**Лабораторная работа 22. Головачев Д. А. Y2231**

**Задание 1.** Написать программу, которая сортирует по убыванию элементы одномерного массива, который может состоять из различных числовых типов данных. Используйте шаблон функции. Все необходимые данные должны передаваться функции в качестве параметров. Продемонстрировать работу шаблона.

Код:

#include <iostream>

#include <stdio.h>

#include <cstring>

using namespace std;

template <class Type>

void sorting(Type \*b, int n) {

Type a;

for (int i = 0; i < (n-1); i++) {

for (int l = 0; l < (n-i-1); l++) {

if ((b[l] < b[l+1])) {

a = b[l];

b[l] = b[l+1];

b[l+1] = a;

} else {continue;}

}

}

}

int main()

{

const int n = 5;

int a[n] = {5,1,6,4,8};

float b[n] = {0.5, 0.1, 0.6, 0.4, 0.8};

char str[80] = "Exams are coming";

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

cout<<a[i]<<" ";

}

cout<<endl;

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

cout<<b[i]<<" ";

}

cout<<endl;

for (int i = 0; i < (strlen(str)-1); i++) {

cout<<str[i];

}

sorting(a, n);

sorting(b, n);

int g=strlen(str);

sorting(str,g);

cout<<endl;

cout<<endl;

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

cout<<a[i]<<" ";

}

cout<<endl;

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

cout<<b[i]<<" ";

}

cout<<endl;

for (int i = 0; i < (strlen(str)-1); i++) {

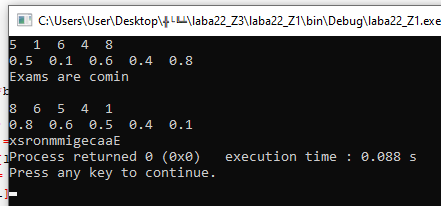
cout<<str[i];

}

return 0;

}

Результат выполнения:



Как видно по скриншоту, массивы символов от сортировались по убыванию для типов int, float и char. То есть шаблон функции сортировки по убыванию работает отлично.

**Задание 2.** Прочитать страницы 211-221 учебника Т.А.Павловской “Программирование на языке высокого уровня C/C++”. Понять прочитанное.

- я, честное слово, выполнил))

**Задание 3.** Создать шаблон класса “стек” (см. лекцию о динамических структурах данных). У класса должно быть хотя бы два поля различных параметризованных типов.

Код(в нем комментарии есть):

#include <stdio.h> //для scanf

#include <cassert> // для assert

#include <iomanip> // для setw

#include <iostream> //она, честно, нужна

using namespace std;

template <typename T,typename F>

class Stack

{

private:

T \*stackPtr;

F \*stackPtr2;

const int size; //максимальный размер

int top; //размер

public:

Stack(int = 10); // по умолчанию размер стека равен 10 элементам

Stack(const Stack<T,F> &); // конструктор копирования

~Stack(); // деструктор

inline void push(const T &,const F & ); // поместить элемент в вершину стека

inline T pop(); // удалить элемент из вершины стека и вернуть его

inline void printStack(); // вывод стека на экран

inline const T &Peek(int ) const; // n-й элемент от вершины стека

inline int getStackSize() const; // получить размер стека

inline T \*getPtr() const; // получить указатель на стек

inline F \*getPtr2() const;

inline int getTop() const; // получить номер текущего элемента в стеке

};

// реализация методов шаблона класса Stack

// конструктор Стека

template <typename T,typename F>

Stack<T,F>::Stack(int maxSize) :

size(maxSize)

{

stackPtr = new T[size];

stackPtr2 = new F[size];

top = 0;

}

// конструктор копирования

template <typename T,typename F>

Stack<T,F>::Stack(const Stack<T,F> & otherStack) :

size(otherStack.getStackSize())

{

stackPtr = new T[size];

stackPtr2 = new F[size];

top = otherStack.getTop();

for(int ix = 0; ix < top; ix++) {

stackPtr[ix] = otherStack.getPtr()[ix];

stackPtr2[ix] = otherStack.getPtr2()[ix];}

}

// деструктор Стека

template <typename T,typename F>

Stack<T,F>::~Stack()

{

delete [] stackPtr;

delete [] stackPtr2;

}

// функция добавления элемента в стек

template <typename T,typename F>

inline void Stack<T,F>::push(const T &value,const F &value2)

{

assert(top < size);

stackPtr[top] = value;

stackPtr2[top++] = value2;

}

// функция удаления элемента из стека

template <typename T,typename F>

inline T Stack<T,F>::pop()

{

assert(top > 0);

stackPtr[--top];

stackPtr2[top];

}

// вывод стека на экран

template <typename T,typename F>

inline void Stack<T,F>::printStack()

{

for (int i = top - 1; i >= 0; i--) {

cout << "|" << stackPtr[i];

cout << "|" << stackPtr2[i] << endl;}

}

// вернуть размер стека

template <typename T,typename F>

inline int Stack<T,F>::getStackSize() const

{

return size;

}

// вернуть указатель на стек (для конструктора копирования)

template <typename T,typename F>

inline T \*Stack<T,F>::getPtr() const

{

return stackPtr;

}

template <typename T,typename F>

inline F \*Stack<T,F>::getPtr2() const

{

return stackPtr2;

}

// вернуть размер стека

template <typename T,typename F>

inline int Stack<T,F>::getTop() const

{

return top;

}

int main()

{

Stack<char,int> stackSymbol(5);

int ct = 0;

char ch;

int f;

cout << "Enter 10 simvols\n";

while (ct++ < 5)

{

scanf("%c",&ch);

scanf("%d",&f);

stackSymbol.push(ch,f); // помещаем элементы в стек

}

cout << endl;

stackSymbol.printStack(); // печать стека

cout << "\n\nDelete element from stek\n";

stackSymbol.pop();

stackSymbol.printStack(); // печать стека

Stack<char,int> newStack(stackSymbol);

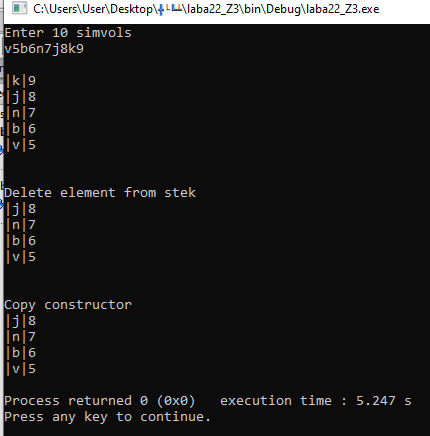
cout << "\n\nCopy constructor\n";

newStack.printStack();

return 0;

}

Результат:



Как видите, все операции работают хорошо, был создан стек объектов из пяти элементов, было продемонстрированно удаление и копирование.