Шаблони (Templates)

Потреба в шаблонах

1) Метод для всіх типів

```
int max(int a, int b) {
   return a>b?a:b;
}
char max(char a, char b) {
   return a>b?a:b;
}

short max(short a, short b) {
   return a>b?a:b;
}
```

Питання:

unsigned?? long ?? long long???

int max(short a, int b)???

2) Структура даних

```
class Stack_int {
    private:
    int data;
    Stack* address;

***
public:
  int pop() {…}
  void push( int x) {...}

};
```

Class Stack_char ???

Class Stack_string ???

Clas Stack_GenericClass????

Шаблони функцій(Template functions)

Декларація:

```
template <class myType1, class myType2, ... >
// or template <typename myType1, ...>
 <ТИП1> Функція (myType1 arg1, myType1 arg2, )
```

1)

```
template <class myType> // or template <typename myType>
myType GetMax (myType a, myType b) {
 return (a>b?a:b);
}
```

2)

```
template <typename myType> //
myType GetMax (myType a, myType b) {
 return (a>b?a:b);
}
```

Використання шаблонів функцій в RTTI

```
int x,y,z;
z =GetMax <int> (x,y);

char x1,y1,z1;
z1 =GetMax <char> (x1,y1);


int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

```
//Runtime type identification
#include <iostream>
using namespace std;
template <class T>
      T GetMax (T a, T b) {
            return (a>b?a:b);
     }
int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax(i,j);
  n=GetMax(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

Приклад

```
using namespace std;
template <class T, int max>
int arrMin(T arr[], int n) {
    int m = max;
    for (int i = 0; i < n; i++)
       if (arr[i] < m){
           m = arr[i];
    }
    return m;
}
```

```
int main() {
    int arr1[]  = {10, 20, 15, 12};
    int n1 = sizeof(arr1)/sizeof(arr1[0]);

    char arr2[] = {1, 2, 3};
    int n2 = sizeof(arr2)/sizeof(arr2[0]);

/* Другий аргумент-шаблон для arrMin
повинен бути константою    */
cout << arrMin<int, 10000>(arr1, n1) << endl;
cout << arrMin<char, 256>(arr2, n2);
 return 0; }
```

Шаблон класу (Class template)

```
template <class T1,class T2, …, class Tn > /* or template <typename myType1, ...>   */
class <Ім'я класу> {
 ****

 T1 member11, member12 …
 T2 member21, member22 …
 ...
}
```

Приклад: (<pair.h>)

```
template <class T>  class mypair {
    T values [2];
  public:
    mypair (T first, T second)
    {
      values[0]=first; values[1]=second;
    }
};
```

Приклад шаблону класу

```
// class templates
#include <iostream>
using namespace std;
template <class T>
class mypair {
    T a, b;
  public:
    mypair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};
template <class T>
T mypair<T>::getmax ()
{
  T retval;
  retval = a>b? a : b;
  return retval;
}
```

```
int main () {
  mypair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

Приклад

```
#include<iostream>
using namespace std;
 template<class T, class U>
class A  {
     T x;
     U y;
public:
 A() {
 cout<<"Constructor called"<<endl;
 }
 T usageMethod(T x, U y);
 T getX();
 void setX(T x)
};
template<typename T, typename U>
T A::usageMethod(T x, U y) {
   cout<<”Hi”;
   T z = x + static_cast<T>(y);
   return z;
}
```

```
template<typename T>
T A::getX() {
   return x;
}

template<typename T>
void A::setX(T x ) {
   this->x = x;
}

int main()  {
 A<char, char> a;
 A<int, double> b;
 a.setX(‘a’);a.setY(‘b’)
 a.usageMethod(‘c’,’d’);
 b.setX(1); b.setY(0);
 b.usageMehtod(0,0);
 return 0;
}
```

Приклад: аргумент шаблону по замовченню

```
#include<iostream>
 using namespace std;
template<class T, class U = char>
class A  {
public:
     T x;
     U y;
  A(T x_, U y_): x(x_),y(y_){
    cout<<"Constructor Called"<<endl;
}
 A() {
    cout<<"Constructor Called"<<endl;
  }
 };
```

```
int main()  {
    A<char> a;// Виклик A<char, char>
    A<int,int>  b; // Виклик A<int, int>

return 0; }
```

Спеціалізація шаблонів (template specialization)

```
#include <iostream>
using namespace std;
// class template

template <class T>
class mycontainer {
    T element;
  public:
    mycontainer (T arg) {
                element=arg;
}
    T increase () {
      return ++element;
 }
};
```

```
template <>// template specialization:
class mycontainer <char> {
    char element;
  public:
    mycontainer(char arg) {element=arg;}
    char uppercase () {
      if((element>='a')&&(element<='z'))
      element+='A'-'a';
      return element;
    }
};

int main () {
  mycontainer<int> myint (7);
  mycontainer<char> mychar ('j');
  cout << myint.increase() << endl;
  cout << mychar.uppercase() << endl;
  return 0;
}
```