσιακές σειρές.

Λέξεις κλειδιά *: character strings, null terminated strings, c strings*

Η αλφαριθμητική σειρά (character string ή απλούστερα string) είναι μια σειρά από χαρακτήρες τους οποίους διαχειριζόμαστε ως ενιαίο σύνολο. Η αλφαριθμητική σειρά είναι ένας αναγκαίος τύπος δεδομένων που υποστηρίζεται από τις περισσότερες γλώσσες προγραμματισμού. Κυρίως χρησιμοποιείται για την αποθήκευση και διαχείριση λεκτικών, π.χ. ονόματα, περιγραφές αντικειμένων, κλπ.

Στην C++ έχουμε δύο διαφορετικούς τύπους για την αναπαράσταση των σειρών, τα παραδοσιακά (conventional) strings που προέρχονται από την C και την κλάση string που έχει αναπτυχθεί στο πλαίσιο του αντικειμενοστραφούς υποδείγματος. Στην ενότητα αυτή η συζήτηση περιστρέφεται αποκλειστικά στις παραδοσιακές σειρές, ενώ η παρουσίαση της κλάσης string θα γίνει στον Αντικειμενοστραφή Προγραμματισμό.

9.1 Παραδοσιακές αλφαριθμητικές σειρές

Η C παραδοσιακά αναπαριστά τις σειρές ως πίνακες χαρακτήρων στους οποίους το τέλος των έγκυρων χαρακτήρων σημειώνεται με τον χαρακτήρα null (άκυρος χαρακτήρας), '\0'. Για παράδειγμα, σε ένα πίνακα μήκους x χαρακτήρων μπορεί να αποθηκευτεί οποιαδήποτε σειρά με μέγεθος x-1, δηλ. πλήθος έγκυρων χαρακτήρων, από 1 έως x-1. Στην περίπτωση που στον προαναφερόμενο πίνακα αποθηκευτεί 1 χαρακτήρας, τότε στην θέση του πίνακα με δείκτη 1 πρέπει να αποθηκευτεί ο null χαρακτήρας. Στην περίπτωση που αποθηκευτούν x-1 χαρακτήρες, δηλ. καταληφθούν οι θέσεις 0 έως x-2, τότε ο null χαρακτήρας πρέπει να τοποθετηθεί στην θέση x-1.

Οι σειρές αυτές είναι γνωστές ως τερματιζόμενες με null σειρές (null-terminated strings). Εφόσον οι τερματιζόμενες με null σειρές, υλοποιούνται ως πίνακες χαρακτήρων, ότι ισχύει για τους μονοδιάστατους πίνακες και τους pointers, ισχύει και για τις τερματιζόμενες με null σειρές. Επιπλέον, μία σειρά από χαρακτήρες που περιβάλλεται από διπλή απόστροφο (double quotes) συνιστά μια τερματιζόμενη με null σειρά. Τέλος, ένα σύνολο από προ-ορισμένες (predefined) συναρτήσεις προσφέρονται στην βιβλιοθήκη <string.h> για την διαχείριση αυτών των σειρών.

Στο εξής, οι τερματιζόμενες με null σειρές θα αποκαλούνται για λόγους συντομίας σειρές χαρακτήρων ή/και απλά σειρές. Ακολουθεί παράδειγμα ορισμού και εξόδου στην οθόνη σειρών χαρακτήρων.

```
#include <iostream>
using namespace std;

main() {
    char s1[]={'L','e','f','t','e','r','i','s','\0'};
    char* s2=s1;
    char s3[9]="Lefteris";

    cout << s1 << " " << s2 << " " << s3 << endl;

    s1[0]='l';
    cout << s1 << " " << s2 << " " << s3 << endl;
}
```

Στο παραπάνω παράδειγμα ορίζεται η σειρά **s1**, ένας pointer **s2** που αρχικοποιείται να δείχνει στην αρχή του **s1** και ένας πίνακας **s3** που αρχικοποιείται στην ίδια τιμή με το **s1**, ωστόσο του κατανέμεται χωριστή μνήμη .

Στην συνέχεια παρουσιάζεται ένα βασικό υποσύνολο των συναρτήσεων διαχείρισης σειρών που περιλαμβάνονται στην βιβλιοθήκη <string.h>.

- **char*** strcpy(**char*** target, const **char*** source). Αντιγράφει τα περιεχόμενα του source στο target. **T**ο target θα πρέπει να έχει μέγεθος που επαρκεί για το source. Επιστρέφει το target.
- **char*** strcat(**char*** dest, const **char*** source). Προσθέτει (appends) ένα αντίγραφο του source στο dest(ination). Σημειώνεται πως ο πρώτος χαρακτήρας του source γράφεται επάνω στον null χαρακτήρα του dest. Επιστρέφει το dest . Το dest θα πρέπει να επαρκεί για να συμπεριλάβει το source. Η πράξη αυτή της ένωσης δύο σειρών είναι γνωστή ως concatenation.
- **int** strcmp(const **char*** s1, const **char*** s2). Συγκρίνει τα περιεχόμενα δύο σειρών. Επιστρέφει 0 αν s1==s2, τιμή >0 αν s1>s2 και τιμή <0 αν s1<s2

Παράδειγμα

```
#include <iostream>
#include <string.h>
using namespace std;

main() {
    char s1[9],s2[4],*s3, s4[9]="nice day";

    strcpy(s1,"nice "); //string constant
    strcpy(s2,"day");   //must seat var

    s3=strcat(s1,s2); //unbounded concatenation
    cout << s1 << " "
         << s2 << " "
         << s3 << endl;

    int cmp01=strcmp(s1,s2), //unbounded
        cmp02=strcmp(s2,s1), //string
        cmp03=strcmp(s3,s4); //comparison

    cout << cmp01 << " "
           << cmp02 << " "
           << cmp03 << endl;
}
```

Στο παραπάνω **παράδειγμα 2** οι σταθερές σειρές (string constants) “**nice** “ & “**day**” αντιγράφονται στις μεταβλητές **s1** & **s2** αντίστοιχα. Αμέσως μετά προστίθενται και ένας pointer **s3** τοποθετείται να δείχνει στο αποτέλεσμα. Τέλος, ακολουθούν 3 συγκρίσεις, που επιστρέφουν τρείς ακεραίους (**cmp01=10,cmp02=-10,cmp03=0**). Ο θετικός ακέραιος δείχνει

ότι το **s1** είναι μεγαλύτερο του **s2**, ο αρνητικός ότι το **s2** είναι μικρότερο του **s1** και το 0 δείχνει ότι το **s3** είναι ίσο με το **s4**.

Οι συναρτήσεις διαχείρισης των strings συχνά είναι υλοποιημένες σε δύο εκδόσεις, με (**bounded**) ή χωρίς (**unbounded**) όρια, π.χ. strcpy(**char***, const **char***) (unbounded) και η **char*** strncpy (**char** *, const **char***, **size_t**) (bounded).

Στο ακόλουθο παράδειγμα η **size_t** strlen **(** const **char** ***** str **)** επιστρέφει το μήκος σε χαρακτήρες της σειράς, str; Σημειώστε πως η **size_t** είναι ένας μη προσημασμένος ακέραιος τύπος και χρησιμοποιείται για την μέτρηση της μνήμης σε χαρακτήρες (char).

```
#include <iostream>
#include <string.h>
using namespace std;

main() {
  char s1[]="Good morning";
  char s2[]="Nice";
  strncpy(s1,s2,strlen(s2));    //bounded C string copy
  cout << s1 << endl;
}
```

Στο προηγούμενο παράδειγμα **strlen(s2)** χαρακτήρες αντιγράφονται από το **s2** στο **s1**.

Μια πλήρη σειρά των συναρτήσεων της string.h θα βρείτε εδώ.

Σημείωση: Ένας άλλος τρόπος διαχείρισης των αλφαριθμητικών σειρών είναι η κλάση string που παρέχει η C++. Η κλάση αυτή παρουσιάζεται στα πλαίσια του Αντικειμενοστραφούς Προγραμματισμού.

9.2 Μετατροπή μεταξύ σειρών και αριθμών

Συχνά στα προγράμματά μας καλούμαστε να μετατρέψουμε μεταξύ σειρών και αριθμών. Οι ακόλουθες συναρτήσεις της cstdlib.h μπορεί να χρησιμοποιηθούν για την μετατροπή από σειρά σε αριθμό.

```
int atoi ( const char * str );
long int atol ( const char * str );
double atof ( const char * str );
```

Προσοχή, το μειονέκτημα αυτών των συναρτήσεων είναι ότι όταν η μετατροπή είναι αδύνατη, δηλ. η σειρά περιέχει χαρακτήρες που δεν είναι ψηφία, τότε επιστρέφουν 0. Επομένως, δεν διαφοροποιούν μεταξύ μιας σειράς που αντιστοιχεί στο 0 και μιας σειράς που δεν μετατρέπεται σε αριθμητικό τύπο.

Για την μετατροπή από αριθμητικό τύπο σε σειρά δεν υπάρχουν συναρτήσεις ορισμένες στην ANSI-C ή την C++. Διάφοροι μεταγλωττιστές παρέχουν υλοποιήσεις σχετικών συναρτήσεων που όμως δεν είναι standard. Για τον λόγο αυτό δίνεται η ακόλουθη βιβλιοθήκη που περιλαμβάνει σχετικές συναρτήσεις σε standard C++.

//cStrConvertion.h

```
#ifndef CSTRCONVERTIONS_H_INCLUDED
#define CSTRCONVERTIONS_H_INCLUDED
#include <iostream>
    char * itoa ( int i );
    char * ltoa ( long int l );
    char * dtoa ( double d );
#endif // CSTRCONVERTIONS_H_INCLUDED
```

```
#include "cStrConvertions.h"
#include <sstream>
#include <string.h>

char* itoa(int i) {
    std::ostringstream strs;
    strs << i;
    std::string str = strs.str();
    char* c= new char [strs.str().size()+1];
    return strcpy(c,strs.str().c_str());
}

char* ltoa(long int l) {
    std::ostringstream strs;
    strs << l;
    std::string str = strs.str();
    char* c= new char [strs.str().size()+1];
    return strcpy(c,strs.str().c_str());
}

char* dtoa(double d) {
   std::ostringstream strs;
   strs << d;
   std::string str = strs.str();
   char* c= new char [strs.str().size()+1];
   return strcpy(c,strs.str().c_str());
}
```

Προσέξτε πως οι συναρτήσεις αυτές κατανέμουν δυναμικά τους pointers που επιστρέφουν. Επομένως, στο περιβάλλον κλήσης οι pointers αυτοί θα πρέπει να αποδεσμεύονται.

