Αντικειμενοστρεφής Προγραμματισμός

Εργαστήριο 8

Πρότυπα (Templates)

Τα templates, που διακρίνονται σε function templates και class templates, μας επιτρέπουν να ορίσουμε συναρτήσεις και κλάσεις γενικής χρήσης, τις οποίες θα μπορούμε να χρησιμοποιήσουμε ώστε να χειρίζονται διαφορετικούς τύπους δεδομένων κάθε φορά, με τον ίδιο τρόπο. Για παράδειγμα, μπορούμε να ορίσουμε ένα function template μιας συνάρτησης abs (absolute), το οποίο θα μπορεί να χρησιμοποιηθεί να δώσει την απόλυτη τιμή κάθε είδους μεταβλητής, είτε είναι int, float, κλπ:

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

template <class T>              //function template
T abs(T n)
{
    return (n < 0) ? -n : n;
}

int main()
{
    int int1 = 5;
    int int2 = -6;
    long lon1 = 70000L;
    long lon2 = -80000L;
    double dub1 = 9.95;
    double dub2 = -10.15;
                                        //calls instantiate functions
    cout << "\nabs(" << int1 << ")=" << abs(int1);  //abs(int)
    cout << "\nabs(" << int2 << ")=" << abs(int2);  //abs(int)
    cout << "\nabs(" << lon1 << ")=" << abs(lon1);  //abs(long)
    cout << "\nabs(" << lon2 << ")=" << abs(lon2);  //abs(long)
    cout << "\nabs(" << dub1 << ")=" << abs(dub1);  //abs(double)
    cout << "\nabs(" << dub2 << ")=" << abs(dub2);  //abs(double)
    cout << endl;
    return 0;
}
```

Κάθε φορά που ο compiler συναντά την συνάρτηση abs( ), π.χ. την abs(dub1), δημιουργεί (instantiates) την συνάρτηση που αντιστοιχεί στον τύπο της μεταβλητής που δώσαμε ως παράμετρο (εδώ: double).

Άσκηση 1:
Να υλοποιήσετε ένα template συνάρτησης my_swap(a, b), το οποίο θα αντιμεταθέτει τις τιμές των a και b ως εξής:

```
int a = 10;
int b = 5;
my_swap(……);
cout << a; //να εμφανίζει πλέον 5 και όχι 10
```

Να την δοκιμάσετε με μεταβλητές τύπου string.

Άσκηση 2:
Να υλοποιήσετε ένα template συνάρτησης add(a, b) η οποία θα προσθέτει δύο αριθμούς:

```
int a = 10;
int b = 5;
int c = add(a, b);
```

Ερώτηση: Μπορείτε να χρησιμοποιήσετε την add( ) για να προσθέσετε δύο αντικείμενα μιας κλάσης (όχι απλούς τύπους, όπως int, double κλπ.)? Τι χρειάζεται για να γίνει αυτό?

Function templates με πολλαπλές παραμέτρους και πολλαπλά template arguments

Μια συνάρτηση μπορεί να πάρει πολλαπλές παραμέτρους ως template. Επίσης, μπορεί να δεχτεί ως παράμετρο, δύο ή και περισσότερα templates, όπως φαίνεται παρακάτω:

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

//function returns index number of item, or -1 if not found
template <class atype, class btype>
btype find(atype* array, atype value, btype size)
{
   for(btype j=0; j<size; j++)     //note use of btype
      if(array[j]==value)
         return j;
   return static_cast<btype>(-1);
}

char chrArr[] =   {1, 3, 5, 9, 11, 13};  //array
char ch = 5;                              //value to find
int intArr[] =    {1, 3, 5, 9, 11, 13};
int in = 6;
long lonArr[] =   {1L, 3L, 5L, 9L, 11L, 13L};
long lo = 11L;
double dubArr[] = {1.0, 3.0, 5.0, 9.0, 11.0, 13.0};
double db = 4.0;

int main()
{
    cout << "\n 5 in chrArray: index=" << find(chrArr, ch, 6);
    cout << "\n 6 in intArray: index=" << find(intArr, in, 6);
    cout << "\n11 in lonArray: index=" << find(lonArr, lo, 6);
```

```
    cout << "\n 4 in dubArray: index=" << find(dubArr, db, 6);
    cout << endl;
    return 0;
}
```

Class templates

Τα class templates μας επιτρέπουν αντίστοιχα να ορίσουμε μια γενικής χρήσης κλάση η οποία θα μπορεί να χρησιμοποιηθεί ανάλογα με την περίσταση, για διαφορετικά είδη member variables. Κλασσικό παράδειγμα αποτελούν τα class templates για γενικής χρήσης κατασκευές, όπως είναι οι ουρές, οι συνδεδεμένες λίστες, οι στοίβες κ.α., που μπορούν να αποθηκεύσουν κάθε είδους μεταβλητή, είτε πρόκειται για απλή μεταβλητή (int, char, κλπ.) είτε για κλάση ορισμένη από τον χρήστη.

```
#include <iostream>
using namespace std;
const int MAX = 10;

template <class Type>
class Stack
{
   private:
      Type st[MAX];             //stack: array of any type
      int top;                  //number of top of stack
   public:
      Stack();                  //constructor
      void push(Type var);      //put number on stack
      Type pop();               //take number off stack
};

template<class Type>
Stack<Type>::Stack()            //constructor
{
   top = -1;
}

template<class Type>
void Stack<Type>::push(Type var) //put number on stack
{
   st[++top] = var;
}

template<class Type>
Type Stack<Type>::pop()          //take number off stack
{
   return st[top--];
}

int main()
{
   Stack<float> s1;      //s1 is object of class Stack<float>

   s1.push(1111.1F);     //push 3 floats, pop 3 floats
   s1.push(2222.2F);
   s1.push(3333.3F);
   cout << "1: " << s1.pop() << endl;
   cout << "2: " << s1.pop() << endl;
   cout << "3: " << s1.pop() << endl;

   Stack<long> s2;       //s2 is object of class Stack<long>

   s2.push(123123123L);  //push 3 longs, pop 3 longs
   s2.push(234234234L);
   s2.push(345345345L);
```

```
    cout << "1: " << s2.pop() << endl;
    cout << "2: " << s2.pop() << endl;
    cout << "3: " << s2.pop() << endl;
    return 0;
}
```

Άσκηση 3:
Να ορίσετε μια κλάση Fraction για την αναπαράσταση κλασμάτων, και να την υλοποιήσετε ως class template, ώστε να χρησιμοποιούμε integers, doubles, κλπ.. Να υλοποιήσετε επίσης:

α) τους απαραίτητους constructors της κλάσης
β) μια συνάρτηση display
γ) να υπερφορτώσετε τον operator+ για την προσθήκη δύο κλασμάτων

Στην main( ) να χρησιμοποιήσετε μια στοίβα όπως αυτή του προηγούμενου παραδείγματος (μεγέθους 10).

Να τοποθετήσετε (μέσω push) 10 κλάσματα σε αυτήν (τα αντικείμενα τύπου Fraction να έχουν παραμέτρους τύπου int).

Στην συνέχεια, σε ένα loop, να αφαιρέσετε (μέσω pop) ένα-ένα τα κλάσματα και να τα προσθέσετε μεταξύ τους ώστε στο τέλος να προκύψει το συνολικό κλάσμα (το άθροισμα των 10 κλασμάτων).