ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημιουργία και χειρισμός LIFO λιστών μεταβλητού μήκους με στοιχεία ακεραίους αριθμούς.

Γενίκευση για χειρισμό λιστών πραγματικών αριθμών και χαρακτήρων με χρήση template.

Καλαμαράς Δημήτρης
Α.Μ. 96055

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
ΟΜΗΡΟΣ ΡΑΓΓΟΣ

2001

ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή

Η εργασία αυτή έχει σαν σκοπό τον λεπτομερή σχεδιασμό και την υλοποίηση στην γλώσσα προγραμματισμού C++, μιας κλάσης, της οποίας τα αντικείμενα θα είναι Last-In-First-Out λίστες μεταβλητού μήκους με στοιχεία ακέραιους αριθμούς και την γενίκευση, με χρήση template, για χειρισμό LIFO λιστών πραγματικών αριθμών ή χαρακτήρων.

Στο πρώτο κεφάλαιο της εργασίας αυτής γίνεται κατ' αρχήν μια σύντομη εισαγωγή στην θεωρία των κλάσεων της C++. Στη συνέχεια, στο δεύτερο κεφάλαιο, θα σχεδιάσουμε και θα υλοποιήσουμε στη γλώσσα αυτή μια κλάση που μπορεί να χρησιμοποιηθεί για δημιουργία και χειρισμό Last-In-First-Out λιστών ακεραίων αριθμών με μεθόδους όπως πρόσθεση και αφαίρεση στοιχείων, αναζήτηση στοιχείων, σύγκριση και συνένωση λιστών, εκτύπωση στοιχείων κτλ.

Κατόπιν, στο τρίτο κεφάλαιο, θα εισάγουμε μερικές βασικές έννοιες από την θεωρία των προτύπων (templates) της C++. Τέλος θα δημιουργήσουμε ένα πρότυπο-γενίκευση της προηγούμενης κλάσης, το οποίο είναι δυνατό να χρησιμοποιηθεί και για λίστες στοιχείων άλλου τύπου όπως πραγματικών αριθμών και χαρακτήρων.

Η υλοποίηση και ο έλεγχος των αντίστοιχων προγραμμάτων έγινε με την βοήθεια της Borland C++ έκδοση 4.

ΚΕΦΑΛΑΙΟ 1

1.1 Εισαγωγή στη θεωρία των κλάσεων

Σε αυτήν την ενότητα θα περιγραφούν οι ορισμοί κλάσεων και τύπων δεδομένων στην C++ με συνοπτικό τρόπο. Μετά ακολουθεί ένα γενικό παράδειγμα για τον ορισμό μιας κλάσης, το οποίο δημιουργεί λίστες μεταβλητού μήκους. Δεν ενδιαφέρει, ακόμη, η αρχιτεκτονική των λιστών αυτών. Το παράδειγμα αυτό θα αποτελέσει στη συνέχεια το πρότυπο με βάση το οποίο θα υλοποιηθούν οι εφαρμογές που περιλαμβάνει αυτή η εργασία και κρίθηκε σκόπιμο να συμπεριληφθεί για την ευκολότερη κατανόηση αυτών που θα ακολουθήσουν στα επόμενα κεφάλαια.

Οι κλάσεις (**classes**) της C++ είναι επεκτάσεις των δομών (structures) της C. Μια κλάση στην C++ είναι μια δομή, η οποία έχει μέλη δεδομένα αλλά και συναρτήσεις για την επεξεργασία των δεδομένων αυτών. Ένα στιγμιότυπο μιας κλάσης λέγεται αντικείμενο (**object**). Οι κλάσεις χρησιμοποιούνται στην C++ για την δημιουργία νέων τύπων δεδομένων, επιπλέον αυτών που περιλαμβάνονται στην γλώσσα αυτή όπως οι **float**, **integer** κτλ. Ένας τύπος δεδομένων αποτελείται από δύο συνιστώσες:

Α) Μια περιγραφή-ορισμό της δομής και του είδους των τιμών του.

Β) Ένα σύνολο λειτουργιών για τον χειρισμό των αντικειμένων δεδομένων του.

Εκτός από αυτές τις λειτουργίες καμιά άλλη λειτουργία ή τελεστής της γλώσσας δεν μπορεί να χειριστεί τα αντικείμενα αυτής της κλάσης. Για αυτό το λόγο οι λειτουργίες αυτές χαρακτηρίζουν το νέο τύπο δεδομένων, αφού ορίζουν τι μπορεί να συμβεί στα αντικείμενα του.

Ένας ορισμός μιας κλάσης στην C++ αποτελείται από δύο μέρη: την επικεφαλίδα (**Header**) και το σώμα (**Body**). Στην επικεφαλίδα, μετά τη λέξη-κλειδί **class** ακολουθεί το όνομα της κλάσης και πιθανόν οι κλάσεις από τις οποίες η νέα κλάση παράγεται (**base classes**). Στο σώμα, υπάρχουν οι ορισμοί των μελών της κλάσης. Τα μέλη μπορεί να είναι δύο ειδών:

1) Μέλη-δεδομένα (**data members**) της κλάσης που ορίζονται με κλασικού τύπου δηλώσεις και καθορίζουν τα αντικείμενα της κλάσης.
2) Συναρτήσεις-μέλους (**member functions**) για τον χειρισμό των μελών δεδομένων, που καθορίζουν τις λειτουργίες της κλάσης.

Τα μέλη μιας κλάσης μπορεί να χαρακτηριστούν σαν

- Ιδιωτικά (**private**), δηλαδή μόνο οι συναρτήσεις της ίδιας της κλάσης μπορούν να τα χρησιμοποιήσουν.
- Δημόσια (**public**), δηλαδή μπορεί να χρησιμοποιηθούν από κάθε χρήστη της κλάσης και τις συναρτήσεις-μέλους.

- Προστατευμένα (**protected**), που σημαίνει ότι μπορούν να χρησιμοποιηθούν από τις συναρτήσεις της κλάσης αλλά και από τις παραγόμενες από αυτήν κλάσεις.

Εδώ, αξίζει να αναφέρουμε το χαρακτηριστικό των «φίλων» συναρτήσεων στην C++. Μερικές φορές είναι απαραίτητο μια συνάρτηση, που δεν είναι μέλος της κλάσης, να έχει πρόσβαση στα ιδιωτικά μέλη της. Αυτό μπορούμε να το επιτύχουμε καθορίζοντας την συνάρτηση σαν «φίλη» (**friend**) της κλάσης. Δηλαδή, γράφουμε στο σώμα της κλάσης την έκφραση :

friend [void] *function_name* (function arguments);

Έτσι, η συνάρτηση αποκτά πρόσβαση σε όλα τα ιδιωτικά μέλη της κλάσης. Αν και μια δήλωση «φίλης» συνάρτησης βρίσκεται στο σώμα της κλάσης, αυτό δεν σημαίνει ότι η συνάρτηση γίνεται μέλος της. Τέλος, η θέση μιας δήλωσης «φίλης» στο εσωτερικό της κλάσης δεν έχει σημασία είτε είναι στο τμήμα των ιδιωτικών, των δημόσιων είτε στο τμήμα των προστατευμένων μελών.

Συνήθως, ο ορισμός μιας κλάσης περιλαμβάνει μόνο τις επικεφαλίδες (**prototypes**) των συναρτήσεων μελών. Ο πλήρης ορισμός αυτών γίνεται εκτός της κλάσης, συνήθως μετά και το κυρίως πρόγραμμα, με χρήση του τελεστή εμβέλειας (**scope operator**) **::** ως εξής: Μετά το τύπο της συνάρτησης και το όνομα της κλάσης στην οποία ανήκει, ακολουθεί ο τελεστής εμβέλειας και μετά το όνομα της συνάρτησης και οι παράμετροι της. Στη συνέχεια, γράφουμε τον υπόλοιπο κώδικα της συνάρτησης.

Με βάση τα προηγούμενα ένας τυπικό παράδειγμα ορισμού κλάσης είναι:

```
- class list
-  {
-  private:
-       int *List;
-       int *Temp;
-       int Length = 0;
-    public:
-       list ();
-       void insert (int ) ;
-       void delete (int );
-       void print ();
-       int search (int) ;
-       int compare ( list &);
-       void concatenate (list & , list &);
-       void copy (list &);
-       ~ list();
-  };
```

Η κλάση αυτή ορίζει ένα τύπο δεδομένων που ονομάζεται λίστα (list). Μια λίστα είναι ένα διατεταγμένο σύνολο δεδομένων. Στο παράδειγμα αυτό, η λίστα αποτελείται από ακέραιους αριθμούς. Ανάλογα με τον τρόπο εισαγωγής και εξαγωγής-διαγραφής στοιχείων από την λίστα, μπορούμε να έχουμε τις **Last In Last Out (LILO)** λίστες, οι οποίες αναπαριστούν ουρές (**Queues**) και τις **Last In First Out (LIFO)** λίστες, οι οποίες αναπαριστούν στοίβες (**Stacks**). Στις μεν LILO λίστες, η εισαγωγή ενός στοιχείου γίνεται στο τέλος της λίστας ενώ η εξαγωγή ενός στοιχείου γίνεται από την αρχή της. Στις δε LIFO λίστες, η εισαγωγή και η εξαγωγή ενός στοιχείου γίνονται στο τέλος της λίστας. Μια LIFO λίστα παρομοιάζεται συχνά με μια στοίβα πιάτων.

Στην πρώτη γραμμή του παραδείγματος , υπάρχει η επικεφαλίδα της κλάσης με το όνομά της.