Ακολουθεί παράδειγμα χρήσης.

```
#include <iostream>
#include <cstring>
#include <stdlib.h>
#include "cStrConvertions.h"
using namespace std;

int main() {
   char ic[]="12345";
   char dc[]="123.4";
   char lc[]="99999999";
```

```
  int i= atoi( ic);
  double d= atof( dc);
  long int l = atol( lc);

  char * nic= itoa( i);
  char * ndc= dtoa( d);
  char * nlc = ltoa ( l );
  cout << nic << end
       << ndc << endl
       << nlc << endl;

  delete nic;
  delete ndc;
  delete nlc ;
}
```

9.3 Ασκήσεις – Αλφαριθμητικές σειρές

Να αναπτυχθεί βιβλιοθήκη συναρτήσεων που υλοποιεί τις συναρτήσεις

```
size_t strlen (const char * str);
char* strcpy (char* target, const char* source);
char* strcat (char* dest, const char* source);
int strcmp (const char* s1, const char* s2);
```

Οι προδιαγραφές των συναρτήσεων δίνονται στην σχετική θεωρία. Σημειώνεται ότι το `size_t` είναι μη προσημασμένος ακέραιος τύπος που ορίζεται στην iostream, namespace std. Να δοθεί κατάλληλος τεστ κώδικας.

9.4 Λύση

//myString.h

```
#ifndef MYSTRING_H_INCLUDED
#define MYSTRING_H_INCLUDED
#include <iostream>
    std :: size_t strlen ( const char * str );
    char * strcpy ( char * target , const char * source );
    char * strcat ( char * dest , const char * source );
    int strcmp ( const char * s1 , const char * s2 );
#endif // MYSTRING_H_INCLUDED
```

```
//myString.cpp

#include "myString.h"

size_t strlen ( const char * str ) {
    size_t rVal=-1;
    while (str[++rVal]!='\0');
    return rVal;
}
```

```
char* strcpy(char* target, const char* source) {
     for    (unsigned    int    i=0;    i<=strlen(source);    i++)
     target[i]=source[i];
     return target;
}

char* strcat(char* dest, const char* source) {
     int base=strlen(dest);
     for    (unsigned    int    i=0;    i<=strlen(source);    i++)
     dest[base+i]=source[i];
     return dest;
}

int strcmp(const char* s1, const char* s2) {
     int idx=0;
     while ((s1[idx]==s2[idx]) && s1[idx]!='\0') idx++;
     if (s1[idx]=='\0' && s2[idx]=='\0') return 0;
     return s1[idx]-s2[idx];
}
```

Τεστ κώδικας

```
void tstMyString() {
     char fN[]="Lefteris", sN[]="Moussiades";
     char cN[80], bN[80];
     strcpy(cN,fN);
     strcat(strcat(cN," "), sN);
     strcpy(bN,cN);

     cout << fN << endl
          << sN << endl
          << cN << endl
          << bN << endl << endl;

     cout << strcmp(fN,sN) << endl
          << strcmp(cN,fN) << endl
          << strcmp(cN,bN);
}
```

10 Αναδρομή (Recursion)

Περίληψη : Στην ενότητα αυτή αναλύεται σε βάθος η αναδρομή. Ποια είναι η βασική ιδέα; Ποιοι περιορισμοί πρέπει να τηρούνται; Πως πρέπει να σκεφτόμαστε για να βρίσκουμε αναδρομικές λύσεις σε σύνθετα προβλήματα;

Λέξεις κλειδιά *: αναδρομή (recursion)*

Αναδρομή έχουμε όταν μια συνάρτηση καλεί τον εαυτό της, π.χ. στον ορισμό της συνάρτησης f υπάρχει μια τουλάχιστον κλήση στην f. Η αναδρομή μπορεί να είναι άμεση,

π.χ. η f καλεί άμεσα την f, ή έμμεση, π.χ. η f καλεί την g ή οποία με την σειρά της καλεί την f.

Παράδειγμα

```
void endless (int n) {
  cout << n << endl;
  endless(n- 1);
}
```

Στο παραπάνω παράδειγμα η endless τυπώνει την παράμετρό της n και καλεί τον εαυτό της με n-1. Επομένως, αν η endless κληθεί με παράμετρο, π.χ. 10, θα τυπώσει 10, 9, 8, 7, 6, ….. Μέχρι ποιόν αριθμό θα φτάσει; Στην πράξη, θα ρίξει το πρόγραμμα καθώς η αναδρομή γίνεται χωρίς κανέναν περιορισμό, γεγονός που καταλήγει σε ατέρμονες (αναδρομικές) κλήσεις. Επομένως, ακριβώς όπως οι επαναληπτικές δομές βασίζονται σε μια συνθήκη τερματισμού της επανάληψης· έτσι και οι αναδρομικές συναρτήσεις πρέπει να περιέχουν κάποια συνθήκη τερματισμού της αναδρομής, την αποκαλούμενη βασική περίπτωση ή συνθήκη (**base case**). Η ύπαρξη της βασικής συνθήκης αποτελεί βασική αρχή της αναδρομής.

Παράδειγμα

```
void prtUpToOne(int n) {
  if (n<=0) return;
  cout << n << endl;
  prtUpToOne(n-1);
}
```

Η prtUpToOne τυπώνει τους ακέραιους από n έως και 1. Η βασική συνθήκη της prtUpToOne είναι η `(n<=0)`. Όταν η συνθήκη (n<=0) αληθεύει η αναδρομή τερματίζεται (πρώτη γραμμή). Ας ακολουθήσουμε την εκτέλεση της prtUpToOne προκειμένου να μελετήσουμε την αναδρομή. Αρχικά, ας θεωρήσουμε την κλήση prtUpToOne(0). Στην περίπτωση αυτή είναι προφανές ότι η συνάρτηση δεν έχει κανένα αποτέλεσμα. Για την κλήση με n=1, έχουμε :

- Η βασική συνθήκη είναι ψευδής
- τυπώνεται το 1
- Καλείται η prtUpToOne με n-1, δηλ. με 0
  - Η βασική συνθήκη είναι αληθής και η κλήση prtUpToOne(0) τερματίζει
- Συνεχίζει η εκτέλεση στο περιβάλλον κλήσης, δηλ. στην prtUpToOne(1) η οποία και τερματίζει.

Παράδειγμα

```
void prtSym(int n) {
      if (n<=0) return;
      cout << n << " ";
      prtSym(n-1);
      cout << n << " ";
}
```

Η κλήση prtSym(3) έχει ως αποτέλεσμα να τυπωθούν οι ακέραιοι: 3 2 1 1 2 3. Ακολουθεί η ιχνηλάτηση της κλήσης prtSym(3).

prtSym(3)
Συνθήκη ψευδής
Εκτύπωση του 3
Κλήση prtSym(2)
Συνθήκη ψευδής
Εκτύπωση του 2
Κλήση prtSym(1)
Συνθήκη ψευδής
Εκτύπωση του 1
Κλήση prtSym(0)
Συνθήκη αληθής
τερματισμός της prtSym(0)
Συνέχεια της prtSym(1),
Εκτύπωση του 1
τερματισμός της prtSym(1)
Συνέχεια της prtSym(2)
Εκτύπωση του 2
Τερματισμός της prtSym(2)
Συνέχεια της prtSym(3),
Εκτύπωση του 3
Τερματισμός της αρχικής κλήσης prtSym(3)

Η αναδρομή αποτελεί μια τεχνική επίλυσης προβλημάτων που είναι συχνά πολύ χρήσιμη. Η βασική ιδέα είναι η συνεχής αναγωγή ενός προβλήματος σε ένα απλούστερο πρόβλημα έως ότου φτάσουμε στην βασική περίπτωση της οποίας η λύση της προφανής. Η λύση της βασικής περίπτωσης τροφοδοτεί την περίπτωση της αμέσως ανώτερης τάξης, η λύση της οποίας τροφοδοτεί την λύση της ακόμη ανώτερης τάξης· κοκ, έως την λύση του αρχικού προβλήματος.

Παράδειγμα

Ο υπολογισμός του παραγοντικού (factorial). Υπενθυμίζεται ότι το παραγοντικό ενός θετικού ακεραίου n, συμβολικά n!, ισούται με το γινόμενο 1x2x…xn, π.χ. 4!=1x2x3x4. Στην συνέχεια δίνονται δύο συναρτήσεις υπολογισμού του παραγοντικού, η μια βασίζεται στην αναδρομή και η άλλη στην επαναληπτική δομή for.

```
int factorialR(int n) {
 if (n==1) return 1;
 return n*factorialR(n-1);
}

int factorialF(int n) {
 int rVal = 1;
```

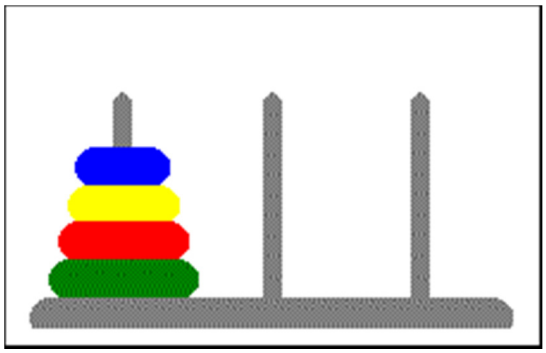
```
 for (int i = 2 ; i <= n; i ++) rVal *= i ;
 return rVal ;
}
```

Γενικότερα, τα προβλήματα που μπορούν να λυθούν με αναδρομή, μπορούν επίσης να λυθούν με την χρήση επαναληπτικών δομών. Επιπλέον, η αναδρομή συνήθως απαιτεί περισσότερη μνήμη και χρόνο εκτέλεσης, π.χ. στο παραπάνω παράδειγμα, για τον υπολογισμό του 4!, η factorialR θα κληθεί 4 φορές ενώ η factorialF μόνο 1 φορά. Ωστόσο, η αναδρομή μας επιτρέπει μια πιο αφαιρετική προσέγγιση απλοποιώντας την λύση πολλών προβλημάτων. Μια τέτοια περίπτωση συνιστά ένα πρόβλημα γνωστό ως οι πύργοι του Ανόι, που παρουσιάζεται ακολούθως.

