**Статическая библиотека в C++**

Создадим статическую библиотеку интерполяции функции по имеющемуся набору известных значений. Создавать библиотеку будем в командной строке с помощью компилятора GCC для Windows.

Для создания библиотеки создаём 2 файла: *interpolation.h* и *interpolation.cpp,* в которые поместим все наши методы интерполирования функции.

1. Файл *interpolation.h* содержит названия функций: **(1)**

/\*Функция построения канонического полинома для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

void kanon(double t,double \*x,double \*y,int n,double \*s);

/\*Функция построения полинома Ньютона для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

void newton(double t,double \*x,double \*y,int n,double \*s);

/\*Функция построения полинома Лагранжа для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

void lagrange(double t,double \*x,double \*y,int n,double \*s);

/\*Функция построения линейного сплайна для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

void lininterp(double t,double \*x,double \*y,int n,double \*p);

/\*Функция построения кубического сплайна для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

void spline(double t,double \*x,double \*y,int n,double \*s);

1. Содержание файла *interpolation.cpp,* которыйвключает коды всех наших функций: **(2)**

#include <math.h>

int gauss(double \*\*aa,double \*pp,int n,double \*x)

{

int i,j,k,r;

double c,m,max,s;

double \*\*a, \*p;

a=new double \*[n];

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

a[i]=new double[n];

p=new double [n];

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

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

a[i][j]=aa[i][j];

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

p[i]=pp[i];

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

{

max=fabs(a[k][k]);

r=k;

for(i=k+1;i<n;i++)

if (fabs(a[i][k])>max)

{

max=fabs(a[i][k]);

r=i;

}

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

{

c=a[k][j];

a[k][j]=a[r][j];

a[r][j]=c;

}

c=p[k];

p[k]=p[r];

p[r]=c;

for(i=k+1;i<n;i++)

{

for(m=a[i][k]/a[k][k],j=k;j<n;j++)

a[i][j]-=m\*a[k][j];

p[i]-=m\*p[k];

}

}

if (a[n-1][n-1]==0)

if(p[n-1]==0)

return -1;

else return -2;

else

{

for(i=n-1;i>=0;i--)

{

for(s=0,j=i+1;j<n;j++)

s+=a[i][j]\*x[j];

x[i]=(p[i]-s)/a[i][i];

}

return 0;

}

}

void kanon(double t,double \*x,double \*y,int n,double \*s)

{

int i,j;

double \*\*c,\*a;

c=new double \*[n];

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

c[i]=new double [n];

a=new double [n];

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

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

c[i][j]=pow(x[i],j);

gauss(c,y,n,a);

for (\*s=i=0;i<n;i++)

\*s=\*s+a[i]\*pow(t,i);

}

void newton(double t,double \*x,double \*y,int n,double \*s)

{

int i,j;

double \*\*c,\*a,p;

c=new double \*[n];

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

c[i]=new double [n];

a=new double [n];

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

c[i][0]=y[i];

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

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

if (i<j)

c[i][j]=0;

for (j=1;j<n;j++)

for (i=1;i<n;i++)

if (i>=j)

c[i][j]=(c[j-1][j-1]-c[i][j-1])/(x[j-1]-x[i]);

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

a[i]=c[i][i];

for (\*s=i=0;i<n;i++)

{

p=1;

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

p=p\*(t-x[j]);

\*s=\*s+a[i]\*p;

}

}

void lagrange(double t,double \*x,double \*y,int n,double \*s)

{

int i,j;

double p;

for (\*s=i=0;i<n;i++)

{

p=1;

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

if (i!=j)

p=p\*(t-x[j])/(x[i]-x[j]);

\*s=\*s+y[i]\*p;

}

}

void lininterp(double t,double \*x,double \*y,int n,double \*s)

{

int i,j;

double a,b;

if (t>x[n-1])

j=n-1;

else

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

if (t<=x[i+1])

{

j=i+1;

break;

}

a=y[j-1]-((y[j]-y[j-1])\*x[j-1])/(x[j]-x[j-1]);

b=(y[j]-y[j-1])/(x[j]-x[j-1]);

\*s=a+b\*t;

}

void koef\_spline(double \*x,double \*y,int n,double \*b,double \*c,double \*d)

{

int i;

double \*h,\*l,\*de,\*lam;

h=new double [n];

l=new double [n];

de=new double [n];

lam=new double [n];

for (i=1;i<n;i++)

{

h[i]=x[i]-x[i-1];

l[i]=(y[i]-y[i-1])/h[i];

}

de[1]=-h[2]/(2\*(h[1]+h[2]));

lam[1]=1.5\*(l[2]-l[1])/(h[1]+h[2]);

for (i=3;i<n;i++)

{

de[i-1]=(-h[i])/(2\*h[i-1]+2\*h[i]+h[i-1]\*de[i-2]);

lam[i-1]=(3\*l[i]-3\*l[i-1]-h[i-1]\*lam[i-2])/(2\*h[i-1]+2\*h[i]+h[i-1]\*de[i-2]);

}

c[n-1]=0;

for (i=(n-1);i>=2;i--)

c[i-1]=de[i-1]\*c[i]+lam[i-1];

for (i=1;i<n;i++)

{

d[i]=(c[i]-c[i-1])/(3\*h[i]);

b[i]=l[i]+(2\*c[i]\*h[i]+h[i]\*c[i-1])/3;

}

}

void spline(double t,double \*x,double \*y,int n,double \*s)

{

int i,j;

double \*a,\*b,\*c,\*d;

a=new double [n];

b=new double [n];

c=new double [n];

d=new double [n];

koef\_spline(x,y,n,b,c,d);

for (i=1;i<n;i++)

a[i]=y[i];

if (t>=x[n-1])

j=n-1;

else

for (i=1;i<=(n-1);i++)

if (t<x[i])