Το κυρίως σώμα της κλάσης αποτελείται από τρία ιδιωτικά μέλη δεδομένων και εννιά δημόσιες συναρτήσεις μέλους. Τα ιδιωτικά μέλη **List** και **Temp** ορίζονται ως δείκτες (**pointers**), δηλαδή μεταβλητές που δείχνουν σε μια θέση μνήμης, η οποία θα περιέχει ακεραίους αριθμούς. Οι δείκτες έχουν σαν σκοπό τον δυναμικό ορισμό και καταστροφή δεδομένων. Έτσι με την εντολή **List=new int [Length]** μπορούμε να ορίσουμε ένα διάνυσμα ακεραίων με όνομα List με μήκος που καθορίζεται από το τρίτο μέλος-δεδομένο, την ακέραια μεταβλητή **Length**, η οποία αρχικά έχει τιμή μηδέν. Με την εντολή **delete [] List** διαγράφεται ο δείκτης List.

Από τις οκτώ δημόσιες συναρτήσεις μέλους της κλάσης, η πρώτη και η τελευταία, δηλαδή οι list() και ~list() είναι ειδικές συναρτήσεις που λέγονται **constructor** και **destructor** αντίστοιχα. Είναι πάντοτε συνώνυμες της κλάσης και δεν επιστρέφουν τιμές σε αντίθεση με τις υπόλοιπες συναρτήσεις μιας κλάσης. Η constructor χρησιμοποιείται για την δημιουργία και αρχικοποίηση των αντικειμένων μιας κλάσης κατά τη στιγμή του ορισμού τους. Η destructor χρησιμοποιείται για την καταστροφή των μελών δεδομένων μιας κλάσης, ειδικά όταν αυτά είναι δείκτες, όπως στο παράδειγμα.

Οι υπόλοιπες δημόσιες συναρτήσεις είναι μέθοδοι πρόσθεσης στοιχείων στη λίστα, αφαίρεσης στοιχείου, εκτύπωσης της λίστας, αναζήτησης μιας τιμής, σύγκρισης δυο λιστών, συνένωσης δυο λιστών σε μια τρίτη λίστα και αντιγραφής μιας λίστας σε άλλη.

Στον προηγούμενο ορισμό της κλάσης περιλαμβάνονται μόνο τα πρωτότυπά των συναρτήσεων αυτών. Η λεπτομερής περιγραφή τους γίνεται μετά το τέλος του ορισμού αυτού με τον εξής τρόπο:

```
- void list :: list ()                          /* constructor */
- { List = new int [Length];          /*  Δημιουργία διανυσμάτων  που */
-   Temp = new int [Length];     /* αντιπροσωπεύουν τις λίστες /*
- }
-
-  void list :: insert ( int x)
-  {
-    /* Αλγόριθμος για την εισαγωγή του στοιχειού σε κάποιο σημείο της
-       λίστας π.χ. στην αρχή ή στο τέλος.
-       Διαφέρει ανάλογα με τον τύπο της λίστας.  */
-  }
-
-  void list :: delete (int x)
-   {
-  /* Αλγόριθμος για την διαγραφή στοιχείου από την λίστα, ανάλογα με τον τύπο της λίστας. */ }

- void list :: print ()
-  {    /* Εκτύπωση των στοιχείων της λίστας. */  }
-
- int  list :: search (int x)
-  {  /* Σειριακή αναζήτηση στοιχείου  Επιστρέφει τη θέση του  */  }
-
- int list :: compare ( list &b)
-  {
-   /* Αλγόριθμος για την σύγκριση της τρέχουσας λίστας a με την λίστα b.
-      Η δεύτερη λίστα περνά στην συνάρτηση με κλήση με αναφορά
```

```
-       χρησιμοποιώντας τον τελεστή &. Η συνάρτηση επιστρέφει  1 αν οι
-       λίστες είναι ίδιες ή 0 αν διαφέρουν.  */
-    }
-
-    void list :: concatenate (list &b , list &c )
-     {
-      /* Συνένωση της  τρέχουσας  λίστας  με την b και  αποθήκευση του   αποτελέσματος στην
     λίστα c.  Οι λίστες b και c  περνούν στην συνάρτηση ως παράμετροι με κλήση με αναφορά  */
     }
-
-    void lifo :: copy ( list &b)
-    { /* Αντιγραφή της τρέχουσας λίστας  με την λίστα b, η οποία περνά στην συνάρτηση με
     κλήση με αναφορά */ }

-    list :: ~ list ()
-    {   delete [] List ;  delete [] Temp;  }   /* Καταστροφή διανυσμάτων-λιστών  */
```

Οι περισσότερες συναρτήσεις μέλους καθορίζονται από τον τύπο δεδομένων που θέλουμε να δημιουργήσουμε, αλλά και αντίστροφα.

Πρέπει να κάνουμε μια παρατήρηση σχετικά με τη χρήση κλήσης με αναφορά (**call by reference**) στο πέρασμα των παραμέτρων ορισμένων συναρτήσεων της κλάσης. Στις παραμέτρους συναρτήσεων, όπως μεταβλητές, δείκτες ή αντικείμενα κλάσεων, χρησιμοποιούμε την κλήση με αναφορά, μέσω του τελεστή **&**, όταν θέλουμε οι τιμές των παραμέτρων αυτών να μπορούν να μεταβληθούν από την συνάρτηση και να επιστραφούν στο πρόγραμμα. Τέτοια συνάρτηση εδώ είναι η **copy (list &b)** . Χρησιμοποιούμε τον τελεστή & για την κλήση του αντικείμενου b της κλάσης list με αναφορά. Έτσι εξασφαλίζουμε ότι η **copy** θα μπορεί να μεταβάλλει το περιεχόμενο του σύμφωνα με τον αλγόριθμο της και να το επιστρέψει στο πρόγραμμα.

Στο κυρίως πρόγραμμα, η δημιουργία ενός αντικειμένου της κλάσης list γίνεται με τον ίδιο τρόπο που ορίζουμε μεταβλητές, π.χ. list a ή list a[100], στην περίπτωση που θέλουμε το αντικείμενο να έχει τη μορφή διανύσματος. Επειδή έχουμε χρησιμοποιήσει **constructor** στον ορισμό της κλάσης, η δημιουργία των αντικειμένων γίνεται ταυτόχρονα και με την αρχικοποίηση τους.

Ένα τυπικό πρόγραμμα που χρησιμοποιεί αντικείμενα της κλάσης list είναι το παρακάτω:

```
-    #include <iostream.h>
-
-    void main ()
-     {
-       int x,y,z;     /* Ακέραιοι x,y,z που θα είναι τα στοιχεία των λιστών */
-       list a,b,c;          /* Δημιουργία τριών αντικειμένων a,b,c */
-
-      a.insert (x); a.insert (y); a.insert (z);   /*Εισαγωγή στοιχείων στην a */
-       b.insert (x);
-       a.print (); b.print (); c.print ();          /* Εκτύπωση λιστών */
```

```
-       if (  a.compare ( b) )  cout << "Οι λίστες a,b είναι όμοιες" ;
-       else cout << "Οι λίστες a,b διαφέρουν ";
-       a.concatenate (b,c);
-       if  ( c.compare (a) ) cout << "Οι λίστες a,c είναι όμοιες";
-       else cout << "Οι λίστες a,c διαφέρουν";
-       if (c.search (x) ) cout << "Το στοιχείο"
-                                << x
-                                << " Βρέθηκε στη θέση : " << c.search(x);
-       c.print ();
-       c.delete (x);
-       c.print ();
-    }
-
-     }
-
```

Το τυπικό αυτό πρόγραμμα χρησιμοποιεί επιπλέον την κλάση **iostream** της C++ για την Είσοδο/Έξοδο (Ε/Ε) στοιχείων με τα αντικείμενα **cout** και **cin**.

Ορίζουμε τρία αντικείμενα **a,b,c** της κλάσης list και τρεις ακέραιους **x,y,z** που εισάγουμε στην λίστα a. Επίσης, εισάγουμε το x στην λίστα b. Κατόπιν εκτυπώνουμε και τις τρεις λίστες. Η λίστα a θα πρέπει να έχει τρία στοιχεία, τα x,y,z, η b μόνο το x, ενώ η c κανένα. Στις επόμενες γραμμές, με χρήση της συνάρτησης-μέλους **compare** και της έκφρασης **if-else,** συγκρίνουμε την λίστα a με τη b. Θα πρέπει να είναι διαφορετικές. Κατόπιν συνενώνουμε την a με τη b και την νέα λίστα την αποθηκεύουμε στην c. Συγκρίνοντας τις λίστες a και c, θα είναι διαφορετικές. Μετά, χρησιμοποιούμε την συνάρτηση μέλους **search** για να ελέγξουμε αν το στοιχείο x υπάρχει στη λίστα c. Το πρόγραμμα θα πρέπει να τυπώσει τη θέση του στοιχείου στη λίστα. Κατόπιν, εκτυπώνουμε τη λίστα c και μετά με χρήση της συνάρτησης-μέλους **delete** σβήνουμε το στοιχείο x από τη λίστα c. Τέλος με εκτύπωση της c, επιβεβαιώνουμε την διαγραφή του στοιχείου x από την λίστα.

Το τυπικό αυτό παράδειγμα του ορισμού της κλάσης list, αλλά και του χειρισμού των αντικειμένων της από ένα κυρίως πρόγραμμα αποτελεί ένα πρότυπο για τον ορισμό και τη χρήση μιας κλάσης της οποίας τα αντικείμενα θα είναι LIFO λίστες.

ΚΕΦΑΛΑΙΟ 2

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο, δημιουργείται λεπτομερώς η κλάση lifo, της οποίας τα αντικείμενα είναι LIFO λίστες ακεραίων αριθμών. Χρησιμοποιούνται επιπλέον στοιχεία και συναρτήσεις της C++, προκειμένου το τελικό πρόγραμμα να είναι λειτουργικό και αξιόπιστο. Αυτά είναι:

Η λέξη-κλειδί **enum**, η συνάρτηση **getch()** και η δομημένη εντολή **switch-case**, η οποία χρησιμοποιείται στο κυρίως πρόγραμμα για την εκτύπωση του βασικού μενού.