10.1 Οι πύργοι του Ανόι (Towers of Hanoi)

Πρόκειται για ένα μαθηματικό πρόβλημα που επινόησε ο Γάλλος μαθηματικός Edouard Lucas το 1883.

Έχουμε τρείς στύλους. Τον s(source), τον d(destination) και τον a(auxiliary). Στον s στύλο είναι τοποθετημένοι n δίσκοι, ο ένας επάνω στον άλλο. Έστω ότι οι δίσκοι ονομάζονται διαδοχικά 1,2,…,n., όπου 1 ο δίσκος στην βάση και n ο δίσκος στην κορυφή και είναι διατεταγμένοι ως προς την διάμετρό τους, δηλ. ισχύει ότι η διάμετρος του δίσκου x είναι μεγαλύτερη από την διάμετρο του δίσκου $y<x$, $x, y \in 1..n$. Ζητείται να μεταφερθούν οι δίσκοι από το s στο d χρησιμοποιώντας σαν βοηθητικό τον a και μετακινώντας 1 δίσκο κάθε φορά, χωρίς ποτέ ένας δίσκος με μεγαλύτερη διάμετρο να βρεθεί πάνω από έναν δίσκο με μικρότερη διάμετρο.



Όποιος επιχειρήσει να λύσει αυτό το πρόβλημα χωρίς την βοήθεια της αναδρομής θα διαπιστώσει ότι πρόκειται για ένα ιδιαίτερα δύσκολο πρόβλημα. Αξιοποιώντας όμως την αναδρομή· μπορούμε να το λύσουμε απλούστερα, σκεπτόμενοι ως εξής:

Στην βασική περίπτωση με n=1, η λύση είναι προφανής· απλά μετακινούμε τον δίσκο νούμερο 1 από το s στο d, αλλιώς· αν δηλ. n>1

- μετακινούμε τους κορυφαίους n-1 δίσκους, δηλ. τους δίσκους 2..n, από το s στο a χρησιμοποιώντας ως βοηθητικό στύλο το d.
- στον s έχει μείνει τώρα ένας δίσκος που μπορεί να μετακινηθεί απευθείας στον d.
- στην συνέχεια μετακινούμε τους n-1 δίσκους από τον a στον d χρησιμοποιώντας ως βοηθητικό το s.

Αφού διατυπώσαμε την λύση, ας την κωδικοποιήσουμε.

```
void moveOne(int diskNo, char from, char to) {
    cout << "move disk " << diskNo <<
            " from " << from << " to " << to << endl ;
}

void moveDisks (int n , char s , char d , char a ) {
    if (n == 1 )
      moveOne(n , s , d );
    else {
      moveDisks (n - 1 , s , a , d );
      moveOne(n ,s ,d );
      moveDisks (n - 1 , a , d , s );
    }
}
```

♦

Όπως φαίνεται από το παράδειγμα των πύργων του Ανόι, η αναδρομή μπορεί σε κάποιες περιπτώσεις να δώσει κομψές και εύκολες λύσεις. Ωστόσο, το μειονέκτημά της είναι η σημαντική μνήμη stack που καταναλώνεται από τις αναδρομικές κλήσεις, οι οποίες μάλιστα δεν γνωρίζουμε πόσες θα είναι για κάθε συγκεκριμένο πρόβλημα. Ενδεικτικά, για τους πύργους του Ανόι με n=3 απαιτούνται 7 αναδρομικές κλήσεις, για n=6 απαιτούνται 63, για n=12 απαιτούνται 4095, για n=13 απαιτούνται 8191, κοκ. Συνεπώς, εύκολα μπορεί να προκληθεί υπερχείλιση του stack (stack overflow) και το πρόγραμμα να καταρρεύσει.

11 Παράμετροι και τιμή επιστροφής της main

Περίληψη : Όπως είναι γνωστό η main αποτελεί την βασική συνάρτηση εισόδου ενός προγράμματος C++. Όπως το όνομά της έτσι και οι παράμετροί της είναι προορισμένοι. Σε αυτήν την ενότητα παρουσιάζονται οι παράμετροι της main και δίνονται παραδείγματα χρήσης αυτών.

Όπως είναι γνωστό, η **main** αποτελεί το σημείο εισόδου κάθε προγράμματος C++. Όταν ένα εκτελεστέο πρόγραμμα C++ καλείται από την **γραμμή εντολών (command line)**, τότε στην πράξη εκτελείται η **main**. Έτσι, οι παράμετροι της main είναι στην ουσία οι παράμετροι του προγράμματος. Για την main, σε αντίθεση με τις υπόλοιπες συναρτήσεις, δεν δίνεται η δυνατότητα ορισμού των παραμέτρων της. Αντίθετα, οι παράμετροι της main είναι πάντα 2,

ένας int και ένας πίνακας από τερματιζόμενες με null σειρές χαρακτήρων. Πιο συγκεκριμένα, το interface της main είναι το ακόλουθο:

```
int main(int length, char** tbl);
```

Στην μηδενική θέση του πίνακα tbl, τοποθετείται το πλήρες path του προγράμματος. Στην συνέχεια, στον ίδιο πίνακα τοποθετούνται οι τιμές τυχόν επιπλέον πραγματικών παραμέτρων. Η τιμή της length μας δίνει το μήκος του πίνακα tbl.

Η τιμή επιστροφής της main είναι int. Κατά σύμβαση, η επιστροφή του 0 σημαίνει ότι όλα πήγαν “καλά” με την εκτέλεση της main. Αντίθετα, αν προκύψει κάποιο πρόβλημα η τιμή επιστροφής της main πρέπει να είναι διάφορη του 0.

Το ακόλουθο πρόγραμμα δημιουργεί αντίγραφο ενός text αρχείου.

```
//copytext.cpp
#include <iostream>
#include <cstdlib>
#include <fstream>
using namespace std ;

void syntax () {
  cout << "Syntax: copytext sourceName targetName" << endl ;
  exit (1);
}

void fileNotFound (char * fName ) {
  cout << "file " << fName << " not Found" << endl ;
  exit (2);
}

void canNotCreateFile (char * fName ) {
  cout << "can not create file: " << fName << endl ;
  exit (3);
}

int main (int noOfParam , char ** param ) {
  if (noOfParam !=3) syntax ();

  ifstream in(param [1]);
  if (!in.is_open ()) fileNotFound (param [1]);

  ofstream out (param [2]);
  if (!out.is_open ()) canNotCreateFile (param [2]);

  char line [256 ];
  while (!in.eof ()) {
    in.getline (line ,256 );
    out << line << endl ;
  }

  in.close ();
  out.close ();
  return 0;
```

```
}
```

Στο πρόγραμμα copytext γίνεται η υπόθεση ότι καμία γραμμή του πηγαίου αρχείου δεν υπερβαίνει τους 256 χαρακτήρες. Το copytext χρειάζεται 2 παραμέτρους: το όνομα του πηγαίου αρχείου και το όνομα του αρχείου στόχου. Επομένως, τα μήκος της noOfParam θα πρέπει να είναι 3, διαφορετικά το copytext δεν μπορεί να συνεχίσει την ομαλή εκτέλεσή του. Ο έλεγχος αυτός γίνεται στην πρώτη γραμμή της main. Στην συνέχεια, ανοίγεται το αρχείο με όνομα param[1] (πηγαίο αρχείο) της main και το αρχείο με όνομα param[2] (αρχείο στόχος). Ας σημειωθεί πως η param[0] μεταφέρει το όνομα του προγράμματος.

Σημείωση: Στον παραπάνω κώδικα, μπορεί κανείς να παρατηρήσει, ότι το ρεύμα in δεν θα κλείσει αν αποτύχει να ανοίξει το ρεύμα out. Κάτι τέτοιο δεν είναι αληθές καθώς τα ρεύματα in & out κλείνουν αυτομάτως με το που βγαίνουν εκτός εμβέλειας. Το πώς αυτό επιτυγχάνεται θα παρουσιάσουμε στον Αντικειμενοστραφή Προγραμματισμό.

12 Μετατροπή τύπου (type cast)

Περίληψη : Μετατροπές τύπων (type castings), αυτόματες (implicit) μετατροπές, σαφείς (explicit) μετατροπές, static_cast, reinterpret_cast, const_cast, dynamic_cast

Λέξεις κλειδιά : *Μετατροπές τύπων (type castings)*

Αυτόματη μετατροπή τύπου (implicit type casting) συμβαίνει στις περιπτώσεις που μια τιμή ενός τύπου εκχωρείται σε μεταβλητή συμβατού τύπου, π.χ.

```
short a=2000;
int b;
b=a;
```

Σε πολλές περιπτώσεις χρειάζεται η σαφής μετατροπή τύπου (explicit type casting). Το συντακτικό μετατροπής τύπου που κληρονομήθηκε από την C, έχει ως εξής:

```
T (T)object, όπου T ο νέος τύπος.
```

Στην C++, έχει προστεθεί και το ακόλουθο συντακτικό (functional notation) :

```
T  T(object)
```

Ακολούθως δίνεται παράδειγμα μετατροπής τύπου

```
int i=8, j=3;
cout << i/j << ' '
     << i/( double ) j << endl;
```

Η έξοδος αυτού του κώδικα είναι 2 2,666, καθώς η πρώτη πράξη είναι διαίρεση μεταξύ ακεραίων· επομένως, ακέραια διαίρεση· ενώ, η δεύτερη είναι πραγματική διαίρεση.

Οι μετατροπές τύπου αποτελούν συχνά πηγή σοβαρών σφαλμάτων στον κώδικά μας και θα πρέπει όπου είναι δυνατόν να αποφεύγονται.

