Лекция № 16

Использование шаблонов

Шаблоны

Повторное использование кода.

Можно создавать функции-шаблоны и классы-шаблоны.

Для функции-шаблона или класса-шаблона тип данных является параметром.

Можно использовать одну функцию или класс для нескольких различных типов данных без необходимости явного написания кода для каждой версии в отдельности.

Функция - шаблон

Функция - шаблон определяет общий набор операций, который будет применен к данным различных типов.

Можно применять некоторые
общие алгоритмы к широкому кругу данных.

Функция - шаблон, функция, которая может автоматически перегружать сама себя.

Шаблон используется для создания каркаса функции, оставляя компилятору
реализацию подробностей.

Функция - шаблон

Функции-шаблоны создаются с использованием ключевого слова **template** (шаблон).

Общая форма функции-шаблона имеет следующий вид:

```
template <class type1> возвращаемый_тип
имя_функции(список параметров);
{
// тело функции
}
-----------------------------------------
// 2 типа-шаблона
template <class type1, class type2>
```

Пример

```
#include <stdio.h>  // Обмен значений переменных
#include <string.h>
#include <conio.h>

template <class T> T swap(T *a, T *b)
{
   T temp;
   temp=*a;  *a=*b;  *b=temp;
}

int main(void)
{
   int a=3, b=7;
   char s='W', p='Z';

   swap(&a,&b);  printf("%d   %d\n",a,b);
   swap(&s,&p);  printf("%c   %c\n",s,p);
   getch();
   return 0;
}
```

Пример

```
#include <stdio.h> // Поиск максимального значения
#include <string.h>
#include <conio.h>

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

int main(void)

{

   printf("%d \n",max(100, 200));

   printf("%f \n",max(5.4321, 1.2345));
   getch();
   return 0;
}
```

Явная перегрузка функций-шаблонов

```
#include <stdio.h>
#include <conio.h>
template <class T> T swap(T *a, T *b)
{
   T temp; temp=*a;  *a=*b;  *b=temp;
}

void swap(int *a, int *b)
{
 int temp;  temp = *a; *a = *b; *b = temp;
 printf("integer only.....");
}

int main(void)
{
   int a=3, b=7;
   char s='W', p='Z';
   swap(&a,&b);  printf("%d   %d\n",a,b);
   swap(&s,&p);  printf("%c   %c\n",s,p);
   getch();
   return 0;
}
```

Перегрузка и функции-шаблоны

Функции-шаблоны более ограничивающие по сравнению с перегруженными функциями.

Перегруженные функций могут выполнять различные действия (алгоритм может быть разный).

Для функции-шаблона необходимо выполнять одни и те же общие действия, и только тип данных может быть различным.

Пример перегрузка функций

```
#include <stdio.h>
#include <conio.h>

void func(int i)
{
 printf("i = %d", i);
}

void func(char c)
{
  int k;
  for (k=1; k<=10; k++) printf("%4c",c++);
}

int main(void)
{
  func(10);
  func('A');
  getch();
  return 0;
}
```

Шаблоны классов

Повторное использование кода

Шаблон класса определяет типонезависимый класс, который в дальнейшем служит для создания объектов требуемых типов.

Если компилятор C++ встречает объявление объекта, основанное на шаблоне класса, то для построения класса требуемого типа он будет использовать типы, указанные при объявлении.

Класс массив (1)

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
class vector
{
  public:
    vector(int, int, int);
    int sum(void);
    void print(void);
  private:
    int data[100]; // ограничение на 100 элементов
    int size;
};

vector::vector(int s, int k1, int k2)
{
  int k;
  if (s>100)
  {
    printf("Error while creating a vector ! Must <100"); // проверка
    exit(1);
  }
  size=s;
  for (k=0; k<size; k++) data[k]=rand()%(k2-k1+1)+k1;
}
```

Класс массив (2)

```
int vector::sum(void)
{
   int s,k;
   s=0;
   for (k = 0; k<size; k++) s+=data[k];
   return(s);
}

void vector::print(void)
{
   int k;
   for (k = 0; k<size; k++) printf("%3d",data[k]);
}

int main(void)
{
  printf("Press any key to create a vector\n");
  int k=0; while(!kbhit()) k++; getch(); srand(k);

  vector A(5,10,25);
  A.print();
  printf("\nSumma = %3d",A.sum());
  getch();
  return 0;
}
```

Класс массив с шаблонами (2)

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
template <class T> class vector
{
  public:
    vector(int);
    void add(int,T);
    void print(const char*);
  private:
    T data[100]; // ограничение на 100 элементов
    int size;
};
template <class T> vector<T>::vector(int s) // ----- конструктор
{
  if (s>100)
  { printf("Error while creating a vector ! Must <100"); exit(1); }
  size=s;
}
template <class T> void vector<T>::add(int k, T value) // --- добавление
{
  if (k>size)
  { printf("Error in index ! index = %d  total = %d",k,size); exit(1);}
  data[k]=value;
}
```

Класс массив с шаблонами (2)

```
template <class T> void vector<T>::print(const char* format) // вывод
{
  int k;
  for (k=0; k<size; k++) printf(format,data[k]);
}


int main(void)
{

  vector <int> A(3);
  A.add(0,10); A.add(1,13); A.add(2,17);
  A.print("%4d");
  printf("\n\n");

  vector <float> F(5);
  F.add(0,-3.84); F.add(1,1.63); F.add(2,7.43);
  F.add(3,6.3); F.add(4, 8.9);
  F.print("%8.3f");

  getch();
  return 0;
}
```

Наследование в шаблонах

Шаблоны классов, как и классы, поддерживают механизм наследования.