Η enum χρησιμοποιείται στην C++ για την απαρίθμηση σταθερών με ακέραια τιμή αλλά και για την δημιουργία ολοκληρωμένων τύπων δεδομένων με ακέραιες τιμές. Η σύνταξη της είναι:

- enum (<ετικέτα τύπου>) { <σταθερά> (=<τιμή>), … } (μεταβλητές)

Εάν παραλείπονται οι τιμές των σταθερών, τότε η πρώτη σταθερά παίρνει τιμή 0, η δεύτερη 1, η τρίτη 2 κοκ.

Π.χ. η εντολή:

- enum Bool { False , True }

ορίζει ένα μοναδικό τύπο δεδομένων που λέγεται Bool με σύνολο τιμών το (False = 0 , True = 1). Έτσι μια μεταβλητή ή συνάρτηση τύπου Bool μπορεί να πάρει τιμές False ή True με εσωτερικές τιμές αυτών 0 και 1 αντίστοιχα. Για παράδειγμα στην κλάση lifo θα ορίσουμε μια συνάρτηση μέλους με όνομα member, η οποία θα ελέγχει αν ένα δοσμένο στοιχείο ανήκει στην λίστα ή όχι και θα επιστρέφει True (=1) ή False (=0) , αντίστοιχα.

Η συνάρτηση getch() ανήκει στην κλάση **conio**. Η σύνταξή της είναι

- int getch (void);

Η συνάρτηση αυτή διαβάζει ένα χαρακτήρα από το πληκτρολόγιο χωρίς να τον τυπώνει στην οθόνη. Θα χρησιμοποιηθεί για το προσωρινό σταμάτημα της ροής του προγράμματος μέχρι ο χρήστης να πατήσει κάποιο πλήκτρο.

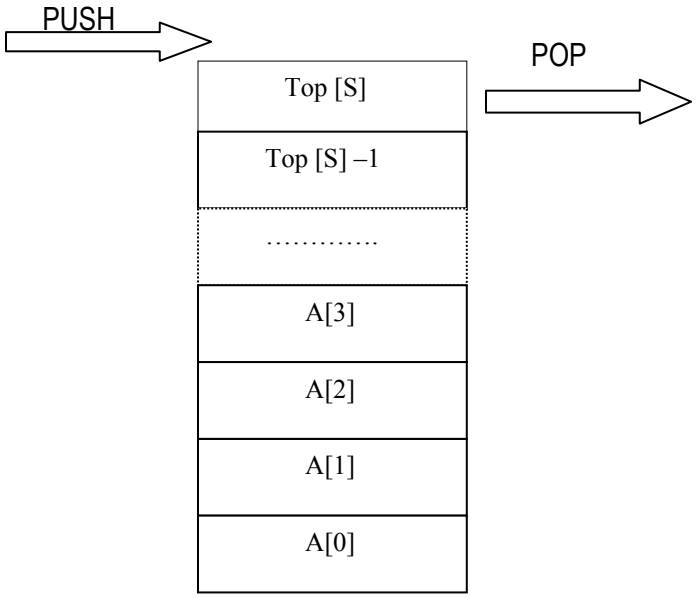
Η δομημένη εντολή switch-case έχει σύνταξη:

```
switch ( <μεταβλητή> )
 {
  case  <τιμή> : <εντολές>; [break;]
    .
    .
    .
  default : <εντολή>;
}
```

Με την έκφραση αυτή, ελέγχεται η μεταβλητή και ανάλογα με την τιμή της, η ροή του προγράμματος μεταφέρεται στο αντίστοιχο case, όπου εκτελούνται οι εντολές. Μετά το τέλος των εντολών πρέπει να υπάρχει η εντολή break για να αποφεύγεται η εκτέλεση των εντολών του επόμενου case. Αν η μεταβλητή έχει τιμή που δεν ταιριάζει με κανένα case, τότε εκτελούνται οι εντολές της default.

2.2 LIFO – ΛΙΣΤΕΣ

Όπως αναφέρθηκε στην εισαγωγή, μια LIFO λίστα είναι ένα διατεταγμένο σύνολο δεδομένων με την ιδιότητα: το τελευταίο στοιχείο που εισάγεται στη λίστα είναι και το πρώτο που εξάγεται από αυτήν. Μια LIFO λίστα καλείται και στοίβα (**stack**). Θεμελιώδεις λειτουργίες σε μια στοίβα είναι η PUSH, η οποία εισάγει ένα νέο στοιχείο στην κορυφή της και η POP, που αφαιρεί ένα στοιχείο από την κορυφή της στοίβας. Το στοιχείο που βρίσκεται στην κορυφή λέγεται Top. Σχηματικά μια στοίβα είναι:



Σχηματική αναπαράσταση Στοίβας

Από τα προηγούμενα είναι προφανές ότι σε μια LIFO λίστα, οι μόνες λειτουργίες εισόδου/εξόδου στοιχείων της λίστας είναι οι PUSH, POP και η TOP, η οποία εμφανίζει το στοιχείο που βρίσκεται στην κορυφή της στοίβας.

Για να κατασκευάσουμε LIFO λίστες θα χρησιμοποιήσουμε ένα δείκτη ***list**, ο οποίος θα δείχνει στη θέση της μνήμης από την οποία αρχίζει η καταγραφή των περιεχομένων κάθε λίστας, και είναι χρήσιμος για την δυναμική διαχείριση λιστών. Στην ουσία η λίστα θα είναι ένα διάνυσμα του οποίου το μήκος θα μπορούμε να μεταβάλλουμε δυναμικά. Το μήκος κάθε λίστας θα δίνεται από την ακέραια μεταβλητή length, η οποία θα πρέπει να αυξάνεται ή να μειώνεται κατά ένα με κάθε εφαρμογή της PUSH ή της POP αντίστοιχα. Οι ***list** και **length** θα είναι δεδομένα-μέλη της **class lifo** ιδιωτικού χαρακτήρα (private), για να μην είναι προσπελάσιμα παρά μόνο από τις συναρτήσεις της κλάσης. Στην αρχή λειτουργίας του προγράμματος όλες οι λίστες θα είναι μηδενικού μήκους, συνεπώς θα είναι length=0. Για αυτό στην constructor συνάρτηση της κλάσης θα πρέπει να υπάρχουν οι εντολές :

- length = 0;
- list = new int [length];

Κατά σειρά εμφάνισης των συναρτήσεων μελών της **class lifo**, αυτές θα είναι:

2.3 MEMBER FUNCTIONS

2.3.1 PUSH

void push (ακέραιος x)
- <έλεγξε το length>
- [Αν length = 0] τότε
- Θέσε length = length +1
- Δημιούργησε το διάνυσμα list με το νέο μήκος length
- Θέσε list [length-1] = x
- [Αν length > 0] τότε
- Δημιούργησε ένα προσωρινό διάνυσμα temp με μήκος length
- Αντέγραψε τα περιεχόμενα του list στο temp
- Δηλαδή temp [I] = list [I] για κάθε I ε [0, length]
- Διέγραψε το *list.
- Θέσε length = length +1
- Δημιούργησε το *list με το νέο μήκος
- Αντέγραψε τα περιεχόμενα του *temp στο *list
- Διέγραψε το *temp
- Θέσε list [length-1]=x

Παρατηρήσεις:

1) Στην C++ , ένα διάνυσμα A μήκους n , αναπαρίσταται ως εξής:

A_0	A_1	A_2	A_3	….	……	…..	A_{n-2}	A_{n-1}

Έτσι, το 1ο στοιχείο του διανύσματος A είναι το A_0, το 2ο είναι το A_1 κοκ. Το n-οστό στοιχείο είναι το A_{n-1}. Γι' αυτό το λόγο στον αλγόριθμο της PUSH, για το στοιχείο x που πρέπει να μπει στο τέλος της list έχουμε : list[length-1]=x. Το ίδιο θα γίνει και στον αλγόριθμο της POP.

2) Επειδή χρειαζόμαστε ένα προσωρινό διάνυσμα temp, για την αντιγραφή των περιεχομένων του *list, τόσο στην λειτουργία PUSH όσο και στην POP, θα χρησιμοποιήσουμε ένα δείκτη *temp, ο οποίος θα είναι δεδομένο-μέλος της class lifo και μάλιστα private αφού δεν υπάρχει λόγος να είναι ορατό εκτός των συναρτήσεων της κλάσης. Γι' αυτό στην constructor συνάρτηση θα προσθέσουμε την εντολή:
   - temp = new int [length]

3) Η λειτουργία PUSH δέχεται ως παράμετρο το στοιχείο που θέλουμε να εισάγουμε και δεν επιστρέφει καμία τιμή.