Η μετατροπή τύπου με τον παραπάνω τρόπο παρουσιάζει 2 σοβαρά μειονεκτήματα:

- Είναι δύσκολο να εντοπιστεί μέσα στον πηγαίο κώδικα, π.χ. για το παραπάνω παράδειγμα θα έπρεπε να ξέρει κανείς να ψάξει για την δεσμευμένη λέξη **double**, προκειμένου να εντοπίσει ότι σε εκείνο το σημείο του κώδικα γίνεται μετατροπή τύπου.
- Ο μηχανισμός αυτός υποστηρίζει οποιαδήποτε μετατροπή τύπου, χωρίς να διαφοροποιεί μεταξύ pointers και μη pointers. Έτσι, δεν παρέχει προστασία απέναντι σε κάποιες μετατροπές που ενώ είναι συντακτικά σωστές και επομένως περνούν από μεταγλώττιση, καταλήγουν σε λάθη χρόνου εκτέλεσης και κατάρρευση του προγράμματος, π.χ.

```
int i=5;
int* p2 =& i;
void (* p ) ( int)= display ;
p =( void (*) ( int)) p2 ;
(* p )( 5);
```

Στο παραπάνω παράδειγμα ένας pointer σε συνάρτηση τοποθετείται να δείχνει στο **data segment**. Στην συνέχεια, κλήση της συνάρτησης στην οποία υποτίθεται ότι δείχνει ένας pointer αυτού του τύπου προκαλεί την κατάρρευση του προγράμματος.

Η λύση στα προαναφερόμενα προβλήματα, επιχειρείται να δοθεί στην C++, με την εισαγωγή ενός συνόλου συναρτήσεων μετατροπής. Πιο συγκεκριμένα, παρέχονται 4 συναρτήσεις μετατροπής τύπου, η **static_cast,** η **reinterpret_cast,** η **const_cast**, και η **dynamic_cast**:

Το συντακτικό της **static_cast** έχει ως εξής:

T static_cast<T> (object);

Η **static_cast<T>** παράγει ένα αντίγραφο του object. Ο τύπος του παραγόμενου αντικειμένου είναι **T**. Η **static_cast** χρησιμοποιείται για ασφαλείς μετατροπές, π.χ.

```
cout << i/ static_cast<double>( j).
```

Αντίθετα η ακόλουθη μετατροπή θα χτυπήσει κατά την μεταγλώττιση.

```
char myN[]="Lefteris";
int* p3=static_cast<int*> (myN);
```

Πράγματι, η μετατροπή μεταξύ μη συμβατών τύπων pointer δεν υποστηρίζεται από την **static_cast.** Εφόσον κάτι τέτοιο είναι απαραίτητο συνίσταται η χρήση της **reinterpret_cast**. Το συντακτικό της έχει ως εξής:

```
T reinterpret_cast<T> (object);
```

Η **reinterpret_cast** χρησιμοποιείται για την μετατροπή μεταξύ μη συμβατών τύπων pointer, π.χ.

```
char myN[]="Lefteris";
int * p3 = reinterpret_cast < int *> ( myN);
```

Συνήθως, το αποτέλεσμά της **reinterpret_cast** μετατρέπεται πάλι πίσω στον αρχικό τύπο. Ένα χαρακτηριστικό παράδειγμα χρήσης της **reinterpret_cast** θα δούμε στην ενότητα 19 για τα δυαδικά αρχεία.

Η **const_cast** χρησιμοποιείται για να "καταργήσουμε" ή να "θέσουμε" *φαινομενικά* την επίδραση **const** σε έναν pointer. Το συντακτικό της έχει ως εξής:

```
T const_cast<T> (object)
```

Με το ακόλουθο παράδειγμα περιγράφεται μια εφαρμογή της **const_cast**.

Έστω, η συνάρτηση

```
void f(char* s) {
      cout << s << endl;
}
```

Στην περίπτωση αυτή ο ακόλουθος κώδικας δεν μεταγλωττίζεται.

```
const char* txt="test text";
f(txt);
```

Προσέξτε, πως η f δέχεται ως παράμετρο ένα non-**const** **char***, ενώ το txt είναι **const**. Παρόλα αυτά η f δεν επιχειρεί να μεταβάλει την τιμή της παραμέτρου της. Στην περίπτωση αυτή μπορούμε να χρησιμοποιήσουμε το **const_cast.**

```
const char* txt="test text";
char* rC=const_cast<char*>(txt);
f(rC);
```

Προσοχή, η μετατροπή αυτή θα πρέπει να χρησιμοποιείται μόνον όταν η f δεν επιχειρεί μετατροπή του περιεχομένου του *s, δηλ. όταν η f κανονικά θα έπρεπε να λαμβάνει **const char***, αλλά λαμβάνει non-**const** **char***, ουσιαστικά από λάθος αυτού που την ανέπτυξε. Σε αντίθετη περίπτωση, δηλ. αν επιχειρείται μετατροπή του *s στην f, το πρόγραμμα θα πέσει.

Τέλος η **dynamic_cast**, με το ακόλουθο συντακτικό

```
T& dynamic_cast<T&> (object) ή
```

```
T* dynamic_cast<T*> (object)
```

μετατρέπει από έναν τύπο pointer ή αναφοράς σε έναν άλλον τύπο pointer ή αναφοράς, ελέγχοντας κατά τον χρόνο εκτέλεσης αν η μετατροπή αυτή είναι ορθή. Οι εφαρμογές της σχετίζονται με το αντικειμενοστραφές μοντέλο και έτσι η συζήτηση για αυτήν θα ολοκληρωθεί στα πλαίσια του Αντικειμενοστραφούς προγραμματισμού.

13 Ακαθόριστο πλήθος παραμέτρων

Σε πολλές περιπτώσεις χρειαζόμαστε συναρτήσεις που να λαμβάνουν ως παράμετρο μια λίστα τιμών. Για την ανάπτυξη τέτοιων συναρτήσεων, η C++ προσφέρει έναν μηχανισμό που βασίζεται στις μακροεντολές[11] (macro): `va_start`, `va_arg` και `va_end` και στον τύπο `va_list`, που ορίζονται στο `stdarg.h`.

Παράδειγμα

```
#include <iostream>
#include <stdarg.h>

using namespace std ;

double sum (int length, ...) {
     double rVal = 0;
     va_list v;
     va_start (v, length);
     for (int i=0; i<length ; i++) rVal+=va_arg(v,double);
     va_end(v);
     return rVal ;
}

int main () {
     cout << sum ( 3 , 2.2 , 3.3 , 5.0 )
          << sum ( 4 , 8.2 , 3.3 , 5.0, 1.5 );
}
```

Η συνάρτηση sum μπορεί να κληθεί με λίστα παραμέτρων μεταβλητού μήκους. Έτσι, στην main καλείται (πρώτη κλήση) για να προσθέσει 3 πραγματικούς και ξανακαλείται για να προσθέσει 4 πραγματικούς. Όπως φαίνεται από το παράδειγμα, η λίστα παραμέτρων μεταβλητού μήκους δηλώνεται με την βοήθεια των 3 τελειών (...). Όταν σε μια συνάρτηση υπάρχει μια τέτοια λίστα, τότε είναι απαραίτητη και μια ακέραιη παράμετρος (στο παράδειγμα η length) στην οποία περνάει το μήκος της μεταβλητής λίστας. Η πρόσβαση στην ίδια την λίστα μεταβλητού μήκους γίνεται μέσα από μεταβλητή τύπου `va_list (v)`, η οποία αρχικοποιείται με "κλήση" της `va_start`. Στην συνέχεια, η `va_arg` διαβάζει με κάθε κλήση της και μια τιμή της λίστας. Ο τύπος αυτής της τιμής

[11] Οι μακροεντολές είναι οδηγίες προς τον προ-επεξεργαστή της μορφής #define [identifier] [expression]. Περισσότερα για τις μακροεντολές στις διευθύνσεις: http://www.cprogramming.com/tutorial/cpreprocessor.html, http://www.ebyte.it/library/codesnippets/WritingCppMacros.html

καθορίζεται από την δεύτερη παράμετρο της `va_arg`. Τέλος, κάθε κλήση `va_start` θα πρέπει να συνοδεύεται και από αντίστοιχη κλήση `va_end`.

14 Υπερφόρτωση συναρτήσεων (function overloading)

Σε κάποιες περιπτώσεις μια λειτουργία εφαρμόζεται σε διαφορετικού τύπου δεδομένα, π.χ. η εύρεση του μεγαλύτερου μεταξύ δύο αριθμών μπορεί να εφαρμοσθεί σε δύο ακέραιους ή σε δύο πραγματικούς. Σε μια τέτοια περίπτωση είναι χρήσιμο, η λειτουργία να διατηρεί το όνομά της είτε αυτή εφαρμόζεται σε ακέραιους είτε σε πραγματικούς, δηλ. είναι χρήσιμο να υλοποιηθεί η λειτουργία με δύο συναρτήσεις που θα έχουν το ίδιο όνομα. Η υλοποίηση συναρτήσεων με το ίδιο όνομα είναι γνωστή ως υπερφόρτωση[12] συναρτήσεων (function overloading)

Παράδειγμα

```
double max(double x, double y) {
    if (x> y) return x;
    return y;
}

int max(int x, int y) {
    if (x> y) return x;
    return y;
}

int max (int x, int y, int z ) {
    return (max(x,max(y,z )));
}

double max(double x, int y) {
    if (x> y) return x;
    return static_cast<double> (y);
}

/* error
int max(double x, int y) {
if (x>y) return static_cast<int> (x);
return y;
}
*/

double max(int x, double y) {
    if (x> y) return static_cast < double> (x);
    return y;
}
```

[12] Άλλοι συγγραφείς αποδίδουν τον όρο function overloading ως επιφόρτωση συναρτήσεων