{

j=i;

break;

}

\*s=a[j]+b[j]\*(t-x[j])+c[j]\*pow(t-x[j],2)+d[j]\*pow(t-x[j],3);

}

1. Пишем программу *interpmain.cxx* интерполяции функции с использованием канонического полинома, полинома Ньютона, Лагранжа, а также с использованием линейной и сплайн-интерполяции: **(3)**

#include <iostream>

#include <math.h>

#include <fstream> //подключение библиотеки для работы с файлами

#include "interpolation.h"

using namespace std;

int main()

{

int i,n=0;

double \*x,\*y,s,t,a;

fstream f1;

fstream f2;

f1.open("x.txt",fstream::in|fstream::out); //открываем файл с массивом X

f2.open("y.txt",fstream::in|fstream::out); //открываем файл с массивом Y

while (!f1.eof()) //узнаём количество узловых точек n

{

f1>>a; n++;

}

x=new double [n];

y=new double [n];

f1.seekg(0);

for (i=0;i<n;i++) //считываем массивы X и Y из файлов

{

f1>>x[i]; f2>>y[i];

}

cout<<"Massive X:"<<endl;

for (i=0,cout<<"|";i<n;i++)

if (i==n-1) cout<<x[i];

else cout<<x[i]<<" ";

cout<<"|"<<endl;

cout<<"Massive Y:"<<endl;

for (i=0,cout<<"|";i<n;i++)

if (i==n-1) cout<<y[i];

else cout<<y[i]<<" ";

cout<<"|"<<endl;

cout<<"Enter point of interpolation:"; //Запрос на введение точки интерполяции

cin>>t; cout<<endl;

/\*Вызываем функцию kanon(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout<<endl<<"Kanon interpolation:"<<endl;

kanon(t,x,y,n,&s);

cout<<s<<endl;

/\*Вызываем функцию newton(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout<<endl<<"Newton interpolation:"<<endl;

newton(t,x,y,n,&s);

cout<<s<<endl;

/\*Вызываем функцию lagrange(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout<<endl<<"Lagrange interpolation:"<<endl;

lagrange(t,x,y,n,&s);

cout<<s<<endl;

/\*Вызываем функцию lininterp(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout<<endl<<"Linear interpolation:"<<endl;

lininterp(t,x,y,n,&s);

cout<<s<<endl;

/\*Вызываем функцию spline(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout<<endl<<"Spline interpolation:"<<endl;

spline(t,x,y,n,&s);

cout<<s<<endl;

system("pause");

return 0;

}

Теперь создадим рабочее приложение и библиотеку интерполирования функции.

Для сборки приложения выполним следующие шаги:

* Переходим в командную строку (Win+R, пишем cmd)
* Переходим через командную строку в каталог с нашими файлами *interpolation.h, interpolation.cpp и interpmain.cxx*
* Компилируем файл *interpolation.cpp* следующей командой:

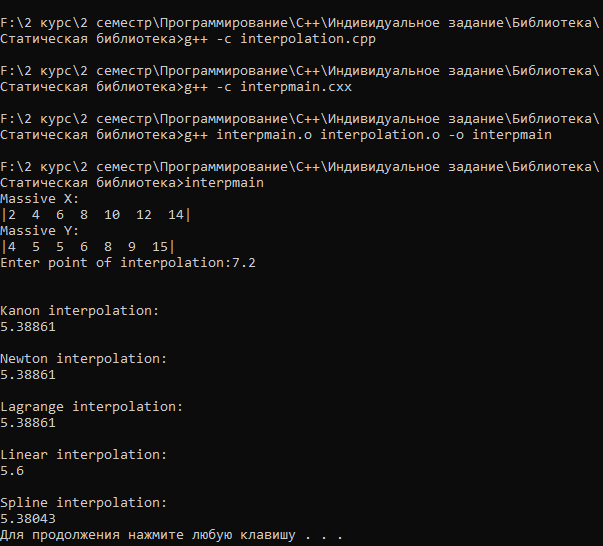
**g++ -c interpolation*.*cpp**

* Компилируем главную программу *interpmain.cxx* следующей командой:
* **g++ -c interpmain.cxx**
* Создаём исполняемый файл с именем *interpmain* из файлов *interpolation.o и interpmain.o:*

**g++ interpmain.o interpolation.o –o interpmain**

* Запустим файл **interpmain** из командной строки

Результат:



Далее создадим саму статическую библиотеку:

* Компилируем файл *interpolation.cpp* следующей командой: (если нет файла *interpolation.o)*

**g++ -c interpolation*.*cpp**

* Используем команду **ar crs** для создания статической библиотеки, где **crs**-ключи команды **ar**: c – создать библиотеку, r – заменить совпадающие объектные файлы внутри библиотеки новыми, s – создать индекс внутри библиотеки:

**ar crs libinterpolation.a ./interpolation.o**

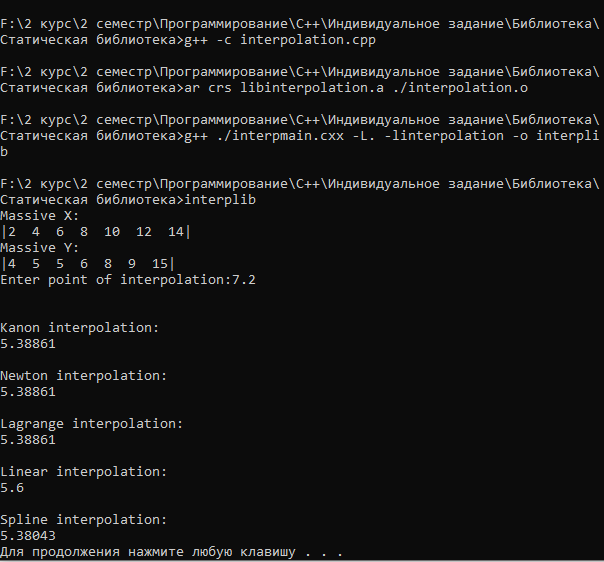
* *libinterpolation.a-*имя библиотеки, которое всегда должно начинаться с *lib* и имеет расширение *.a*
* *interpolation.o*– имя объектного файла, помещаемого в библиотеку

Создали библиотеку *interpolation,* теперь соберём исполняемый файл с ней с помощью следующей команды:

**g++ ./interpmain.cxx -L. -linterpolation -o interplib**

Где ключ -L определяет *расположение библиотеки*, а ключ -l определяет *имя библиотеки,* порядок следования ключей важен!