2.3.2 POP

Ο αλγόριθμος της λειτουργίας POP θα είναι:

int pop ()
- Όρισε μια ακέραια μεταβλητή pop
- < Έλεγξε το length >
- [Αν length = 1] τότε
- Θέσε top = list [length-1]
- Διέγραψε το *list
- Θέσε length = length –1
- Δημιούργησε το *list με νέο μήκος length
- Return top
- [Αν length = 0] τότε
- Εκτύπωσε « Λάθος. UNDERFLOW.»
- Return –1
- [Αν length > 1] τότε
- Διέγραψε το *temp (αν υπάρχει)
- Δημιούργησε το διάνυσμα *temp μήκους length
- Αντέγραψε το *list στο *temp , δηλαδή
- temp [i] = list [i] για κάθε i ε [0, length-1]
- Διέγραψε το *list
- Θέσε length=length-1
- Δημιούργησε εκ νέου το *list με νέο μήκος length
- Αντέγραψε τα περιεχόμενα του *temp στο *list, δηλαδή
- list [i] = temp [i] για κάθε i ε [0, length-1]
- Θέσε top = temp [length]
- return top

Παρατηρήσεις:

1) Η λειτουργία POP επιστρέφει πάντα μια ακέραια τιμή, γι' αυτό η αντίστοιχη συνάρτηση-μέλους θα πρέπει να είναι τύπου **int**. Την ακέραια τιμή προς επιστροφή την αποθηκεύουμε σε μια τοπική μεταβλητή top.
2) Ο αλγόριθμος ελέγχει την τιμή του length της ενεργής λίστας και προβλέπει τρεις περιπτώσεις: length=1 , length=0 , length > 0. Στην πρώτη περίπτωση, η λίστα έχει ένα μόνο στοιχείο, το οποίο η pop επιστρέφει, μηδενίζοντας την τιμή του length. Στην δεύτερη περίπτωση, η λίστα έχει μηδενικό μήκος, συνεπώς δεν υπάρχει στοιχείο για επιστροφή. Η κατάσταση αυτή λέγεται **UNDERFLOW** και τυπώνεται ανάλογο μήνυμα. Στην τρίτη περίπτωση, τα περιεχόμενα της *list αποθηκεύονται στο προσωρινό διάνυσμα του *temp και μειώνεται το μήκος της λίστας κατά 1. Μετά δημιουργείται εκ νέου το *list με το νέο length και τα περιεχόμενα του *temp, εκτός του τελευταίου στοιχείου, αντιγράφονται στο *list. Η μεταβλητή προς επιστροφή παίρνει τιμή top = temp [length-1] .

3) Λόγω των παραπάνω ο δείκτης *temp δεν καταστρέφεται με το τέλος κάθε λειτουργίας POP γι' αυτό πριν χρησιμοποιηθεί από οποιαδήποτε ρουτίνα θα πρέπει να καταστρέφεται και μετά να δημιουργείται ξανά.

2.3.3 PRINT

Η τρίτη συνάρτηση μέλους που θέλουμε να υπάρχει στην κλάση **lifo** είναι η **print**, η οποία θα τυπώνει τα περιεχόμενα της ενεργής λίστας, ένα προς ένα. Ο αλγόριθμός της θα είναι :

void print ()
- Άλλαξε γραμμή στη θέση του δρομέα στην οθόνη
- Τύπωσε το στοιχείο list [i] για κάθε i ε [0 , length-1]

2.3.4 MEMBER

Η συνάρτηση μέλους member δέχεται ως παράμετρο ένα ακέραιο x. Αν και απλούστατη, είναι βασική για τη λειτουργία της κλάσης, καθώς χρησιμοποιείται από άλλες συναρτήσεις-μέλους προκειμένου να ελεγχθεί αν ένας ακέραιος είναι στοιχείο μιας λίστας ή όχι. Η member είναι τύπου Bool, δηλαδή επιστρέφει τιμές True (=1) αν ο x ανήκει στη λίστα και False (=0) στην αντίθετη περίπτωση. Ο αλγόριθμός της είναι:

Bool member (int x)
- Για i=0 έως i<= length-1
- [Αν list[i] = x] return True ;
- [αλλιώς] return False

2.3.5 SEEK

Η επόμενη συνάρτηση της class lifo είναι η seek. Η seek είναι μια ολοκληρωμένη ρουτίνα η οποία ζητά από τον χρήστη ένα στοιχείο και ελέγχει την ύπαρξη και τη θέση του στη τρέχουσα λίστα. Ανάλογα με το αποτέλεσμα τυπώνει αν υπάρχει ή όχι καθώς και τη θέση του στη λίστα. Η seek χρησιμοποιεί την συνάρτηση member για να ελέγξει αν το στοιχείο υπάρχει στη λίστα. Ο αλγόριθμός της θα είναι:

void seek ()
- Όρισε χαρακτήρες x,m
- Είσοδος στοιχείου x από τον χρήστη.
- [Αν όχι member (x)] τότε
- τύπωσε « Δεν βρέθηκε».
- Τύπωσε «Πίεσε οποιοδήποτε πλήκτρο για συνέχεια».
- Θέσε m=getch ()
- [Αν member (x)] τότε
- Για i=0 έως i=length-1
- [Αν list[i] = x] τότε
- τύπωσε «Το στοιχείο βρέθηκε στη θέση» ; τύπωσε (i+1)
- Τύπωσε « Πίεσε οποιοδήποτε πλήκτρο για συνέχεια»
- Θέσε m=getch();

ΠΑΡΑΤΗΡΗΣΗ
Στην seek χρησιμοποιείται και η συνάρτηση getch() για το προσωρινό σταμάτημα της ροής του προγράμματος μετά την εκτύπωση ενός μηνύματος στην οθόνη, μέχρι ο χρήστης να πιέσει κάποιο πλήκτρο.

2.3.6 COPY

Η συνάρτηση μέλος copy αντιγράφει την ενεργή λίστα σε μία άλλη. Η λίστα–στόχος b περνά στην συνάρτηση ως παράμετρος με κλήση με αναφορά προκειμένου να μπορεί η συνάρτηση να αλλάξει τα περιεχόμενα της. Ο αλγόριθμος της είναι:

void copy (lifo & b)
- Δημιούργησε ένα νέο διάνυσμα b.list με μήκος [length]
- Για i =0 έως i <= length-1
- Θέσε b.list[i]=list[i];
- Θέσε b.length=length;

Η copy αντιγράφει ένα προς ένα τα στοιχεία του *list στο νέο διάνυσμα *b.list και, τέλος, θέτει το length της b ίσο με αυτό της πηγαίας λίστας.

2.3.7 COMPARE

Η συνάρτηση compare συγκρίνει την ενεργή λίστα με μια λίστα b, η οποία περνά στην συνάρτηση ως παράμετρος με κλήση με αναφορά. Η compare είναι τύπου Bool. Πρώτα ελέγχει τα μήκη των δύο λιστών. Αν είναι άνισα επιστρέφει False. Αν είναι ίσα, ελέγχει αν τα στοιχεία της τρέχουσας λίστας *list είναι ίσα, ένα προς ένα, με τα αντίστοιχα στοιχεία της λίστας b. Αν οι λίστες διαφέρουν σε κάποιο στοιχείο, τότε και η compare επιστρέφει False. Αλλιώς επιστρέφει True. Ο αλγόριθμός της είναι:

Bool compare (lifo &b)
- [Αν b.length όχι ίσο με length] return False;
- για i = 0 έως i <= length-1
- [Αν b.list[i] όχι ίσο με list[i]] return False ;
- return True;

2.3.8 CONCAT

Η συνάρτηση μέλους concat συνενώνει την τρέχουσα λίστα με μια λίστα b και αποθηκεύει το αποτέλεσμα σε μια τρίτη λίστα c. Οι λίστες b, c περνούν στην συνάρτηση ως παράμετροι με κλήση με αναφορά. Καταρχήν, η concat χρησιμοποιεί την συνάρτηση μέλους copy, για να αντιγράψει την λίστα b στην c. Κατόπιν, κάθε ένα στοιχείο της τρέχουσας λίστας εισάγεται στην c, με διαδοχικά push. Ο αλγόριθμός της είναι:

void concat (lifo &b, lifo &c)
- κάλεσε την συνάρτηση copy, από την λίστα b , στην λίστα c.
- Για i = 0 έως i <= length-1
- Κάλεσε c.push (list[I]) ;

2.3.9 LEN

Η συνάρτηση μέλους len είναι τύπου int και επιστρέφει το μήκος length της τρέχουσας λίστας. Ο αλγόριθμός της είναι:

int len ( )
- return length;

2.3.10 TOP

Η συνάρτηση μέλους top είναι τύπου int και επιστρέφει το στοιχείο που βρίσκεται στην κορυφή της στοίβας. Ο αλγόριθμός της είναι:

int top ()
- return list[length-1];

Με βάση τα προηγούμενα ο ορισμός της κλάσης lifo θα περιέχει δώδεκα συναρτήσεις μέλους, οι οποίες θα είναι public συμπεριλαμβανομένων και των **constructor** και **destructor** συναρτήσεων: **lifo**, **~lifo**. Ακόμη θα υπάρχουν τρία δεδομένα μέλη, οι δείκτες ***list**, ***temp** ακέραιου τύπου και η ακέραια μεταβλητή **length**. Και τα τρία δεδομένα μέλη θα είναι ιδιωτικού χαρακτήρα, έτσι ώστε μόνο οι συναρτήσεις της κλάσης να έχουν πρόσβαση σε αυτά. Οπότε, ο ορισμός της κλάσης θα είναι:

2.4 Ορισμός της κλάσης lifo

```
class lifo
{
 public :
        lifo () ;
        void push (int) ;
        int  pop () ;
        void print () ;
        void seek () ;
        void copy ( lifo& ) ;
        void concat ( lifo&,  lifo& ) ;
        Bool compare ( lifo& ) ;
        Bool member( int ) ;
        int len () ;
        int top () ;
        ~lifo () ;
 private :
        int *list,*temp  ;
```

```
        int length ;
};
```