Στο παραπάνω παράδειγμα, παρατίθεται μια σειρά από υλοποιήσεις της συνάρτησης max. Η πρώτη υπολογίζει τον μεγαλύτερο από δύο πραγματικούς, η δεύτερη υπολογίζει τον μεγαλύτερο από δύο ακέραιους, η τρίτη υπολογίζει τον μεγαλύτερο από τρείς ακεραίους, η τέταρτη και πέμπτη υπολογίζουν τον μεγαλύτερο μεταξύ ενός ακεραίου και ενός πραγματικού. Όπως είναι προφανές η υπερφόρτωση των συναρτήσεων θα πρέπει να γίνεται κατά τέτοιο τρόπο ώστε να δίνονται στον μεταγλωττιστή ικανές πληροφορίες προκειμένου να πραγματοποιεί επίλυση της κλήσης της υπερφορτωμένης συνάρτησης. Σε σχέση με το παράδειγμά μας, όταν ο μεταγλωττιστής συναντήσει κλήση μιας max συνάρτησης θα πρέπει να είναι σε θέση να υπολογίσει η εν λόγω κλήση σε ποια συγκεκριμένη max αναφέρεται. Η διαφοροποίηση μεταξύ μιας ομάδας υπερφορτωμένων συναρτήσεων βασίζεται στην λίστα παραμέτρων. Πιο συγκεκριμένα, η λίστα παραμέτρων δύο συναρτήσεων που έχουν το ίδιο όνομα, θα πρέπει να διαφοροποιείται τουλάχιστον ως προς ένα από τα ακόλουθα:

τον αριθμό των παραμέτρων, π.χ. `max(int,int,int)` και `max(int,int)`

τον τύπο των παραμέτρων, π.χ. `max(double,double)` και `max(int,int)`

την σειρά των παραμέτρων, π.χ. `max(double,int)` και `max(int,double)`

Η υπερφόρτωση **δεν** μπορεί να βασισθεί στην διαφοροποίηση της τιμής επιστροφής, π.χ. `double max(double,int)` και `int max(double,int)`. Όπως είναι σαφές η επίλυση της κλήσης δεν είναι δυνατή με βάση την τιμή επιστροφής.

15 Υπερφόρτωση τελεστών

Στην ενότητα αυτή θα συζητήσουμε για την υπερφόρτωση τελεστών (operator overloading). Όπως έχουμε επισημάνει, οι τελεστές στην C++ αποτελούν συναρτήσεις με παραμέτρους και τιμή επιστροφής που υποστηρίζονται από ειδικό συντακτικό, συνήθως βασιζόμενο σε μη αλφαβητικούς χαρακτήρες. Επομένως, οι τελεστές μπορούν να υπερφορτωθούν όπως και οι τυπικές συναρτήσεις. Η συνηθέστερη αιτία υπερφόρτωσης ενός τελεστή είναι η επέκταση της λειτουργίας του σε μη ενσωματωμένους τύπους, δηλ. στους οριζόμενους από τον χρήστη τύπους, π.χ. struct, union, enumerated.

Δείτε την πλήρη λίστα των τελεστών της C++ στην Wikipedia.

Οι τελεστές μπορούν να υπερφορτωθούν ως καθολικοί ή ως μέλη σύνθετων τύπων. Οι τελεστές μέλη σύνθετων τύπων συζητούνται στον Αντικειμενοστραφή προγραμματισμό. Στην συνέχεια δίνουμε παραδείγματα υπερφόρτωσης καθολικών τελεστών.

Έστω η δομή

```
struct weight {
    double kgrs;
    double grs ;
};
```

αναπαριστά βάρος σε κιλά και γραμμάρια. Καθώς είναι λογικό να θέλουμε να προσθέτουμε μεταβλητές τύπου weight, θα πρέπει να υπερφορτώσουμε την πρόσθεση ώστε να δουλεύει σωστά με τέτοιου τύπου μεταβλητές. Οι τελεστές θα πρέπει να υπερφορτώνονται κατά τέτοιον τρόπο που να διατηρείται ομοιομορφία στην λειτουργικότητά τους μεταξύ διαφορετικών τύπων, ενσωματωμένων ή μη. Έτσι, θα διατηρούμε στον βαθμό που είναι εφικτό την υπογραφή του τελεστή που υπερφορτώνουμε. Η υπογραφή του τελεστή της πρόσθεσης είναι

```
T operator +(const T& a, const T& b);
```

Επομένως, ο τελεστής για την δομή weight, θα υπερφορτωθεί ως εξής:

```
weight operator+(const weight & w1 , const weight & w2) {
    weight rVal;
    rVal. kgrs= w1 . kgrs+ w2. kgrs;
    rVal. grs = w1 . grs + w2. grs ;
    if ( rVal. grs >=1000) {
      rVal. grs -=1000;
      rVal. kgrs++;
    }
    return rVal;
}
```

Αντίστοιχα, οι τελεστές προσαύξησης υπερφορτώνονται ως εξής:

```
weight & operator++(weight & w ) {
    weight l={ 0 , 1};
    w = w + l;
    return w ;
}

weight operator++(weight & w , int ) {
    weight l={ 0 , 1}, rVal= w ;
    w = w + l;
    return rVal;
}
```

Επίσης, είναι λογικό να θέλουμε να στέλνουμε μεταβλητές τύπου weight σε ρεύματα εξόδου· επομένως, θα πρέπει να υπερφορτώσουμε κατάλληλα τον τελεστή διολίσθησης.

```
ostream& operator<<(ostream& os , const weight & w ) {
    if (w.kgrs> 0 ) os << w.kgrs << " Kgrs ";
    if (w.grs > 0 ) os << w.grs << " grs " ;
    return os ;
}
```

Οι περισσότεροι τελεστές μπορούν να υπερφορτωθούν ως καθολικοί τελεστές. Ωστόσο, σημαντικοί τελεστές δεν μπορούν να υπερφορτωθούν παρά μόνον ως μέλη της δομής, π.χ. τελεστής εκχώρησης, “=”, τελεστής θέσης πίνακα (subscript operator), “[]”.

16 Εξορισμού τιμές παραμέτρων (default paramaters)

Στην λίστα τυπικών παραμέτρων μιας συνάρτησης μπορεί να δηλωθούν εξορισμού τιμές. Οι εξορισμού τιμές χρησιμοποιούνται σε μια κλήση της συνάρτησης εφόσον αυτή δεν τροφοδοτηθεί με πραγματικές παραμέτρους.

Παράδειγμα

```
#include <iostream>
using namespace std ;

void dA ( int p1 = 1 , int p2 = 2 , int p3 = 3 ) {
    cout << "p1 value is " << p1 << endl ;
    cout << "p2 value is " << p2 << endl ;
    cout << "p3 value is " << p3 << endl ;
    cout << endl ;
}


int main () {
    dA ( 10 , 20 , 30 );
    dA ( 10 , 20 );
    dA ( 10 );
    dA ();
    return 0 ;
}
```

Στο παραπάνω παράδειγμα, στην πρώτη κλήση της dA (default Arguments) για κάθε τυπική παράμετρο έχει δοθεί αντίστοιχη πραγματική, στην δεύτερη κλήση για την p3 χρησιμοποιείται η εξορισμού τιμή, δηλ. το 3, στην τρίτη κλήση χρησιμοποιείται η εξορισμού τιμή για την p2 και p3, δηλ. οι τιμές 2 και 3, αντίστοιχα· και τέλος στην τελευταία κλήση χρησιμοποιούνται οι εξορισμού τιμές για όλες τις παραμέτρους.

Σε περίπτωση που διαφοροποιείται η δήλωση από τον ορισμό μιας συνάρτησης τότε οι εξορισμού τιμές δίνονται μόνον στην δήλωση, π.χ.

```
#include <iostream>
using namespace std ;

void dA (int p1 = 1 , int p2 = 2 , int p3 = 3);

int main () {
      dA ( 10 , 20 , 30 );
      dA ( 10 , 20 );
      dA ( 10 );
      dA ();
      return 0 ;
}

void dA (int p1, int p2, int p3) {
      cout << "p1 value is " << p1 << endl ;
      cout << "p2 value is " << p2 << endl ;
      cout << "p3 value is " << p3 << endl ;
      cout << endl ;
```

```
}
```

Επίσης, είναι προφανές ότι στην περίπτωση διαδοχικών παραμέτρων ίδιου τύπου θα πρέπει εξορισμού τιμή να δίνεται και στις δύο ή σε καμία ή μόνο στην δεύτερη, δηλ. δεν μπορεί να δίνεται εξορισμού τιμή μόνο στην πρώτη παράμετρο, π.χ.

```
void dA1(int p1, int p2=2); //ok
void dA2(int p1=1, int p2); //compile error
```

17 Πρότυπα[13] συναρτήσεων (function templates)

Σε κάποιες περιπτώσεις προκύπτει η ανάγκη μια συνάρτηση να υπερφορτωθεί κατ' επανάληψη. Ένα χαρακτηριστικό τέτοιο παράδειγμα είναι η συνάρτηση max που περιγράφηκε στην ενότητα 14 για την υπερφόρτωση συναρτήσεων. Προσέχοντας τους σχετικούς ορισμούς της max διαπιστώνει κανείς ότι είναι ίδιοι, δηλ. οι διάφορες υλοποιήσεις της max διαφέρουν μόνον ως προς τους τύπους στους οποίους επενεργούν. Παρόλα αυτά, ο προγραμματιστής εξαναγκάστηκε να γράψει πολλές φορές τον ίδιο κώδικα. Ακριβώς, εδώ η C++ παρέχει τα πρότυπα συναρτήσεων τα οποία αυτοματοποιούν την παραγωγή συναρτήσεων που έχουν την ίδια λειτουργικότητα ενώ επενεργούν σε διαφορετικά δεδομένα.