Результат:



**Создание динамической библиотеки (dll) C++ в компиляторе GCC**

Для создания такой библиотеки нам понадобится также командная строка и компилятор GCC. Создадим 2 файла *interpolation.h* и *interpolation.cpp,* в которые поместим те же коды, описанные выше в кодах (1) и (2).

Откомпилируем файл *interpolation.cpp* с ключом **–fpic** с помощью следующей команды:

**g++ -fpic –c interpolation.cpp**

Объектный файл .o, который мы создавали для статической библиотеки, не подходит для динамических библиотек. Связано это с тем, что все объектные файлы, создаваемые обычным образом, не имеют представления о том в какие адреса памяти будет загружена использующая их программа. Несколько различных программ могут использовать одну библиотеку, и каждая из них располагается в различном адресном пространстве. Поэтому требуется, чтобы переходы в функциях библиотеки использовали не абсолютную адресацию, а относительную. Для этого и предназначен ключ **–fpic.**

Далее создадим файл *interpmain.cpp*, который будет являться нашей главной программой и будет в дальнейшем использовать динамическую библиотеку интерполирования функции. Туда записываем код, описанный выше в коде (3). Компилируем файл *interpmain.cpp* следующей командой:

**g++ -c interpmain.cpp**

Теперь создадим саму dll-библиотеку следующей командой:

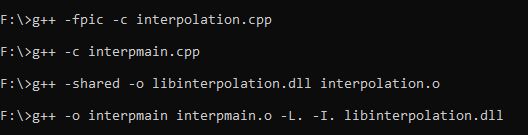
**g++ -shared –o libinterpolation.dll interpolation.o**

Разберём команду подробнее: ключ **–shared** указывает компилятору, что нужно создать именно динамическую библиотеку. После ключа **–o** идёт имя нашей будущей библиотеки**-libinterpolation.dll.** GCC поддерживает прямую линковку с Dll без необходимости создания библиотеки импорта, поэтому библиотеку импорта создавать не нужно.

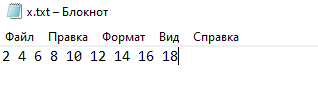
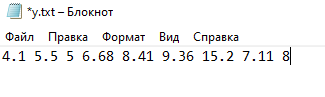
В файле появилась наша Dll-библиотека, с помощью которой мы теперь создадим рабочее приложение. Это можно сделать следующей командой:

**g++ -o interpmain interpmain.o -L. -I. libinterpolation.dll**

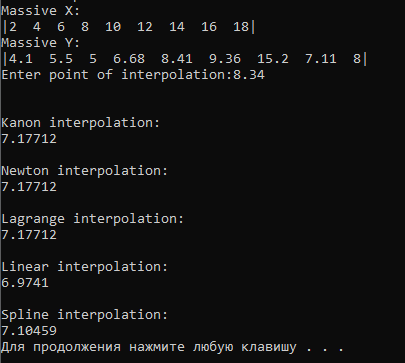
Где *interpmain*-название нашего приложения, ключ **–L** говорит, в какой папке хранится библиотека, ключ **–I** говорит компилятору где искать заголовочный файл (в нашем случае-в текущей директории). Ниже продемонстрированы команды, описанные выше, введённые в командной строке.



Всё готово для использования нашего приложения вместе с библиотекой. Программа интерполяции функции работает с текстовыми файлами, поэтому создаём в папке с программой 2 текстовых файла «x.txt» и «y.txt», которые заполняем нашими любыми данными.



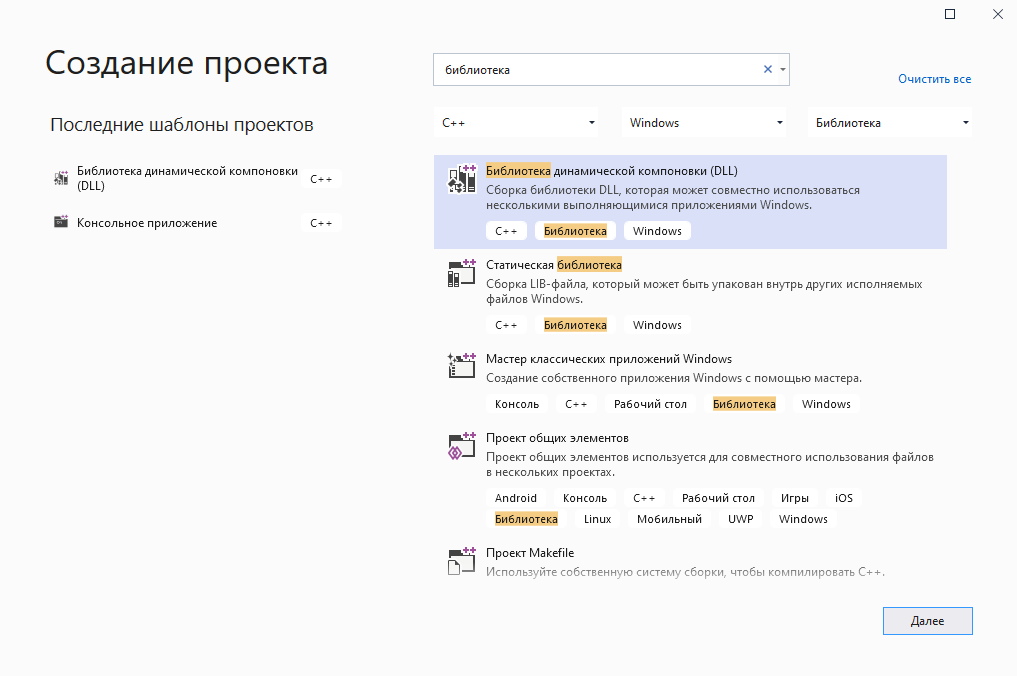
Результат работы программы:



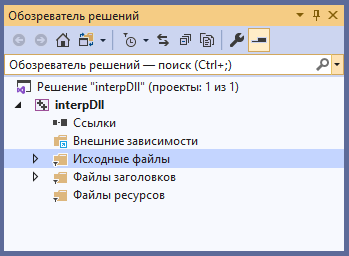
**Динамическая библиотека (dll) C++ в VS**

Создадим теперь динамическую библиотеку интерполирования функции по имеющемуся набору известных значений в программе Microsoft Windows Visual Studio 2019 с подробным объяснением.

1. Запускаем Visual Studio, выбираем пункт «Создание проекта», в поисковой строке пишем «библиотека» и выбираем пункт «Библиотека динамической компоновки (dll)». Далее пишем имя проекта (в моём случае «interpDll»), выбираем нужное расположение и убираем галочку с пункта «Поместить решение и проект в одном каталоге». Нажимаем «Создать».



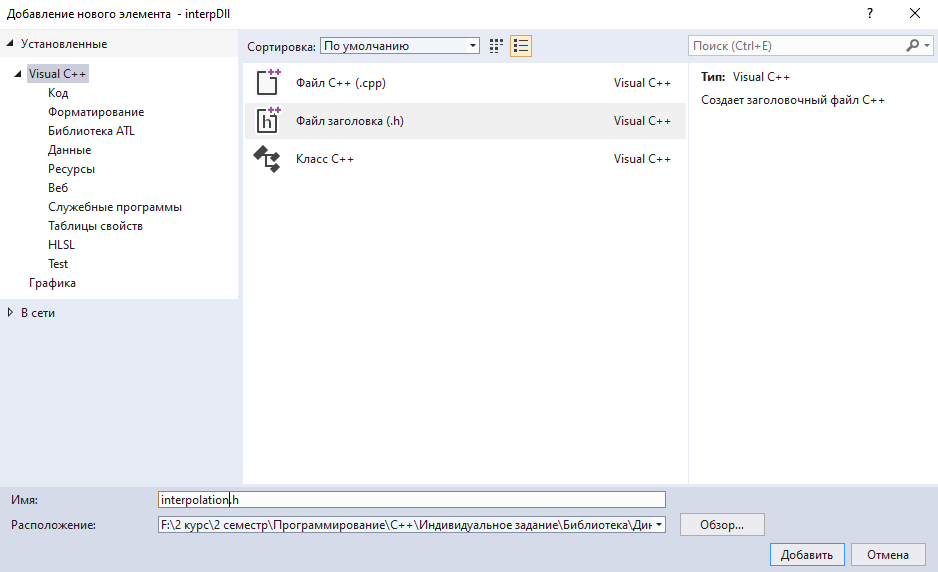
1. Теперь весь наш проект и его составляющие можно видеть в левой окне, в так называемом «Обозревателе решений»



1. После создания проекта заходим во вкладку (Проект→Свойства), выбираем пункт (C/C++→предварительно откомпилированные заголовки) и выбираем «не использовать предварительно откомпилированные заголовки». Файл с именем *pch.h* создаётся в VS автоматически и позволяет несколько ускорить сборку больших проектов, но, так как наша библиотека не является большой, файл *pch.h* просто не нужен. Удаляем файлы *pch.h*, *pch.cpp;*

**Теперь приступим непосредственно к созданию библиотеки:**

1. Создадим новый заголовочный файл с расширение .h (Проект→Добавить новый элемент→Файл заголовка), пусть его имя будет *interpolation.h;*



1. После создания заголовка пишем в него следующий код:

#pragma once

#ifdef INTERPOLATION\_EXPORTS

#define INTERPOLATION\_API \_\_declspec(dllexport)

#else

#define INTERPOLATION\_API \_\_declspec(dllimport)

#endif

/\*Функция построения канонического полинома для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

extern "C" INTERPOLATION\_API void kanon(double t,double \*x,double \*y,int n,double \*s);

/\*Функция построения полинома Ньютона для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

extern "C" INTERPOLATION\_API void newton(double t,double \*x,double \*y,int n,double \*s);

/\*Функция построения полинома Лагранжа для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

extern "C" INTERPOLATION\_API void lagrange(double t,double \*x,double \*y,int n,double \*s);

/\*Функция построения линейного сплайна для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

extern "C" INTERPOLATION\_API void lininterp(double t,double \*x,double \*y,int n,double \*p);

/\*Функция построения кубического сплайна для интеролирования функции

t-точка, в которой нужно найти значение интерполирующей функции

(x,y)-массивы исходных данных функции, n-количество узловых точек

\*s-указатель на переменную типа double (значение функции в точке t после интерполирования)\*/

extern "C" INTERPOLATION\_API void spline(double t,double \*x,double \*y,int n,double \*s);