Στον παραπάνω ορισμό, οι συναρτήσεις μέλους αναφέρονται μόνο με τα πρωτότυπά τους. Ο λεπτομερής ορισμός τους γίνεται μετά και έχει ως εξής:

```
lifo::lifo ()
                {
                        length=0;
                        list=new int [length];
                        temp = new int [length];
                }

Bool lifo :: member ( int x )
        {
                for (register i=0 ; i<=length -1 ; ++i)
                 if ( list[i] == x ) return True;
                 return False ;
        }

void lifo :: push (int x )
        {
                 if ( length == 0 )
                        {
                                delete  [] list;
                                list = new int [++length];
                                list[length-1] = x; }
                 else
                        {
                         delete [] temp;
                         temp = new int [length] ;
                         for (register i=0 ; i <= length-1 ; i++ )
                                {
                                 temp [i] = list [i];
                                }
                          delete [] list;
                          list = new int [++length];
                          for ( register =0 ; i <= (length-2) ; i++ )
                                {
                                        list [i] = temp [i];
                                }
                          list [length-1] = x;
                        }
                }

        int lifo :: pop ()
```

```
{
        int top;
        if  ( length == 1 )
                { top = list [length-1];
                  delete [] list;
                  list = new int [--length];
                  return top ;
                }
        else if ( length == 0 )
                        {
                                cout << " \nError. UNDERFLOW.";
                                return -1;
                         }
        else
            {
             delete [] temp;
             temp = new int [length];
             for ( register i=0 ; i <= length-1 ; i++ )
             {    temp [i] = list [i];       }
             delete [] list;
             list = new int [--length];
             for (  i=0 ; i <= length-1 ; i++ )
             {       list [i] = temp [i];       }
             top = temp [length] ;
             return  top;
            }
}

void lifo :: seek ()
 {
        int x, m;
        cout << "\nEnter element to seek : ";
        cin >> x;
        if (! member (x) )
            {
             cout << "\nSorry. Not Found.\n";
             cout << "\nPress any key.";
             m=getch();
            }
        else
            {
              for ( register i=0 ; i<=length-1 ; i++)
                        {
                         if (list[i] == x )
                                {
                                cout << "\nFound at "
                                        << (i+1)
```

```
                                                << " position of current stack.";
                                        cout << "\nPress any key.";
                                        m=getch();
                                        }
                                }
                        }
}

void lifo::print ( )
        {
                cout << "\n";
                for ( register i = 0 ; i <= length-1 ; i ++ )
                        {
                         cout << "[ " << list [i] << " ]" << "\t";
                        }
                }

void lifo :: copy ( lifo &b  )
        {
                b.list = new int [length];
                for (register i=0 ; i<=length-1 ; i++)
                b.list[i]=list[i];
                b.length=length;
        }


 Bool lifo :: compare (lifo &b)
        {
                if (b.length != length ) return False;
                for (register i=0 ; i<=length-1 ; i++)
                if ( b.list[i] != list[i] ) return False ;
                return True;
        }


void lifo :: concat (lifo &b, lifo &c)
 {
                b.copy (c);
                for ( register i=0 ; i<=length-1 ; ++i)
                                        c.push ( list[i] ) ;
 }

 int lifo :: len ( )
        {               return length;          }


int lifo :: top ()
```

```
 {
                return list[length-1];
        }

lifo :: ~lifo ()
        {
                delete [] list;
                delete [] temp;
        }
```

2.5 MAIN program

Για να γίνει χρήση της κλάσης που δημιουργήθηκε απαιτείται ένα κυρίως πρόγραμμα, το οποίο θα δημιουργεί τα αντικείμενα-λίστες της κλάσης και παράλληλα θα προσφέρει ένα περιβάλλον εργασίας με αυτά. Η δημιουργία των αντικειμένων γίνεται για λόγους ευκολίας αλλά και δυναμικότητας, με την εντολή **lifo a[100]**, μέσω της οποίας δημιουργούνται 100 διαφορετικά αντικείμενα–lifo λίστες, από την a[0] έως την a[99]. Έτσι μπορεί κανείς να δει και την δυναμικότητα των κλάσεων στην C++, ακόμη και με τις απλές τεχνικές προγραμματισμού που χρησιμοποιούνται εδώ, καθώς κάθε μία από αυτές τις λίστες μπορεί να αποθηκεύσει πάνω από 5.000 στοιχεία. Συνολικά, μπορούν να αποθηκευτούν 100 x 5.000 = 500.000 στοιχεία και αφού ένας ακέραιος αποθηκεύεται σε 2 bytes μνήμης, η κλάση ή μάλλον τα αντικείμενα της μπορούν να «χωρέσουν» 2 x 500.000 = 1.000.000 bytes τουλάχιστον , δηλαδή πάνω από 1 Megabyte.

Το κυρίως πρόγραμμα (**main ()**) τώρα το οποίο θα χρησιμοποιεί τις LIFO λίστες της παραπάνω κλάσης και τις συναρτήσεις της, θα πρέπει υποχρεωτικά να εμφανίζει ένα μενού επιλογών στο χρήστη, η κάθε μια από τις οποίες θα αντιστοιχεί σε μια συνάρτηση της κλάσης lifo, π.χ. push, pop, print, copy κοκ. Το μενού είναι απαραίτητο για την αλληλεπίδραση προγράμματος και χρήστη. Για να γίνει αυτό θα χρησιμοποιηθεί ένας άπειρος βρόχος **for (;;)**, στο εσωτερικό του οποίου θα εμφανίζονται με τη βοήθεια των αντικειμένων cout της κλάσης iostream οι επιλογές στην οθόνη και με τη χρήση της δομημένης εντολής **switch () … case … case …** θα γίνονται οι επιλογές των απαραίτητων συναρτήσεων της κλάσης καθώς και των ορισμάτων τους.

Συγκεκριμένα, στο κυρίως πρόγραμμα θα οριστούν έξι τοπικές μεταβλητές, εκ των οποίων η μία θα είναι τύπου χαρακτήρα (**char**) και οι υπόλοιπες ακέραιες. Η μεταβλητή τύπου χαρακτήρα καλείται **key** και σε αυτήν θα αποθηκεύεται ο χαρακτήρας του πληκτρολόγιου που επιλέγει ο χρήστης . Εν συνεχεία , την μεταβλητή key θα ελέγχει η **switch** (**key**) και αναλόγως της τιμής της θα γίνεται η επιλογή από το μενού. Οι υπόλοιπες ακέραιες μεταβλητές καλούνται x, y, z, num, m. Η μεταβλητή num αποθηκεύει τον αριθμό της τρέχουσας λίστας (0 – 99) και αρχικά τίθεται 0. Η μεταβλητή m χρησιμοποιείται μόνο για τις ανάγκες ορισμού της συνάρτησης getch(), δηλαδή m=getch(), η οποία θα καλείται για το προσωρινό σταμάτημα της ροής του προγράμματος μέχρι ο χρήστης να πιέσει κάποιο πλήκτρο. Οι υπόλοιπες ακέραιες μεταβλητές x, y, z χρησιμοποιούνται τόσο για το πέρασμα στοιχείου σε λίστα (η x) όσο και σαν ορίσματα των συναρτήσεων μέλους copy, concat, compare (οι x,y,z).

Σύμφωνα με τα παραπάνω, μερικά παραδείγματα χρήσης των συναρτήσεων είναι:

- a[x].copy (a[y]) /* Αντιγράφει την λίστα a[x] στην a[y] */
- a[num].pop() /* εξάγει το top στοιχείο της τρέχουσας λίστας a[num] */
- a[num].member (x) /*ελέγχει αν το x ανήκει στην τρέχουσα λίστα */

- a[y].concat (a [x], a[z]) /* Συνενώνει τις λίστες a[x],a[y] στην a[z] */

Έτσι, η main () συνάρτηση του προγράμματος θα είναι:

```
void main ()
{
        char key;               /* the answer-key in MAINMENU */
        int x,y,z, num,m;
        lifo a[100];            /* Creates an array-style object of LIFO */

/* default list number is 0 */
        num=0;
/* printing of main menu -- repeated by for loop*/
        for ( ;; )
         {
                cout << "\n";
                                        cout << "\n/* MAIN MENU */\n";
                                        cout << "\n(_1_)  Push x ";
                                        cout << "\n(_2_)  Pop x ";
                                        cout << "\n(_3_)  Print List ";
                                        cout << "\n(_4_)  Search element ";
                                        cout << "\n(_5_)  Check element";
                                        cout << "\n(_6_)  Choose Active List ";
                                        cout << "\n(_7_)  Copy List ";
                                        cout << "\n(_8_)  Compare Lists";
                                        cout << "\n(_9_)  Concatenate Lists";
                                        cout << "\n(_0_)  Return TOP    ";
                                        cout << "\n(_E_)  Exit\n ";
                                        cout << "\n< Current List : "
                                    << num <<",     Length = "
                                                << a[num].len () << " > ";
                cout << "\n> " ;
                cin >> key;
                cout << "\n\n";
                switch (key)
                {
                     case '1'  :   {   cout << "/* PUSH */\n";
                                        cout << "Enter integer to push : ";
                                   cin >> x;
                                        a[num].push (x);  /* push x onto a(num) */
                                        break;
                                    }
                case '2'  :   {   cout << " /* POP */ ";
                                        if (a[num].len () != 0 )
                                        {
                                        cout << "\nLast entered integer : "
                                        << a[num].pop ();
```

```
                        cout << "\n\nPress any key.";
                        m=getch();
                        break ;
                        }
                        else a[num].pop();
                        cout << "\nPress any key.";
                m=getch();
                        break;
                        }
        case '3'  :   {   cout << "/* PRINT LIST */\n";
                        a[num].print ();
                cout << "\n\nPress any key.";
                        m=getch();
                        break;
                        }
        case '4'  :   {   a[num].seek ();  break;  }
case '5'  :   {   cout << "/* CHECK */ \n";
                cout << "Enter int:"; cin >> x ;
                if (a[num].member (x)) cout <<"\nFound!\n";
                else cout << "\n Not Found.";
                cout << "\n\nPress any key.";
                m=getch();
                break;
                }
case '6'  :   {   cout << " /* CHANGE CURRENT LIST */ ";
                cout << "\nChoose LIFO list number (0-99): ";
                cin >> num; break;
                }
case '7'  :   {   cout << "/* COPY LISTS  */ " ;
                cout << "\nSource List : "; cin >> x ;
         cout << "\nTarget List : "; cin >> y;
                a[x].copy ( a[y] );
                cout <<"\n List : "<< y ;
                 a[y].print ();
                cout << "\n\nPress any key.";
                m=getch();
                break;
             }
case '8'  :   {   cout << "/* COMPARE LISTS */ ";
                cout << "\nFirst List : " ; cin >> x;
                cout << "\nSecond List : " ; cin >> y;
                if ( ! a[x].compare ( a[y] ) )
                        {cout << "\nLists are different." ;}
                else
                        {cout << "\nLists " << x << " and "
                        << y << " are the same. ";}
                 cout << "\n\nPress any key.";
```

```
                                m=getch();
                                 break;
                                }
                case '9'   :   {   cout << "/* LISTS' CONCATENATION */ ";
                                cout << "\nFirst List : "; cin >> x;
                                cout << "\nSecond List : "; cin >> y ;
                                cout << "\nTarget List : "; cin >> z;
                                a[y].concat ( a [x], a[z] );
                                cout << "\n List : " << z ;
                                a[z].print ();
                                cout << "\n\nPress any key.";
                                m=getch();
                                break;
                                }
                case '0'  :    {  cout << "/* TOP */ ";
                                if (a[num].len() != 0 )
                                {
                                cout << "\n Top Of List ["<< num << "] is : ";
                                cout << a[num].top();
                                cout << "\n\nPress Any Key To Continue.\n";
                                m=getch();
                                break;
                                }
                                else break;
                                }
                case 'E'  :
                case 'e'  :    {      goto telos ;  }
                default   : { cout << "Wrong. Try Again." ; break; }
                 }
        }

         telos:  cout << "\n\nGoodbye.";

}
```