Παράδειγμα

```
#include <iostream>
using namespace std ;

template < class T > T max( T a , T b ) {
    if ( a > b ) return a ;
    return b ;
}

int main () {
    cout << max<int>(3,5) << endl
         << max<double>(3.5,6.7) << endl;
    return 0 ;
}
```

Στο παραπάνω παράδειγμα, ορίζεται το πρότυπο (**template**) συνάρτησης

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

Το εν λόγω πρότυπο διαθέτει έναν απροσδιόριστο τύπο που ονομάζουμε T (**<class** T>). Σε αυτό το πλαίσιο η τιμή επιστροφής της max είναι T· επίσης, T είναι οι τύποι των παραμέτρων της max. Κατά την κλήση της max, ο χρήστης δίνει τον πραγματικό τύπο, π.χ. max**<int>**. Στην περίπτωση αυτή ο μεταγλωττιστής αυτομάτως παράγει μια συνάρτηση αφαιρώντας τα χαρακτηριστικά του προτύπου (**template <class** T>) και αντικαθιστώντας όπου T με **int**, παράγεται δηλ. η ακόλουθη συνάρτηση

[13] Από άλλους συγγραφείς ο όρος function template αποδίδεται ως περίγραμμα συνάρτησης

```
int max( int a , int b ) {
    if ( a > b ) return a ;
    return b ;
}
```

Αντίστοιχα, για την κλήση `max<double>`, το T αντικαθίσταται με το **double**. Για τις συγκεκριμένες κλήσεις που ο απροσδιόριστος τύπος χρησιμοποιείται στην λίστα των παραμέτρων δεν είναι υποχρεωτικό αυτός να ορισθεί σαφώς· αντίθετα, συνάγεται από τις πραγματικές παραμέτρους. Έτσι, στο παραπάνω παράδειγμα, οι κλήσεις `max<int>(3,5)` και `max<double>(3.5,6.7)` μπορούν να αντικαθιστούν από τις κλήσεις `max(3,5)` και `max(3.5,6.7)`, αντίστοιχα.

Τα πρότυπα συναρτήσεων μπορούν να παραμετροποιηθούν ως προς περισσότερους από έναν τύπους.

Παράδειγμα

```
#include <iostream>
using namespace std ;


template < class T , class U > T max ( T a , U b ) {
    if ( a > b ) return a ;
    return static_cast < T >( b );
}

int main () {
    cout << max(3,5) << endl
    << max(3.5,6.7) << endl
    << max(3,6.7);
    return 0 ;
}
```

18 Τύποι οριζόμενοι από τον χρήστη (User defined types)

Πολύ συχνά στα προγράμματα μας προκύπτει η ανάγκη ορισμού νέων τύπων (διαφορετικών από τους ενσωματωμένους). Για την δημιουργία οριζόμενων από τον χρήστη τύπων παρέχεται μια σειρά από δεσμευμένες λέξεις η κάθε μια από τις οποίες αντιστοιχεί σε διαφορετική ανάγκη:

18.1 Δομή (Structure)

Με την δεσμευμένη λέξη **struct** ορίζουμε έναν σύνθετο τύπο, π.χ.

```
struct name {
    char fName[31];
    char sName[31];
};
```

```
struct sStudent {
    name n;
    char am[5];
    double mo ;
};
```

Τα αναγνωριστικά fName και sName δηλώνουν τα **πεδία (fields)** ή **μέλη (members)** της δομής name. Αντίστοιχα, τα αναγνωριστικά n, am και mo δηλώνουν τα μέλη της δομής sStudent. Έχοντας ορίσει την δομή, μπορούμε να δηλώσουμε μεταβλητές με ανάλογο τρόπο που δηλώνουμε μεταβλητές ενσωματωμένων τύπων, π.χ.

```
sStudent st;
```

Στον ορισμό της, η δομή μπορεί να αρχικοποιηθεί σε μια γραμμή ως ακολούθως:

```
sStudent John = {"John", "Niro", "871", 7.5};
```

Στην συνέχεια, τα μέλη της st προσπελαύνονται με χρήση του τελεστή "." (dot operator), π.χ. st.fName, John.n.fName;

Γενικότερα, χρησιμοποιούμε τις δομές κατά τρόπο ανάλογο με τους ενσωματωμένους τύπους, π.χ.

```
sStudent createStudent(char f[31], char s[31], char am[5],
double mo) {
  sStudent st ;
  strcpy (st.fName, f);
  strcpy (st.sName, s);
  strcpy (st.am, am);
  st.mo = mo ;
  return st ;
}

void assignStudent (sStudent& target, const sStudent& source)
{
  strcpy (target.fName, source.fName);
  strcpy (target.sName, source.sName);
  strcpy (target.am, source.am);
  target.mo = source.mo ;
}

void prtStudent (sStudent s) {
  cout << s.fName << ' ' << s.sName << ' ' << s.am << ' ' <<
  s.mo
  << endl ;
}

void tstStruct () {
  sStudent John = {"John", "Niro", "871", 7.5};
  prtStudent (John);
  sStudent * Mary = new sStudent ;
```

```
  assignStudent (*Mary, createStudent ("Mary", "Nickolson",
  "343", 7.8));
  prtStudent (* Mary);
  delete Mary ;
}
```

Ας σημειωθεί ότι μια δομή μπορεί να περιέχει μεταβλητές οι οποίες είναι επίσης δομές (**nested structure**).

Σχόλιο: Παραδοσιακά, η δομή χρησιμοποιείται για την δημιουργία συνόλων πληροφοριών που αφορούν μια οντότητα, π.χ. η δομή sStudent περιλαμβάνει πληροφορίες που αφορούν ένα μαθητή. Στην C++, η δομή μπορεί να είναι πολύ πιο σύνθετη· ανάλογη με την έννοια της κλάσης για την οποία θα μιλήσουμε στον Αντικειμενοστραφή Προγραμματισμό.

18.2 Ψευδώνυμο τύπου

Με την δεσμευμένη λέξη **typedef** μπορούμε να δημιουργήσουμε ένα ψευδώνυμο (**alias**) για ένα υφιστάμενο τύπο δεδομένων, π.χ.

```
Typedef int integer;
```

```
struct date {
      int day ;
      int month ;
      int year ;
};
```

```
typedef date DATE;
```

```
Στο εξής οι δηλώσεις date και DATE είναι ισοδύναμες.
```

```
typedef date year[366];
```

Στο παράδειγμα αυτό έναν πίνακα 366 ημερομηνιών (date) τον ονομάζουμε έτος (year). Γενικότερα, το συντακτικό της typedef έχει ως εξής: **typedef** existingTypeName newTypeName;

18.3 Απαριθμητοί τύποι

Με την δεσμευμένη λέξη **enum** ορίζεται ένας **απαριθμητός τύπος** (**enumerated type**). Ένας απαριθμητός τύπος ορίζεται ως ένα διατεταγμένο σύνολο από σταθερές, π.χ.

```
enum eColor {red, yellow, green, blue};
```

```
eColor c;
```

Ο τύπος eColor περιλαμβάνει τις σταθερές red=0, yellow=1, green=2, blue=3. Οι εξορισμού τιμές των σταθερών μπορούν να αντικατασταθούν, π.χ.

```
enum eColor {red=10, yellow, green, blue};
```

Τώρα, ο τύπος eColor περιλαμβάνει τις τιμές red=10, yellow=11, green=12, blue=13.

Προσοχή, δεν ισχύει η υπόθεση ότι οι σταθερές ενός απαριθμητού τύπου λαμβάνουν διαδοχικές τιμές, π.χ. ο απαριθμητός τύπος

```
enum eColor {red, yellow=10, green, blue};
```

με τιμές red=0, yellow=10, green=11, blue=12, είναι αποδεκτός. Επομένως, οι απαριθμητοί τύποι θα πρέπει να χρησιμοποιούνται με προσοχή ή να αποφεύγονται ως μεταβλητές ελέγχου επαναληπτικών διαδικασιών.

Έχοντας δηλώσει τον τύπο eColor, μπορούμε να ορίζουμε και να διαχειριζόμαστε μεταβλητές του, π.χ.

```
char* strColor(eColor c) {
    switch (c) {
      case red: return "red";
      case yellow: return "yellow";
      case green: return "green";
    }
    return "blue";
}


void tstENum() {
    cout << strColor(red) << " "
         << strColor(yellow) << " "
         << strColor(green) << " "
         << strColor(blue) << endl;
}
```

18.4 Ένωση (union)

Η δεσμευμένη λέξη **union** μας επιτρέπει να κατανέμουμε στην **ίδια** διεύθυνση μνήμης δύο ή περισσότερες μεταβλητές διαφορετικού τύπου, π.χ.

```
union intToBool {
      int readVal;
      bool ok;
};
```

Το μέγεθος μιας ένωσης ισούται με το μέγεθος του μεγαλύτερου μέλους της. Έτσι, ισχύει **sizeof**(intToBool)==**sizeof**(int).

Ακολουθεί παράδειγμα χρήσης της union intToBool.

Ορίζουμε ένα πίνακα στον οποίο εισάγονται οι απαντήσεις των πελατών.

```
intToBool custAnswer[10];
```

Οι απαντήσεις αυτές έχουν την μορφή int

```
for (int i=0; i<10; i++) custAnswer[i].readVal=i%2;
```

Ωστόσο μπορούμε να τις δούμε και ως bool

```
for (int i=0; i<10; i++)
if (custAnswer[i].ok)
      cout << custAnswer[i].readVal << endl;
```

18.5 Άσκηση

Έστω ο απαριθμητός τύπος

```
enum eMonth { Jan, Feb, Mar, Apr, May, Jun,
              Jul, Aug, Sep,Oct,Nov,Dec
            };
```

και η δομή

```
struct sDate {
    int day;
    eMonth month;
    int year;
    static char sep;
};
```

όπως είναι φανερό η δομή struct αναπαριστά την ημερομηνία. Το πεδίο sep(arator) αναπαριστά τον διαχωριστικό χαρακτήρα. Για παράδειγμα, άν sep==’-‘, η ημερομηνία τυπώνεται ως 10-Mar-2010, αν sep==’/’ τότε η ημερομηνία τυπώνεται ως 10/Mar/2010.

Επίσης, δίνεται η ακόλουθη δήλωση

```
typedef sDate tYear[366];
```

και η συνάρτηση

```
void check() {
    sDate d = date ( 1 , Jan , 1980 );
    tYear y1980;
    for ( int i=0; i<daysOfYear(1980); i++) y1980[i]=d++;
    for ( int i=0; i<daysOfYear(y1980[0].year); i++)
      cout << y1980[i] << endl ;
    cout << endl ;
}
```

Ζητείται να συμπληρωθεί κατάλληλα ο κώδικας ώστε να υποστηρίζεται η εκτέλεση της check.