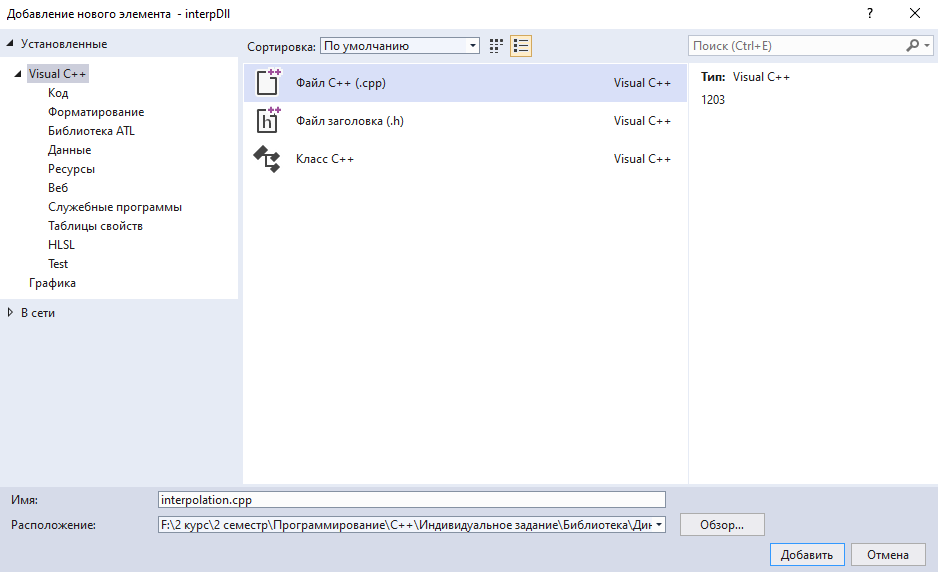
1. Теперь разберём детальнее код, приведённый выше:

В самом начале кода находятся оператора препроцессора. Шаблон нового проекта для библиотеки DLL добавляет **PROJECTNAME\_EXPORTS** к определенным макросам препроцессора.

В нашем случае Visual Studio определяет **INTERPOLATION\_EXPORTS** при сборке проекта библиотеки *interpDll*.

Если определён макрос **INTERPOLATION\_EXPORTS**, то действует следующий макрос **INTERPOLATION\_API,** который устанавливает в объявлениях функций модификаторы ***\_\_declspec(dllexport)*** или ***\_\_declspec(dllimport)*** , которые предназначены для использования приложением функций, хранящихся в dll.

Поскольку C++ имеет множество функций, невозможно просто использовать имя функции в качестве имени символа. Поэтому C++ использует искажение имени, которое преобразует имя функции и другую о ней информацию в строку, обрабатываемую только компилятором и компоновщиком. Но если использовать функцию ***extern “C”*** , то компилятор не делает ничего с именем функции, и тогда доступ к функции может осуществляться непосредственно по её имени.

1. Теперь создадим C++-файл *interpolation.cpp*, в котором будут храниться сами коды всех наших функций: (Проект→Добавить новый элемент→Файл C++)  
   
2. После создания файла записываем в него следующий код:

#include <math.h>

#include "interpolation.h"

int gauss(double\*\* aa, double\* pp, int n, double\* x)

{

int i, j, k, r;

double c, m, max, s;

double\*\* a, \* p;

a = new double\* [n];

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

a[i] = new double[n];

p = new double[n];

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

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

a[i][j] = aa[i][j];

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

p[i] = pp[i];

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

{

max = fabs(a[k][k]);

r = k;

for (i = k + 1; i < n; i++)

if (fabs(a[i][k]) > max)

{

max = fabs(a[i][k]);

r = i;

}

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

{

c = a[k][j];

a[k][j] = a[r][j];

a[r][j] = c;

}

c = p[k];

p[k] = p[r];

p[r] = c;

for (i = k + 1; i < n; i++)

{

for (m = a[i][k] / a[k][k], j = k; j < n; j++)

a[i][j] -= m \* a[k][j];

p[i] -= m \* p[k];

}

}

if (a[n - 1][n - 1] == 0)

if (p[n - 1] == 0)

return -1;

else return -2;

else

{

for (i = n - 1; i >= 0; i--)

{

for (s = 0, j = i + 1; j < n; j++)

s += a[i][j] \* x[j];

x[i] = (p[i] - s) / a[i][i];

}

return 0;

}

}

void kanon(double t, double\* x, double\* y, int n, double\* s)

{

int i, j;

double\*\* c, \* a;

c = new double\* [n];

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

c[i] = new double[n];

a = new double[n];

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

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

c[i][j] = pow(x[i], j);

gauss(c, y, n, a);

for (\*s = i = 0; i < n; i++)

\*s = \*s + a[i] \* pow(t, i);

}

void newton(double t, double\* x, double\* y, int n, double\* s)

{

int i, j;

double\*\* c, \* a, p;

c = new double\* [n];

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

c[i] = new double[n];

a = new double[n];

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

c[i][0] = y[i];

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

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

if (i < j)

c[i][j] = 0;

for (j = 1; j < n; j++)

for (i = 1; i < n; i++)

if (i >= j)

c[i][j] = (c[j - 1][j - 1] - c[i][j - 1]) / (x[j - 1] - x[i]);

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

a[i] = c[i][i];

for (\*s = i = 0; i < n; i++)

{

p = 1;

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

p = p \* (t - x[j]);

\*s = \*s + a[i] \* p;

}

}

void lagrange(double t, double\* x, double\* y, int n, double\* s)

{

int i, j;

double p;

for (\*s = i = 0; i < n; i++)

{

p = 1;

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

if (i != j)

p = p \* (t - x[j]) / (x[i] - x[j]);

\*s = \*s + y[i] \* p;

}

}

void lininterp(double t, double\* x, double\* y, int n, double\* s)

{

int i, j;

double a, b;

if (t > x[n - 1])

j = n - 1;

else

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

if (t <= x[i + 1])

{

j = i + 1;

break;