ΠΑΡΑΤΗΡΗΣΗ:

Στην επιλογή case ‘2’ για εξαγωγή στοιχείου (pop), αλλά και στην case ‘0’ για την ανάγνωση της κορυφής της στοίβας (top), ελέγχεται πρώτα το μήκος της λίστας και αν είναι μεγαλύτερο του μηδενός καλείται η pop (η top αντίστοιχα). Αυτό γίνεται, γιατί στην περίπτωση που η λίστα δεν περιέχει στοιχείο, η συνάρτηση μέλους pop εκτυπώνει το μήνυμα UNDERFLOW και μετά επιστρέφει την τυπική τιμή –1. Με αυτόν τον τρόπο αποφεύγεται η περίπτωση να ζητηθεί pop στοιχείου και να τυπωθεί μήνυμα της μορφής:

- Last entered integer : UNDERFLOW

Δηλαδή θα εμφανίζονται:

- Last entered integer : x /* αν a[num].len() != 0 */

ή

- Error. UNDERFLOW. /*αν a[num].len() = 0 */

ΚΕΦΑΛΑΙΟ 3

3.1 TEMPLATES

Σε αυτό το κεφάλαιο δημιουργείται ένα πρότυπο κλάσεων lifo της οποίας τα αντικείμενα είναι LIFO λίστες (stacks), που όμως τα στοιχεία τους θα μπορούν να είναι είτε πραγματικοί αριθμοί (**float**) είτε χαρακτήρες (**char**). Στην ουσία, πρόκειται για γενίκευση της κλάσης lifo ακεραίων που δημιουργήθηκε στο προηγούμενο κεφάλαιο. Η γενίκευση αυτή επιτυγχάνεται εύκολα με χρήση των **templates** που υποστηρίζει η C++.

Σύμφωνα με τον ορισμό της Borland C++, τα πρότυπα (**templates**) ή παραμετροποιημένοι τύποι βοηθούν στον ορισμό μιας οικογένειας σχετικών συναρτήσεων ή κλάσεων. Τα πρότυπα χρησιμοποιούνται προκειμένου να δημιουργηθεί ένα σχέδιο περιγραφής γενικού τύπου συναρτήσεων ή κλάσεων όπου ο τύπος της συνάρτησης ή των δεδομένων-μελών και των συναρτήσεων-μέλους είναι παραμετροποιημένος. Για παράδειγμα, αν θέλουμε μια συνάρτηση να επιστρέφει τη μέγιστη μεταξύ των τιμών δύο ακέραιων μεταβλητών θα είχαμε:

```
- int  max(int  x, int y)
  {
    return (x > y) ? x : y;
  };
```

Αυτό είναι αρκετό στην περίπτωση που οι x, y είναι πάντοτε ακέραιες. Αν όμως θέλουμε οι x,y να είναι float ή char τύπου, προφανώς η παραπάνω συνάρτηση δεν λειτουργεί. Σε αυτήν την περίπτωση θα μπορούσαμε να δημιουργήσουμε, χρησιμοποιώντας την τεχνική της υπερφόρτωσης συναρτήσεων (**function overloading**) της C++, μερικές παρόμοιες συναρτήσεις max όπου το μόνο που θα άλλαζε θα ήταν ο τύπος της συνάρτησης και των παραμέτρων της. Αντί αυτού, για λόγους συντομίας στον κώδικα αλλά και συμπάγειας του προγράμματος, κάνουμε χρήση των προτύπων. Ο γενικός ορισμός τους είναι:

```
- template <class T>
  T function_name ( T x, T y, кок )
  {
   /*  Function_Body  */
  }
```

Με αυτόν τον τρόπο, ο τύπος της συνάρτησης, αλλά και των παραμέτρων που δέχεται, αναπαρίσταται από το όρισμα <class T> του προτύπου. Είναι δηλαδή παραμετροποιημένος, οπότε ο μεταγλωττιστής (**compiler**) θα αναλάβει να κάνει την σωστή απόδοση, δημιουργώντας την «ακριβή» συνάρτηση, ανάλογα με τον τύπο δεδομένων που χρησιμοποιήθηκε κατά την κλήση της στο πρόγραμμα. Έτσι, στο παραπάνω παράδειγμα της max συνάρτησης θα είχαμε με χρήση template:

```
-    template <class T>
-    T max (T x, T y)
-    {
```

```
-       return  (x > y ) ? x : y ;
-    };
```

Οπότε αν στο κυρίως πρόγραμμα είχαμε:

```
-    int k,x,y;
-    float a;
-    k=max(x,y);                    /* ορίσματα ακεραίων x, y, επιστρέφει int */
-    a = max (5.4,5.091);           /* ορίσματα πραγματικών, επιστρέφει float*/
```

Ανάλογη είναι και η διαδικασία για τον ορισμό μιας κλάσης με χρήση template. Με τον τρόπο αυτό μπορούμε να παραμετροποιήσουμε τον τύπο των συναρτήσεων μέλους αλλά και των δεδομένων:

```
-    template <class T>
-    class name
-    {
-     public:
-      T member_function_name ( T arguments if any);
-     private:
-      T member_data;
-    };
```

Αυτή την τεχνική χρησιμοποιούμε στη συνέχεια για τον ορισμό της κλάσης lifo λιστών με στοιχεία πραγματικούς ή χαρακτήρες. Ορίζουμε το πρότυπο template <class Type> της class lifo, για να παραμετροποιήσουμε τον τύπο των δεδομένων μελών αλλά και των επίμαχων συναρτήσεων-μέλους προκειμένου τα αντικείμενα της κλάσης να είναι λίστες είτε πραγματικών είτε χαρακτήρων κατά το δοκούν και φυσικά να έχουμε τις ίδιες λειτουργίες με τις λίστες ακεραίων. Φυσικά, οι δείκτες *list και *temp, που περιέχουν τα στοιχεία κάθε λίστας θα πρέπει να είναι τύπου Type, δηλαδή παραμετροποιημένα ώστε να μπορούμε να αποθηκεύουμε είτε float είτε char στοιχεία χρησιμοποιώντας την ίδια κλάση. Ακόμη οι συναρτήσεις μέλους που εξάγουν - επιστρέφουν στοιχεία, δηλαδή οι pop και top, θα πρέπει να είναι παραμετροποιημένου τύπου. Τέλος, οι συναρτήσεις μέλους που δέχονται στοιχεία για επεξεργασία, όπως οι push και member, θα δέχονται ορίσματα τύπου Type, έτσι ώστε να λειτουργούν και με float αλλά και με char στοιχεία. Έτσι θα έχουμε:

3.2 CLASS TEMPLATE

```
template <class Type>
class lifo
{
 public :
          lifo ()                    ;
          void push (Type)           ;
          Type  pop ()               ;
          void print ()              ;
          void seek ()               ;
          void copy (lifo&  )        ;
```

```
        void concat (lifo&, lifo&)  ;
        Bool compare (lifo&)        ;
        Bool member( Type )         ;
        int len ()                  ;
        Type top ()                 ;
        ~lifo ()                    ;
 private :
        Type *list,*temp            ;
        Type Top                    ;
        int length                  ;
};
```

Όπως φαίνεται παραπάνω, η κλάση lifo ελάχιστα μεταβλήθηκε στην ουσία από αυτήν του 2ου κεφαλαίου. Οι κύριες διαφορές εντοπίζονται στα επίμαχα δεδομένα *list, *temp, Top που περιέχουν τα στοιχεία της λίστας και αφορούν τον τύπο τους, αλλά και στις συναρτήσεις «εισόδου/εξόδου» push & member, pop & top στοιχείων προς /από την λίστα, που προφανώς ήταν αναγκαίο να παραμετροποιηθούν οι τύποι ή τα ορίσματα προκειμένου να μπορούν να χρησιμοποιήσουν τα γενικού τύπου δεδομένα-μέλη.