18.6 Λύση

```
#include <iostream>
using namespace std ;

enum eMonth{Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec};

char
monthLekt[12][4]={"Jan","Feb","Mar","Apr","May","Jun","Jul","A
ug","Sep","Oct","Nov","Dec"};

struct sDate {
  int day ;
  eMonth month ;
  int year ;
  static char sep ;
};

char sDate :: sep = '-' ;
typedef sDate tYear [ 366 ];

sDate date ( int d , eMonth m , int y ) {
  sDate rVal ;
  rVal . day = d ;
  rVal . month = m ;
  rVal . year = y ;
  return rVal ;
}

ostream& operator<< ( ostream& os, const sDate & d ) {
  return os<< d . day << sDate :: sep << monthLekt [ d . month
  ]<<sDate :: sep << d
. year ;
}

bool isLeap( int year ) {
  return year %400==0 || ( year %100 != 0 && year %4 == 0);
}

int daysOfMonth( int year , eMonth month ) {
  if ( month ==Apr || month ==Jun || month ==Sep || month
  ==Nov ) return 30;
  else if ( month ==Feb )
  if ( isLeap( year )) return 29;
  else return 28;
  return 31;
}

sDate & operator++(sDate & d ) {
  if ( daysOfMonth( d . year , d . month )==d . day )
  if ( d . month ==Dec ) d =(sDate ) {1, Jan , d . year +1};
  else d =(sDate ) {1, eMonth ((int ) d . month +1),d . year};
  else d . day ++;
  return d ;
}
```

```
sDate operator++(sDate & d , int ) {
  sDate l= d ;
  ++d ;
  return l;
}

int daysOfYear( int year ) {
  if ( isLeap( year )) return 366 ;
  return 365;
}

void check() {
  sDate d = date ( 1, Jan , 1980);
  tYear y1980;
  for ( int i= 0; i<daysOfYear( 1980); i++) y1980[ i]= d ++;
  for ( int i= 0; i<daysOfYear( y1980[ 0].year ); i++) cout <<
  y1980[ i] << endl;
  cout << endl;
}
```

19 Δυαδικά (μη μορφοποιημένα) αρχεία

Όπως είναι γνωστό τα text αρχεία περιλαμβάνουν δεδομένα που είναι κωδικοποιημένα με βάση το ascii (ή unicode) σύστημα. Αντίθετα, τα δυαδικά αρχεία (binary files) όπως φανερώνει η ονομασία τους, συνιστούν σειρές από bitmaps. Στην ενότητα αυτή παρουσιάζεται η βασική διαχείριση των δυαδικών αρχείων. Για την διαχείριση των δυαδικών αρχείων χρησιμοποιούνται τα “ρεύματα” ofstream, ifstream και fstream που ορίζονται στην βιβλιοθήκη fstream.

Συνήθως, κατά την εργασία με “ρεύματα”, η πρώτη ενέργεια είναι ο συσχετισμός του ρεύματος με το φυσικό αρχείο. Ο συσχετισμός αυτός επιτυγχάνεται με την open. Η open λαμβάνει 2 παραμέτρους: open(fileName, mode). Η πρώτη μεταφέρει την όνομα του αρχείου ενώ η δεύτερη σχετίζεται με το τι θέλουμε να κάνουμε στο αρχείο αυτό. Πιο συγκεκριμένα, η δεύτερη παράμετρος λαμβάνει τιμές από τον ακόλουθο πίνακα:

ios::in	Άνοιγμα αρχείου για είσοδο.
ios::out	Άνοιγμα αρχείου για έξοδο.
ios::binary	Άνοιγμα δυαδικού αρχείου
ios::ate	Αρχική θέση εγγραφής-ανάγνωσης στο τέλος του αρχείου. Αν δεν χρησιμοποιηθεί αυτός ο μετατροπέας τότε η αρχική

	θέση τοποθετείται στην αρχή του αρχείου
ios::app	Όλα τα δεδομένα που στέλνονται για εγγραφή προστίθεται στο τέλος του αρχείου
ios::trunc	Εάν το αρχείο που ανοίγετε για εγγραφή υπάρχει ήδη, τότε διαγράφεται.

Οι τιμές της παραμέτρου mode μπορεί να συνδυάζονται με χρήση του τελεστή "|" (bitwise or). Στην συνέχεια περιγράφονται βασικές λειτουργίες των δυαδικών αρχείων με την βοήθεια παραδειγματικού κώδικα.

Δημιουργία και γράψιμο

```
const int  fileSize = 20;

ofstream oF;
oF. open ( "..\longs.bin" , ios :: binary | ios :: out );
if  (! oF. is_open ()) return;
for  ( int  i = 1; i <= fileSize ; i ++) {
        long  wl =-i ; //implicit typecasting
        oF. write ( reinterpret_cast  < char  *>(& wl ), sizeof  ( wl ));
}
oF. close ();
```

Η κλάση ofstream χρησιμοποιείται για το γράψιμο ενός αρχείου (binary ή text). Εάν στην μέθοδο open δεν δοθεί παράμετρος που να καθορίζει ότι το αρχείο είναι binary τότε ανοίγει ως text αρχείο. Στο παραπάνω παράδειγμα χρησιμοποιείται ο τροποποιητής (modifier) ios::binary|ios::out. Έτσι, το αρχείο ανοίγει ως δυαδικό και προετοιμάζεται για εγγραφή. Στο αρχείο αυτό γράφουμε χρησιμοποιώντας την μέθοδο write που έχει το ακόλουθο interface.

```
ostream& write ( const char* s , streamsize n );
```

Η μέθοδος αυτή λαμβάνει δύο παραμέτρους, ένα buffer s, δηλ. μια περιοχή μνήμης και έναν (unsigned) integral (long ή int) n που δείχνει το μέγεθος (σε chars) του buffer. Όπως φαίνεται στο interface της μεθόδου ο buffer δίνεται ως char*. Επομένως, αν θέλουμε να γράψουμε doubles (όπως στο παράδειγμά μας), τότε πρέπει να εφαρμόσουμε reinterpret_cast.

Σειριακή ανάγνωση δυαδικού αρχείου

```
ifstream iF ;
long  wl ;
iF . open ( "longs.bin" , ios :: binary | ios :: in );
if  (! iF . is_open ()) return  ;
bool  eof = iF . eof ();

while  (! eof ) {
        iF . read ( reinterpret_cast  < char  *>(& wl ), sizeof  ( long  ));
        eof = iF . eof ();
        if  (! eof ) cout << wl << "/" ;
}

iF . close ();
```

Στην περίπτωση αυτή χρησιμοποιούμε την κλάση ifstream και τον modifier ios::binary|ios::in. Αντίστοιχη με την μέθοδο write είναι η μέθοδος read. Καθώς, γνωρίζουμε ότι στο αρχείο είναι αποθηκευμένοι longs, διαβάζουμε τα δεδομένα του αρχείου στην long μεταβλητή wl. Μετά από κάθε διάβασμα ελέγχεται εάν φθάσαμε ή όχι στο τέλος του αρχείου.

19.1 Τυχαία προσπέλαση (Random access)

Όταν έχουμε ένα σύνολο από πληροφορίες και ο χρόνος που απαιτείται για την προσπέλαση οποιασδήποτε από τις πληροφορίες του συνόλου είναι σταθερός, τότε λέμε πως έχουμε τυχαία προσπέλαση. Χαρακτηριστική δομή τυχαίας προσπέλασης είναι οι πίνακες. Η δυνατότητα τυχαίας προσπέλασης παρέχεται και στα δυαδικά αρχεία.

Για να ανοίξουμε ένα αρχείο ταυτοχρόνως για διάβασμα και γράψιμο, χρησιμοποιούμε την κλάση fstream. Η μέθοδος seekp(ut) τοποθετεί τον δείκτη εγγραφής στο byte που δηλώνει η παράμετρός της. Ετσι, η επόμενη εγγραφή (write) γίνεται στην θέση που έχει ορισθεί από την seekp. Αντίστοιχα, η μέθοδος seekg(et) τοποθετεί τον δείκτη ανάγνωσης στην θέση που ορίζει η παράμετρός της.

Ανάγνωση και εγγραφή με τυχαία προσπέλαση

```
fstream ioF ;
ioF . open ( "longs.bin" , ios :: binary | ios :: in | ios :: out );
if  (! ioF . is_open ()) return  ;
for  ( int  i = 10 ; i <= fileSize ; i ++) {
        wl = i + 100 ;
        ioF . seekp ( i * sizeof  ( long  ));
        ioF . write ( reinterpret_cast  < char  *>(& wl ), sizeof  ( long  ));
}

ioF . seekg ( 0 );
do  {
```

```
        ioF . read ( reinterpret_cast < char *>(& wl ), sizeof ( long ));
        eof = ioF . eof ();
        if (! eof ) cout << wl << "/" ;
} while (! eof );

ioF . close ();
```

20 Θεσιακά αριθμητικά συστήματα

Στον χώρο της πληροφορικής συχνά μας ενδιαφέρουν τα αριθμητικά συστήματα με βάση το 2 (δυαδικό), το 8 (οκταδικό) και το 16 (δεκαεξαδικό). Για να μετατρέψουμε στο δεκαδικό σύστημα τον αριθμό N που εκφράζεται στο σύστημα με βάση το B, χρησιμοποιούμε τον ακόλουθο τύπο.

$$N_B = \psi_{L-1} * B^{L-1} + \psi_{L-2} * B^{L-2} + \ldots + \psi_0 * B^0$$

Παραδείγματα

$345_{10} = 3 * 10^2 + 4 * 10^1 + 5 * 10^0$

$01001101_2 =$

$0 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 =$

$2^6 + 2^3 + 2^2 + 2^0 = 64 + 8 + 4 + 1 = 77_{10}$

$345_8 = 3 * 8^2 + 4 * 8^1 + 5 * 8^0 = 192 + 32 + 5 = 229_{10}$

$345_{16} = 3*16^2+4*16^1+5*16^0 = 768+64+5 = 837_{10}$

20.1 Μετατροπές στο Δυαδικό σύστημα

Ο ακόλουθος πίνακας είναι βοηθητικός και δίνει την αναπαράσταση στο δυαδικό σύστημα των δεκαεξαδικών ψηφίων 0..F

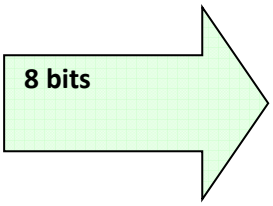
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Πίνακας 1.