}

a = y[j - 1] - ((y[j] - y[j - 1]) \* x[j - 1]) / (x[j] - x[j - 1]);

b = (y[j] - y[j - 1]) / (x[j] - x[j - 1]);

\*s = a + b \* t;

}

void koef\_spline(double\* x, double\* y, int n, double\* b, double\* c, double\* d)

{

int i;

double\* h, \* l, \* de, \* lam;

h = new double[n];

l = new double[n];

de = new double[n];

lam = new double[n];

for (i = 1; i < n; i++)

{

h[i] = x[i] - x[i - 1];

l[i] = (y[i] - y[i - 1]) / h[i];

}

de[1] = -h[2] / (2 \* (h[1] + h[2]));

lam[1] = 1.5 \* (l[2] - l[1]) / (h[1] + h[2]);

for (i = 3; i < n; i++)

{

de[i - 1] = (-h[i]) / (2 \* h[i - 1] + 2 \* h[i] + h[i - 1] \* de[i - 2]);

lam[i - 1] = (3 \* l[i] - 3 \* l[i - 1] - h[i - 1] \* lam[i - 2]) / (2 \* h[i - 1] + 2 \* h[i] + h[i - 1] \* de[i - 2]);

}

c[n - 1] = 0;

for (i = (n - 1); i >= 2; i--)

c[i - 1] = de[i - 1] \* c[i] + lam[i - 1];

for (i = 1; i < n; i++)

{

d[i] = (c[i] - c[i - 1]) / (3 \* h[i]);

b[i] = l[i] + (2 \* c[i] \* h[i] + h[i] \* c[i - 1]) / 3;

}

}

void spline(double t, double\* x, double\* y, int n, double\* s)

{

int i, j;

double\* a, \* b, \* c, \* d;

a = new double[n];

b = new double[n];

c = new double[n];

d = new double[n];

koef\_spline(x, y, n, b, c, d);

for (i = 1; i < n; i++)

a[i] = y[i];

if (t >= x[n - 1])

j = n - 1;

else

for (i = 1; i <= (n - 1); i++)

if (t < x[i])

{

j = i;

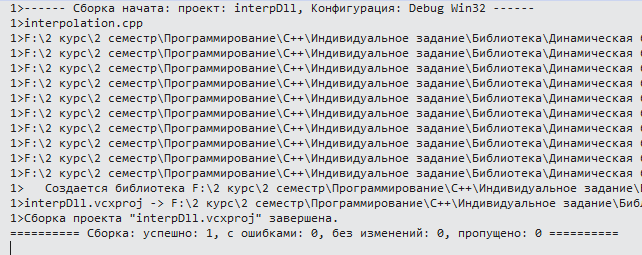
break;

}

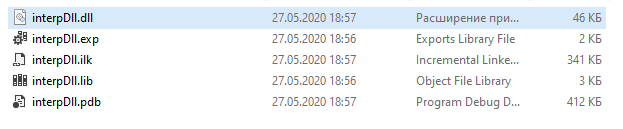
\*s = a[j] + b[j] \* (t - x[j]) + c[j] \* pow(t - x[j], 2) + d[j] \* pow(t - x[j], 3);

}

1. Нужно убедиться, что всё работает, поэтому проведём **сборку** нашей dll-библиотеки. В верхней панели выбираем (Сборка-Собрать решение) и ждём завершения сборки:

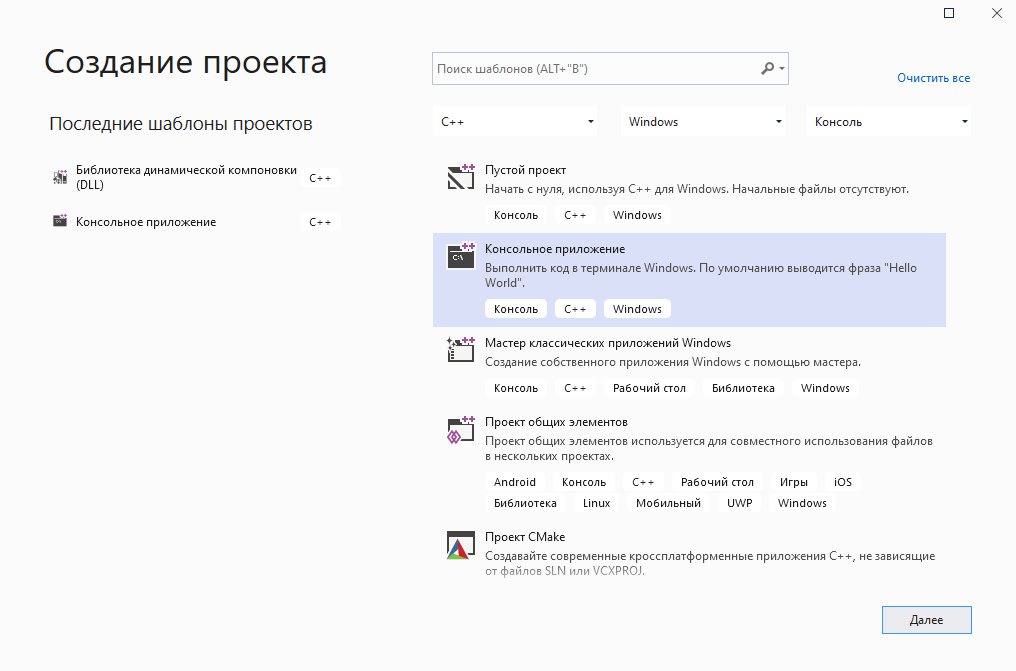


1. После успешной сборки наша dll-библиотека и другие файлы компилятора будут храниться впапке *“..\interpDll\Debug”*

