**Практическое занятие 10\_1**

**«Шаблоны функций»**

**Теория**

Многие алгоритмы не зависят от типов обрабатываемых данных. В этом случае желательно иметь возможность создавать функции, реализующие нужный алгоритм и настраивающиеся на основе получаемой информации на применение запрограммированного алгоритма к конкретному типу данных. Одним из возможных способов решения этой задачи может быть передача указателя на данные и длины обрабатываемых данных. Однако в этом случае отсутствует возможность контроля типов. Другой способ состоит в написании нескольких перегружаемых функций. Но в этом случае программа должна содержать несколько одинаковых по логике функций, причем для каждого типа данных должна быть своя функция. Это тоже не очень удобно.

В языке C++ есть удобное средство программирования – шаблоны функций. С помощью шаблона функции можно запрограммировать алгоритм, который будет применяться к данным разных типов, а конкретный тип данных передается в функцию в виде параметра на этапе компиляции. Компилятор генерирует код, соответствующий переданному типу данных. Формат простейшей функции-шаблона выглядит следующим образом

Template <class Type> заголовок функции

{

/\* тело функции \*/

}

Вместо слова Type может использоваться произвольное имя, например, TD. Шаблон функции может содержать несколько параметров, каждый из которых может быть типом или переменной. Приведем пример программы с функцией, осуществляющей сортировку элементов массива методом пузырька.

// пример использование шаблона функции

#include "stdio.h"

#include "conio.h"

#include <math.h>

template <class TD>

void sortmin( int n,TD \*a)

{int i,j,k;

bool flag;

TD b; //переменная b будет иметь тот же тип, что и второй фактический

// параметр, передаваемый в функцию

k=0;

do

{k++;

flag=false;

for (i=0;i<n-k;i++)

if (a[i]>a[i+1])

{ b=a[i];

a[i]=a[i+1];

a[i+1]=b;

flag=true;

}

}while (flag);

}

int main()

{ const int nn=20;

int n,i;

float a[nn];

char ch[nn];

printf("Wwedite kol-wo elementow\n");

scanf\_s("%d",&n);

printf("\nWwedite elementi\n");

for (i=0;i<n;i++)

scanf\_s("%f",&a[i]);

printf("\n massiw elementow\n");

for (i=0;i<n;i++)

printf("%f ",a[i]);

sortmin(n,a);

printf("\n otsortirowannij massiw elementow\n");

for (i=0;i<n;i++)

printf("%f ",a[i]);

\_getch();

printf("\n Simwoli Wwedite kol-wo elementow\n");

scanf\_s("%d",&n);

printf("\n Wwedite elementi\n");

//считывает символ 10 перевод влево курсора

i=0;

do

{scanf\_s("%c",&ch[i]);

if (ch[i]!=10) i+=1;

}while(i<n);

printf("\nmassiw elementow\n");

for (i=0;i<n;i++)

printf("%c ",ch[i]);

sortmin(n,ch);

printf("\n Otsortirowannij massiw elementow\n");

for (i=0;i<n;i++)

printf("%c ",ch[i]);

\_getch();

return 0;

}

Первый вызов функции, который использует конкретный тип данных, приведет к созданию компилятором кода для соответствующей версии функции. При повторном вызове функции с тем же типом данных новый код не генерируется.

**Практика**

1. Определить шаблон функций, вычисляющих абсолютные значения числовых величин разных типов (lab10\_1.cpp).
2. Определить шаблон функций для обмена значений двух параметров (lab10\_2.cpp).
3. Определить шаблон функций для поиска в массиве (lab10\_3.cpp).
4. Прототип шаблона для семейства функций (lab10\_4.cpp).
5. Создать программу с шаблоном функции, осуществляющей сортировку элементов массива методом пузырька (lab10\_5.cpp).

**Задания:**

1. Определить шаблон функции, определяющей максимальное из двух значений параметров.
2. Написать шаблон функции для вычисления суммы элементов массива и нахождения максимального элемента массива. Проверить работу этой функции для массива int и double.
3. Написать шаблон функции для вычисления среднего значения элементов массива и проверить эту функцию для типа double.
4. Написать шаблон функции для вычисления произведения всех положительных элементов массива, проверить для типов int и double.