Με όμοιο τρόπο, αφού το σώμα της κλάσης περιέχει μόνον τα πρωτότυπά τους, ορίζονται λεπτομερώς και φυσικά με χρήση template οι συναρτήσεις-μέλους της κλάσης:

3.3 MEMBER FUNCTIONS

```
template <class Type>
lifo<Type>::lifo ()
        {
                length=0;
                list=new Type [length];
                temp = new Type [length];
                Top = list [length-1];
        }

template <class Type>
void lifo<Type> :: push (Type x )
{
        if ( length == 0 )
                {
                        delete  [] list;
                        list = new Type [++length];
                        list[length-1] = x;
                        Top = list [length-1];
                }
        else
                {
                 delete [] temp;
                 temp = new Type [length] ;
                 for (register i=0 ; i <= length-1 ; i++ )
```

```
                        {
                         temp [i] = list [i];
                        }
                delete [] list;
                list = new Type [++length];
                for ( i=0 ; i <= (length-2) ; i++ )
                        {
                                list [i] = temp [i];
                        }
                list [length-1] = x;
                Top=list [length-1];
               }
        }


template <class Type>
Type lifo<Type> :: pop ()
 {
        Type top;
        if  ( length == 1 )
                        { top = Top;

                          delete [] list;
                          list = new Type [--length];
                          Top= list[length-1];
                                return  top ;
                        }
        else if ( length == 0 )
                {
                         cout <<"\nError. Underflow.";
                                return -1;
                }
        else
               {
                delete [] temp;
                temp = new Type [length];
                for ( register i=0 ; i <= length-1 ; i++ )
                {    temp [i] = list [i];       }
                delete [] list;
                list = new Type [--length];
                for (  i=0 ; i <= length-1 ; i++ )
                {       list [i] = temp [i];       }
                top = temp [length] ;
                Top = list [length-1];
                return  top;
               }
 }
```

```
template <class Type>
void lifo<Type>::print ( )
        {
                cout << "\n";
                 for ( register i = 0 ; i <= length-1 ; i ++ )
                        {           cout << "[ " << list [i] << " ]" << "\t";                                }
       }

 template <class Type>
 void lifo<Type> :: seek ()
        {
                int m;
                Type x;
                cout << "Enter element to seek : ";
                cin >> x;
                if (! member (x) )
                 {
                        cout << "\nSorry. Not Found.\n";
                        cout << "\nPress any key.";
                        m=getch();
                 }
                else
                {
                  for ( register i=0 ; i<=length-1 ; i++)
                {
                        if (list[i] == x )
                         {
                        cout << "\nFound at " << (i+1)
                        << " position of current stack.";
                                cout << "\nPress any key.";
                                m=getch();
                          }
                        }
                }
        }

template <class Type>
Bool lifo<Type> :: member ( Type x )
        {
                for (register i=0 ; i<=length -1 ; ++i)
                 if ( list[i] == x ) return True;
                 return False ;
        }
```

```
template <class Type>
void lifo<Type> :: copy ( lifo &b  )
        {
                b.list = new Type [length];
                for (register i=0 ; i<=length-1 ; i++)
                b.list[i]=list[i];
                b.length=length;
                b.Top=list[length-1];
        }

template <class Type>
Bool lifo<Type> :: compare (lifo &b)
        {
                 if (b.length != length ) return False;
                for (register i=0 ; i<=length-1 ; i++)
                if ( b.list[i] != list[i] ) return False ;
                return True;
        }

template <class Type>
void lifo<Type> :: concat (lifo &b, lifo &c)
        {
                b.copy (c);
                for ( register i=0 ; i<=length-1 ; ++i)
                                        c.push ( list[i] ) ;

        }

template <class Type>
int lifo<Type> :: len ( )
                        {
                 return length;
                                }

template <class Type>
Type lifo<Type> :: top ()
        {
                return Top;
        }

template <class Type>
lifo<Type> :: ~lifo ()
                {                       delete [] list;                 delete [] temp;
       }
```

Από τα προηγούμενα είναι προφανές ότι η χρήση template για τον ορισμό των συναρτήσεων μέλους σχεδόν καθόλου δεν μεταβάλλει τον αλγόριθμό τους. Αυτό που αλλάζει είναι στην ουσία η

επικεφαλίδα της συνάρτησης στα οποία υπεισέρχεται πρώτη η έκφραση **template <class Type>**, καθώς και πριν από τον τελεστή εμβέλειας **::** , ο τύπος **<Type>**. Ο αλγόριθμος κάθε συνάρτησης είναι ίδιος με του 2ου κεφαλαίου, εκτός από τα σημεία στα οποία χρησιμοποιούνται είτε δεδομένα μέλη γενικού τύπου Type, όπως φυσικά στις συναρτήσεις push (type x) και pop(), είτε τοπικές μεταβλητές τύπου Type, όπως π.χ. στην συνάρτηση seek().

3.4 MAIN program

Στο κυρίως πρόγραμμα, τώρα, οι αλλαγές είναι ακόμη μικρότερου βαθμού. Απλώς, ορίζονται δύο ξεχωριστά αντικείμενα **a[100]** και **b[100]** της κλάσης lifo, το μεν **b** για τις λίστες με πραγματικούς αριθμούς, το δε **a** για τις λίστες με στοιχεία χαρακτήρες. Και εδώ, για λόγους ευκολίας και δυναμικότητας, τα αντικείμενα ορίζονται υπό μορφή διανυσμάτων, με αποτέλεσμα να έχουμε 100 λίστες με στοιχεία **float** και άλλες 100 με στοιχεία **char**. Ο ορισμός αυτών γίνεται με τις εντολές:

- **lifo <float> b [100]**
- **lifo <char> a [100]**

Λόγω της δομής του προγράμματος, δεν είναι δυνατό ο χρήστης να δουλεύει ταυτόχρονα και με λίστες float και με λίστες char στοιχείων, γι' αυτό στην αρχή του προγράμματος έχει γίνει μια προσθήκη για την επιλογή από τον χρήστη του τύπου της λιστών με τον οποίο θέλει να δουλέψει. Αυτό γίνεται με μια λογική συνθήκη διακλάδωσης **if-else**. Ελέγχεται η τιμή της char μεταβλητής choice και αν ισούται με '**c**' ή '**C**' , τότε το πρόγραμμα ακολουθεί τον κλάδο του προγράμματος που χρησιμοποιεί τα αντικείμενα a[100]. Αν η choice έχει τιμή '**f**' ή '**F**' τότε ακολουθείται ο κλάδος του προγράμματος που χρησιμοποιεί τα αντικείμενα b[100]. Στην ουσία και οι δύο κλάδοι είναι *αλγοριθμικά* όμοιοι, οι μόνες δε διαφορές τους εντοπίζονται στα διαφορετικά αντικείμενα a ή b που χρησιμοποιούν καθώς και στις παρακάτω αναφερόμενες τοπικές μεταβλητές:

Στον κλάδο για τις λίστες τύπου char χρησιμοποιείται η τοπική μεταβλητή **xf** τύπου char στην συνάρτηση push (Type x) για την προσθήκη χαρακτήρα σε λίστα με χαρακτήρες, ενώ στον κλάδο για λίστες πραγματικών χρησιμοποιείται η **xf1** τύπου float για την ίδια συνάρτηση αλλά προφανώς για λίστες με float στοιχεία. Το ίδιο επαναλαμβάνεται και στην συνάρτηση member (Type x).