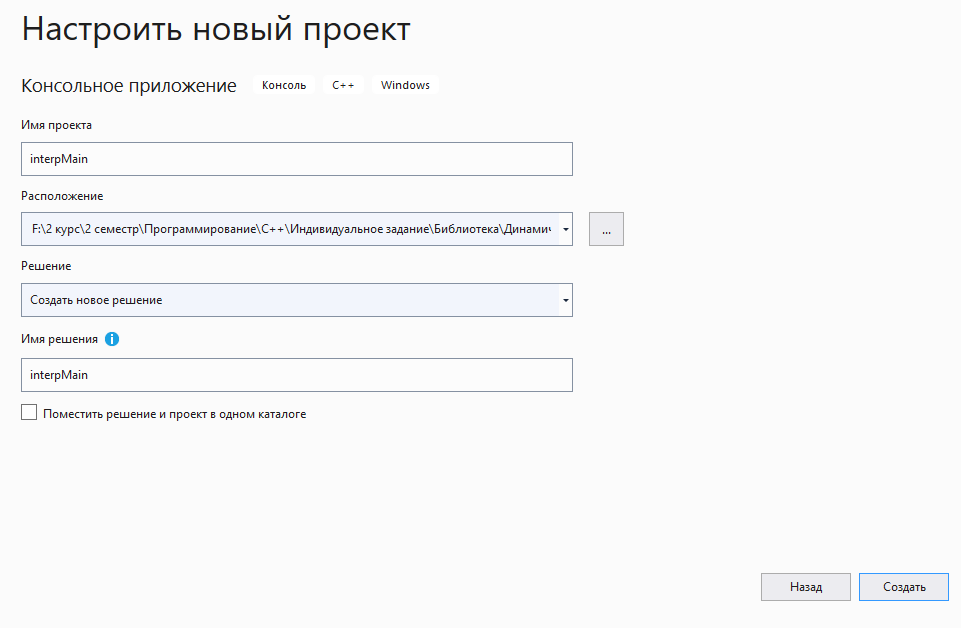


**Создание приложения, использующего Dll-библиотеку**

1. Для создания приложения нужно начать новый проект: (*Файл→Создать→Проект***)**, в открывшемся окне выберем «*Создание проекта***»**.
2. Выбираем язык **C++**, платформа **Windows**, тип проекта-**Консоль.** После поиска выбираем«*Консольное приложение*» и жмём далее.



1. В новом окне также пишем имя проекта, пусть будет «***interpMain***», выбираем нужное расположение, в параметре решения выбираем «*Создать новое решение*» и убираем галочку с пункта «Поместить решение и проект в одном каталоге». Нажимаем «Создать».

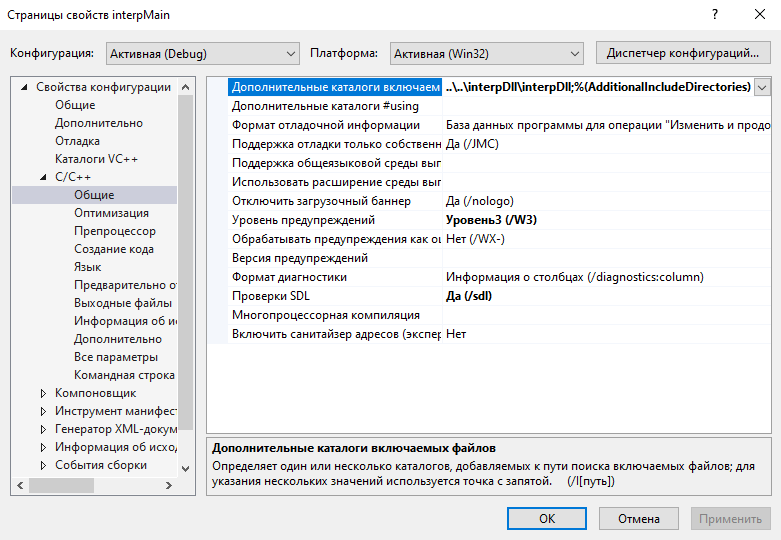


Созданный проект пока не использует dll-библиотеку. Чтобы использовать функции «*interpolation*» в нашем коде, проект должен содержать файл *interpolation.h*.

Замечание: данный файл можно просто добавить в проект приложения, но этот метод подходит для сторонних библиотек. Однако если вы работаете с кодом для библиотеки Dll и приложения одновременно, то файлы заголовков могут оказаться несинхронизированными. Чтобы избежать этой проблемы, нужно задать путь «***Дополнительные каталоги включаемых файлов*»** в проекте, чтобы добавить путь к исходному заголовку.

**Добавление заголовка библиотеки DLL в путь включения**

В верхней панели выбираем (**Проект**→**Свойства**→**C++**→**Общие**). В параметре «***Дополнительные каталоги включаемых файлов*»** нажимаем «*Изменить*», затем «*Создать строку*», куда мы прописываем путь к нашему заголовочному файлу *interpolation.h.* Можно также записать относительный путь к папке с заголовком следующим образом: *..\..\interpDll\interpDll.* После ввода нажимаем «*OK*»

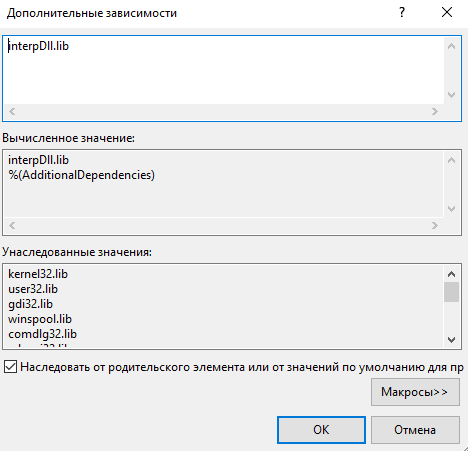


Однако код может быть не скомпилирован, так как мы не указали, что проект зависит от «*interpDll.lib*» и не указали путь к этому файлу.