20.2 Μετατροπή από το δεκαδικό στο δυαδικό

$28_{10} : 2 = 14 \blacktriangleright 0$

$14_{10} : 2 = 7 \blacktriangleright 0$



7_{10} : 2 = 3 ▶1

3_{10} : 2 = 1 ▶1 00011100_2

1_{10} : 2 = 0 ▶1

20.3 Μετατροπή από το Οκταδικό στο δυαδικό

Για την μετατροπή από το οκταδικό στο δυαδικό μετατρέπεται κάθε ένα ψηφίο σε δυαδική μορφή μήκους 3. π.χ.

$27_8 = 010\ 111_2$

$0345 = 011\ 100\ 101$

Σημείωση 2 : Μία ακέραια σταθερά που αρχίζει από το 0 θεωρείται από τον μεταγλωττιστή αριθμός εκφρασμένος στο οκταδικό σύστημα π.χ $0345==229_{10}$.

20.4 Μετατροπή από το δεκαεξαδικό στο δυαδικό

Για την μετατροπή από το δεκαεξαδικό στο δυαδικό μετατρέπεται κάθε ένα ψηφίο σε δυαδική μορφή μήκους 4. π.χ.

$27_{16} = 0010\ 0111_2$

$0x345 = 0011\ 0100\ 0101$

Σημείωση 3 : Μία ακέραια σταθερά που αρχίζει από το 0x θεωρείται από τον μεταγλωττιστή αριθμός εκφρασμένος στο δεκαεξαδικό σύστημα $0x345==837_{10}$.

20.5 Αναπαράσταση μη προσημασμένων ακεραίων (Unsigned integer types)

Λέξη μήκους 8 bits

►2^8 (256) διαφορετικοί συνδυασμοί

►0..255

20.6 Αναπαράσταση προσημασμένων ακεραίων (Signed integer types)

Λέξη μήκους 8 bits

Θετικοί **Πρόσημο**(1 bit) **και μέτρο**(7 bits)

Ελάχιστος 0 0000000 = 0

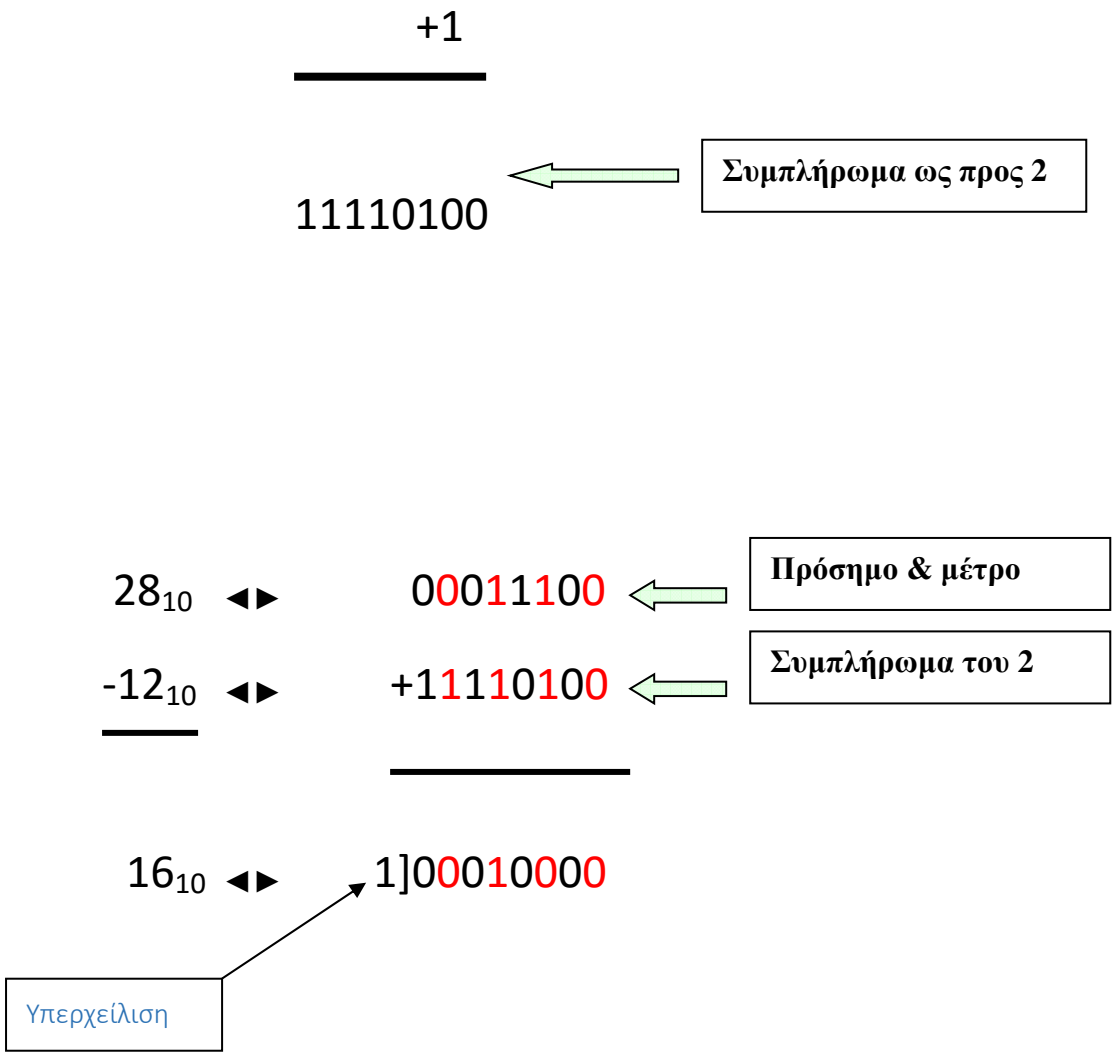
Μέγιστος 0 1111111 = 127_{10}

Αρνητικοί Παράσταση **συμπληρώματος ως προς 2** (**two's complement**)

-12_{10} ►12► 00001100

▼

11110011 ⟸ **Συμπλήρωμα ως προς 1**



Στην συνέχεια παρουσιάζεται η bitMapLib.h. Η συνάρτηση toString είναι βοηθητική και χρησιμοποιείται για την μετατροπή ακεραίου σε string, π.χ. μετατρέπει το 149 σε "149". Οι σπουδαστές οφείλουν να γνωρίζουν **μόνον την χρήση της**. Οι υπόλοιπες συναρτήσεις είναι συναρτήσεις διαχείρισης δεδομένων σε επίπεδο bit. Πιο συγκεκριμμένα, η setBit κάνει 1 το bit του ακεραίου T που βρίσκεται στην θέση p. Η resetBit κάνει 0 το αντίστοιχο bit. Η getBit το επιστρέφει και η reverseBit το αντιστρέφει.

Στην συνέχεια εξηγούμε αναλυτικά την setBit. Αντίστοιχη είναι η ερμηνεία των resetBit και getBit ενώ η ερμηνεία της reverseBit είναι προφανής.

Εστω η κλίση setBit(t,3) όπου t=82 δηλ. 01010010 σε λέξη μήκους 8 bits. Η κλίση setBit(t,3) θα έχει ως αποτέλεσμα η t να γίνει 01011010. Ας σημειωθεί ότι η θέσεις των bits μέσα στον ακέραιο αριθμούνται από δεξιά προς αριστερά ξεκινώντας από το 0, δηλ.

Θέσεις	7 6 5 4 3 2 1 0
t	0 1 0 1 0 0 1 0

Ας δούμε τώρα πως λειτουργεί εσωτερικά η setBit. Στην πρώτη γραμμή ορίζεται η local ίση με 1. Επομένως η εσωτερική αναπαράσταση της local είναι 00000001. Στην συνέχεια γίνεται shift στην local κατά 3 θέσεις. Αυτό έχει ως αποτέλεσμα, τα 3 ψηφία που βρίσκονται

αριστερά στην ψηφιολέξη (bitmap) της local να “χαθούν” και να προστεθούν 3 μηδέν από δεξιά της local. Επομένως η local θα τροποποιηθεί ως εξής: 00001000. Στην συνέχεια γίνεται bitwise or (|), δηλ. or bit by bit, μεταξύ της local και της t. Προσέξτε πως η local έχει διαμορφωθεί έτσι ώστε όλα τα bit είναι 0 εκτός από αυτό που βρίσκεται στην θέση 3. Επομένως, το bitwise or με το t θα έχει ως αποτέλεσμα μια ψηφιολέξη που όλα τα bit θα παραμείνουν ίδια με τα αντίστοιχα bit του t εκτός από το bit στην θέση 3 που θα γίνει 1.

```
local   0 0 0 0 1 0 0 0
t       0 1 0 1 0 0 1 0
---------------------------------
result  0 1 0 1 1 0 1 0
```

```
#ifndef _BITMAPLIB_H
#define _BITMAPLIB_H

#include <string>
#include <sstream>
#include <iostream>
//using namespace std;

template <class T> std::string toString(T t) {
  std::ostringstream oss;
  oss << std::dec << t;
  return oss.str();
}

template <class T> void setBit(T& v, int p) {
 T local=1;
 local<<=p;
 v=v | local;
}

template <class T> void resetBit(T& v, int p) {
 T local=0;
 setBit(local,p);
 local=~local;
 v=v & local;
}

template <class T> int getBit(T v, int p) {
 T local=0;
 setBit(local,p);
 if (local & v) return 1;
 else return 0;
}

template <class T> void reverseBit(T v, int p) {
 if (getBit(v,p)==1) resetBit(v,p);
 else setBit(v,p);
}
```

```
#endif
```

Βιβλιογραφία:

[1] Current C standard PDF (3.61 MB) (as of 2009). In particular, see section 6.2.4 “Storage durations of objects”.

[2] The C++ Programming Language (special edition) by Bjarne Stroustrup (Addison Wesley, 2000; ISBN 0-201-70073-5)