Τελικά, η main () συνάρτηση για τη *templated* κλάση lifo που δημιουργήθηκε με βάση αυτήν του προηγούμενου κεφαλαίου θα είναι:

```
void main ()
{
        char key,choice,xf;          /* Key & choice  για τις επιλογές μενού */
        int x,y,z, num,m ;
        float xf1,amount;
        lifo <char> a[100];           /* Δημιουργία δυο αντικειμένων */
        lifo <float> b[100];          /* ένα για char , το άλλο για float*/

/* Αρχικά επιλεγμένη λίστα */
        num=0;

ARXH :
```

```
cout << " \nChoose the type of your LIFO List:";
cout << " \n(_F_)  Float. ";
cout << " \n(_C_)  Chars. ";
cout << " \n>";
cin  >> choice;
if ( choice == 'c' || choice == 'C' )        /* για  char- λίστες */
{
          for ( ;; )
           {
           cout << "\n";
                                        cout << "\n/* MAIN MENU */\n";
                                        cout << "\n(_1_)  Push element ";
                                        cout << "\n(_2_)  Pop element ";
                                        cout << "\n(_3_)  Print List ";
                                        cout << "\n(_4_)  Search element ";
                                        cout << "\n(_5_)  Check  element ";
                                        cout << "\n(_6_)  Choose List ";
                                        cout << "\n(_7_)  Copy Lists ";
                                        cout << "\n(_8_)  Compare Lists";
                                        cout << "\n(_9_)  Concatenate Lists";
                                        cout << "\n(_0_)  Return TOP ";
                                        cout << "\n(_E_)  Exit ";
                                        cout << "\n(_T_)  Perform SelfTest.\n";
                                        cout << "\n< Current List : " << num
                                   <<", Length = "
                                             << a[num].len () << " > ";
                                    cout << "\n>" ;
cin >> key;
cout << "\n\n";
switch (key)
{
          case '1'  :   {   cout << "/* PUSH */\n";
                            cout << "Enter element to push : ";
                            cin >> xf;
                   a[num].push (xf);
                            break;
                  }
          case '2'  :   {   cout << " /* POP */ ";
                            if (a[num].len() != 0 )
                                     {
                                     cout << "\nLast entered element : "
                                              << a[num].pop ();
                                     cout << "\n\nPress any key.";
                                     m=getch();
                                     break ;
                                     }
                            else a[num].pop();
```

```
                    cout << "\nPress any key.";
            m=getch();
                    break;
                  }
  case '3'  :   {   cout << "/* PRINT LIST */\n\n";
                    a[num].print ();
                    cout << "\n\nPress any key.";
                    m=getch();
                    break;
                   }
  case '4'  :   {
                a[num].seek ();  break;  }
  case '5'  :   {   cout << "/* CHECK */ \n";
            cout << "Enter element :"; cin >> xf ;
                    if (a[num].member (xf)) cout <<"\nFound!\n";
                    else cout << "\n Not Found.";
                    cout << "\n\nPress any key.";
                    m=getch();
                    break;
                    }
  case '6'  :   {   cout << " /* CHANGE CURRENT LIST */ ";
                      cout << "\nChoose LIFO list number (0-99): ";
                    cin >> num; break;
                   }
  case '7'  :   {   cout << "/* COPY LISTS  */ " ;
                    cout << "\nSource List : "; cin >> x ;
                    cout << "\nTarget List : "; cin >> y;
                    a[x].copy ( a[y] );
            cout <<"\n List : "<< y ;
                     a[y].print ();
                    cout << "\n\nPress any key.";
                    m=getch();
                    break;
                    }
  case '8'  :   {   cout << "/* COMPARE LISTS */ ";
                    cout << "\nFirst List : " ; cin >> x;
                    cout << "\nSecond List : " ; cin >> y;
                    if ( ! a[x].compare ( a[y] ) )
                            {cout << "\nLists are different." ;}
                    else
                            {cout << "\nLists " << x << " and "
                                    << y << " are the same. ";}
                     cout << "\n\nPress any key.";
                    m=getch();
                     break;
                    }
case '9'   :  {   cout << "/* LISTS' CONCATENATION */ ";
```

```
                                cout << "\nFirst List : "; cin >> x;
                                cout << "\nSecond List : "; cin >> y ;
                                cout << "\nTarget List : "; cin >> z;
                                a[y].concat ( a [x], a[z] );
                                cout << "\n List : " << z ;
                                a[z].print ();
                                cout << "\n\nPress any Key to Continue.";
                                m=getch();
                        break;
                                }
                case '0'  :    { cout << "/* TOP */";
                                 if (a[num].len() != 0 )
                                        {
                                 cout << "\nTop of List [" << num << "] is : ";
                                 cout << a[num].top() ;
                         cout << "\n\n Press Any Key To Continue.";
                                 m=getch();
                                break ;
                                        }
                                 else break;
                                }
                case 'E'  :
                case 'e'  :    {     goto TELOS ;  }
                case 'T'  :
                case 't'  :   {
                                 cout << "\n */ SelfTest /*";
                        cout << "\nEnter amount of PUSH actions to do: ";
        cin >> amount;
                        cout << "\n";
                 xf='!';
                        for (xf1=0;xf1<=amount;xf1++)
                                        {
                                         xf++; if (xf>='~') xf='!';
                                         a[num].push(xf);
                         cout << "\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
                         cout << "Progress: "
                                << ceil((100*xf1)/amount) << "%";
                                         }
                        cout << "\nList : " << num;
                        a[num].print();
                        cout << "\nPress any Key to Continue:";
                        m=getch();
                        break;
                }
default   : { cout << "Wrong. Try Again." ; break; }
 }
        }
```

```
        }
 else
if ( choice == 'f' || choice == 'F')    /* για  float- Λίστες */
        {
                for ( ;; )
                 {
                 cout << "\n";
                        cout << "\n/* MAIN MENU */\n";
                        cout << "\n(_1_)  Push element ";
                        cout << "\n(_2_)  Pop element ";
                        cout << "\n(_3_)  Print List ";
                        cout << "\n(_4_)  Search element ";
                        cout << "\n(_5_)  Check  element ";
                        cout << "\n(_6_)  Choose List ";
                        cout << "\n(_7_)  Copy Lists ";
                        cout << "\n(_8_)  Compare Lists";
                        cout << "\n(_9_)  Concatenate Lists";
                        cout << "\n(_0_)  Return TOP ";
                        cout << "\n(_E_)  Exit ";
                        cout << "\n(_T_)  Perform SelfTest.\n";
                        cout << "\n< Current List : " << num <<", Length = "
                                                        << b[num].len () << " > ";
                 cout << "\n>" ;
                 cin >> key;
                 cout << "\n\n";
switch (key)
        {
                case '1'  :   {   cout << "/* PUSH */\n";
                                  cout << "Enter element to push : ";
                                  cin >> xf1;
                                  b[num].push (xf1);
                                  break;
                                 }
                case '2'  :   {   cout << " /* POP */ ";
                                  if (b[num].len() != 0 )
                                  {
                                  cout << "\nLast entered element : "
                                          << b[num].pop ();
                                  cout << "\n\nPress any key.";
                                  m=getch();
                                  break ;
                                  }
                                  else b[num].pop();
                                  cout << "\nPress any key.";
                                  m=getch();
                                  break;
                                 }
```

```
        case '3'  :   {   cout << "/* PRINT LIST */\n\n";
                          b[num].print ();
                          cout << "\n\nPress any key.";
                          m=getch();
                          break;
                         }
        case '4'  :   {   b[num].seek ();  break;  }
        case '5'  :   {   cout << "/* CHECK */ \n";
                          cout << "Enter element :"; cin >> xf1 ;
                          if (b[num].member (xf1)) cout <<"\nFound!\n";
                          else cout << "\n Not Found.";
                          cout << "\n\nPress any key.";
                          m=getch();
                          break;
                          }
        case '6'  :   {   cout << " /* CHANGE CURRENT LIST */ ";
                          cout << "\nChoose LIFO list number (0-99): ";
                          cin >> num; break;
                         }
        case '7'  :   {   cout << "/* COPY LISTS  */ " ;
                          cout << "\nSource List : "; cin >> x ;
                          cout << "\nTarget List : "; cin >> y;
                          b[x].copy ( b[y] );
                          cout <<"\n List : "<< y ;
                           b[y].print ();
                          cout << "\n\nPress any key.";
                          m=getch();
                          break;
                          }
        case '8'  :   {   cout << "/* COMPARE LISTS */ ";
                          cout << "\nFirst List : " ; cin >> x;
                          cout << "\nSecond List : " ; cin >> y;
                          if ( ! b[x].compare ( b[y] ) )
                                    {cout << "\nLists are different." ;}
                          else
                                    {cout << "\nLists " << x << " and "
                                    << y << " are the same. ";}
                           cout << "\n\nPress any key.";
                          m=getch();
                           break;
                          }
case '9'   :  {   cout << "/* LISTS' CONCATENATION */ ";
                  cout << "\nFirst List : "; cin >> x;
                  cout << "\nSecond List : "; cin >> y ;
                  cout << "\nTarget List : "; cin >> z;
                  b[y].concat ( b [x], b[z] );
                  cout << "\n List : " << z ;
```

```
                        b[z].print ();
                        cout << "\n\nPress any key.";
                        m=getch();
                        break;
                        }
        case '0'  :    { cout << "/* TOP */";
                         if (b[num].len() != 0 )
                         {
                                  cout << "\nTop of List [" << num << "] is : ";
                                  cout << b[num].top() ;
                                  cout << "\n\n Press Any Key To Continue.";
                                  m=getch();
                                 break ;
                         }
                         else break;
                        }
        case 'E'  :
        case 'e'  :    {     goto TELOS ;  }
        case 'T'  :
        case 't'  :   {
                         cout << "\n */ SelfTest /*";
                         cout << "\nEnter amount of PUSH actions to do : ";
                         cin >> amount;
                         cout << "\n";
                         for (xf1=0;xf1 <= amount;xf1++)
                                  {
                                  b[num].push(xf1);
                                  cout << "\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
                                  cout << "Progress: "
                                           << ceil((100*xf1)/amount) << "%";
                                  }
                         cout << "\nList : " << num;
                         b[num].print();
                         cout << "\nPress any Key to Continue.";
                         m=getch();
                         break;
                        }
default   : { cout << "Wrong. Try Again." ; break; }
                  }
         }
         }
 else {cout << "\nWrong choice. Try Again.\n" ; goto ARXH;}

 TELOS:  cout << "\n\nGoodbye.";

}
```

ΠΑΡΑΤΗΡΗΣΕΙΣ

1. Σε σχέση με το κυρίως πρόγραμμα του προηγούμενου κεφαλαίου, η main() εδώ είναι διπλάσια σε αριθμό εντολών. Αυτό οφείλεται στο ότι υπάρχουν δυο κλάδοι σχεδόν όμοιοι, ένας για τον χειρισμό των λιστών με πραγματικούς αριθμούς και ένας για τις λίστες με χαρακτήρες. Η επιλογή ενός από των δυο γίνεται στην αρχή του προγράμματος με μια συνθήκη διακλάδωσης if – else. Σε περίπτωση που ο χρήστης πιέσει κάποιο άλλο πλήκτρο από αυτά που προβλέπονται ('f','F','c','C') τότε ο αλγόριθμος επιστρέφει στην αρχή του προγράμματος μέσω της ετικέτας 'ARXH' (βλ. Προτελευταία γραμμή της main ())
2. Στο μενού υπάρχει και μια επιπλέον εντολή 'SelfTest', η οποία δεν υπήρχε πριν. Είναι μια απλή ρουτίνα για τον έλεγχο των δυνατοτήτων κάθε λίστας. Ο χρήστης καλείται να δώσει τον αριθμό των διαδικασιών PUSH που θέλει να γίνουν στη λίστα και ο αριθμός αυτός αποθηκεύεται στην ακέραια μεταβλητή amount. Μετά αυτόματα το πρόγραμμα εισάγει νέα στοιχεία στη τρέχουσα λίστα με διαδοχικά PUSH τα οποία είναι πλήθους amount. Τέλος τα περιεχόμενα της λίστας τυπώνονται για του λόγου το αληθές. Έτσι μπορεί εύκολα κανείς να δει πόσα στοιχεία συνολικά μπορεί να αποθηκεύσει στις λίστες–αντικείμενα της κλάσης lifo.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1) **Data Structures & Algorithm Analysis in C++**, *Mark Allen Weiss*, Florida International University, 2000.
2) **C++ Programming**, *Sharam Hekmat,* Pragmatix Software Pty. Ltd, 1998.
3) **Δομές Δεδομένων**, *Παναγιώτη Αλεβίζου*, Πανεπιστήμιο Πατρών, 1999.
4) **Γλώσσες Προγραμματισμού ΙΙ**, *Όμηρου Ράγγου*, Πανεπιστήμιο Πατρών, 1998.
5) **Προγραμματισμός Η/Υ C++**, *Ανδρέα Λεοντίτση*, Πανεπιστήμιο Ιωαννίνων, 1996.