Все основные идеи наследования при этом остаются неизменными, что позволяет построить иерархическую структуру шаблонов, аналогичную иерархии классов.

Библиотека STL

Первая часть содержит объявления стандартных функций, макросы, глобальные константы, унаследованные у классического языка С (файлы *.h)

Вторая часть содержит классы, шаблоны и дополнительные средства языка С/С++ (файлы без расширения .h)

Основную часть библиотеки С++ составляет стандартная библиотека шаблонов-Standart Template Library (STL). Для использования подключаем «стандартное пространство имен»

```
using namespace std;
```

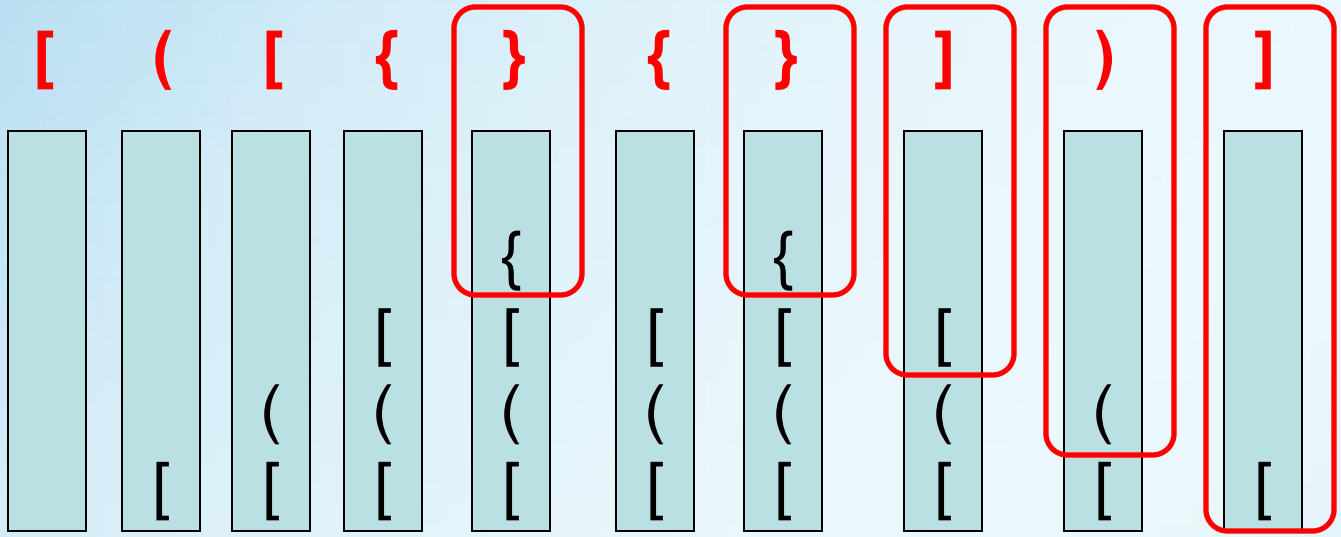
Стек

Стек - структура данных, в которой элемент, занесенный первым, извлекается последним (структура данных FIFO).
Определены две операции: занесение и выборка.
Для работы со стеком достаточно одной переменной — указателя на его вершину.

push( )- добавление последнего элемента
pop( ) – удаление последнего элемента
top( ) – ссылка на последний элемент
empty( ) – проверка на наличие элементов
size( ) – число элементов

Проверка скобочного выражения

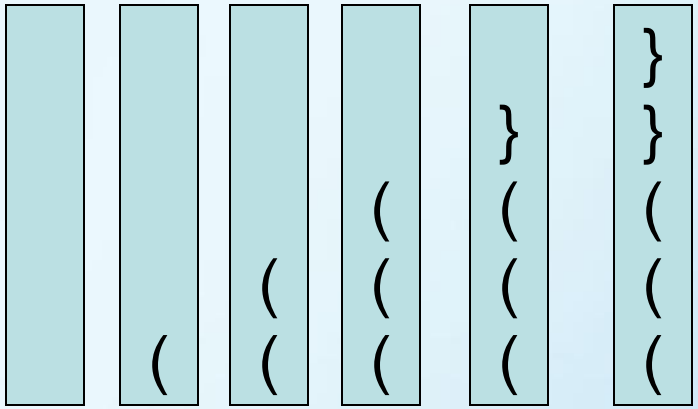
[([{}{}])] правильно



стек пуст, ошибки нет.

(((}}] ошибка

(((} }]



в стеке символы, ошибка

Проверка скобочного выражения

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

int main(void)
{
  char str[100]="[](()}";
  int k=0;
  stack <char> s; // стек
  s.push(str[k]);
  while (str[++k])
  {
    if (s.size()==0) { s.push(str[k]); continue; }

    if ((s.top()=='{')&&(str[k]=='}') ||
        (s.top()=='(')&&(str[k]==')') ||
        (s.top()=='[')&&(str[k]==']')  )  s.pop();
    else
      s.push(str[k]);
  }
  printf("Stack size = %d\n", s.size());
  system("pause"); return 0;
}
```