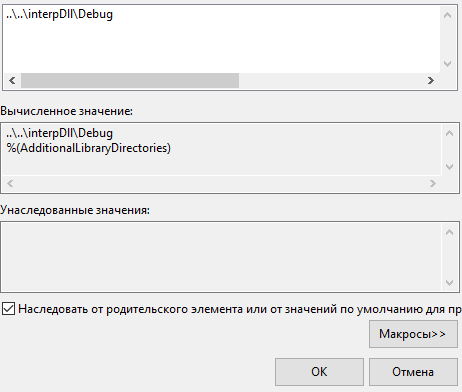
Как и в предыдущем пункте, мы можем просто скопировать файл библиотеки в наш проект, но если и приложение и библиотека будут одновременно редактироваться, то это может привести к изменениям в одной копии, которые не будут отображаться в другой. Чтобы избежать этой проблемы, можно задать свойство «***Дополнительные зависимости*»**, чтобы сообщить системе сборки о том, что проект зависит от «*interpDll.lib»*.

**Добавление библиотеки импорта DLL в проект**

В верхней панели выбираем (**Проект**→**Свойства**→**Компоновщик**→**Ввод**). В параметре «***Дополнительные зависимости*»** нажимаем «*Изменить*», затем добавляем «*interpDll.lib*». После ввода нажимаем «*OK*».



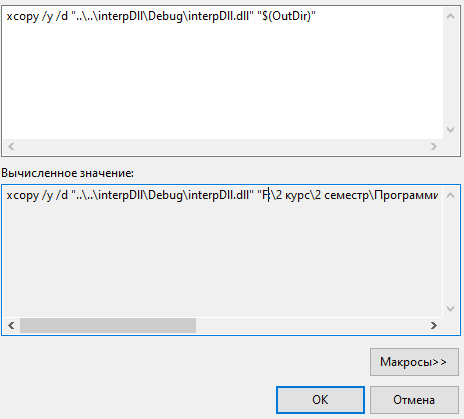
Потом слева выбираем (**Компоновщик**→**Общие**). В параметре «***Дополнительные каталоги библиотек*»** нажимаем «*Изменить*», затем добавляем строку, куда записываем путь к папке с файлом «*interpDll.lib*». Можно также использовать относительный путь: *..\..\ interpDll \Debug;* После ввода нажимаем «*OK*».



Теперь наше приложение можно компилировать и собирать, ему всё равно не хватает нужных файлов. Когда будет загружаться наше приложение, оно будет искать dll-библиотеку «*interpDll.dll*» Чтобы избежать этой проблемы, нужно скопировать библиотеку Dll в папку, в которой находится исполняемый файл приложения, в процессе сборки. Можно добавить **событие после сборки** в проект, чтобы добавить команду копирования библиотеки Dll в папку нашей сборки.

**Копирование библиотеки DLL в событие после сборки**

В верхней панели выбираем (**Проект**→**Свойства**→**События сборки**→**Событие после сборки**). В поле «*командная строка*» нажимаем «изменить» и вводим команду копирования Dll-библиотеки: *xcopy /y /d "..\..\interpDll\Debug\interpDll.dll" "$(OutDir)",* где ключ **/y** означает, нужно ли запрашивать подтверждение перед заменой файлов при копировании, а ключ **/d** позволяет копировать только файлы, измененные не ранее указанной даты. После ввода нажимаем «*OK*».



Теперь в вашем приложении есть все, что нужно для сборки и запуска. В файл «*interpMain.cpp*» записываем следующий код и нажимаем в верхнем меню (**Сборка→Собрать решение**):

#include <iostream>

#include <math.h>

#include <fstream> //подключение библиотеки для работы с файлами

#include "interpolation.h"

using namespace std;

int main()

{

int i, n = 0;

double\* x, \* y, s, t, a;

fstream f1;

fstream f2;

f1.open("x.txt", fstream::in | fstream::out); //открываем файл с массивом X

f2.open("y.txt", fstream::in | fstream::out); //открываем файл с массивом Y

while (!f1.eof()) //узнаём количество узловых точек n

{

f1 >> a; n++;

}

x = new double[n];

y = new double[n];

f1.seekg(0);

for (i = 0; i < n; i++) //считываем массивы X и Y из файлов

{

f1 >> x[i]; f2 >> y[i];

}

cout << "Massive X:" << endl;

for (i = 0, cout << "|"; i < n; i++)

if (i == n - 1) cout << x[i];

else cout << x[i] << " ";

cout << "|" << endl;

cout << "Massive Y:" << endl;

for (i = 0, cout << "|"; i < n; i++)

if (i == n - 1) cout << y[i];

else cout << y[i] << " ";

cout << "|" << endl;

cout << "Enter point of interpolation:"; //Звпрос на введение точки интерполяции

cin >> t; cout << endl;

/\*Вызываем функцию kanon(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout << endl << "Kanon interpolation:" << endl;

kanon(t, x, y, n, &s);

cout << s << endl;

/\*Вызываем функцию newton(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout << endl << "Newton interpolation:" << endl;

newton(t, x, y, n, &s);

cout << s << endl;

/\*Вызываем функцию lagrange(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout << endl << "Lagrange interpolation:" << endl;

lagrange(t, x, y, n, &s);

cout << s << endl;

/\*Вызываем функцию lininterp(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout << endl << "Linear interpolation:" << endl;

lininterp(t, x, y, n, &s);

cout << s << endl;

/\*Вызываем функцию spline(t,x,y,n,&s), где t-точка интерполяции

s-значение интерполированной функции в этой точке\*/

cout << endl << "Spline interpolation:" << endl;

spline(t, x, y, n, &s);

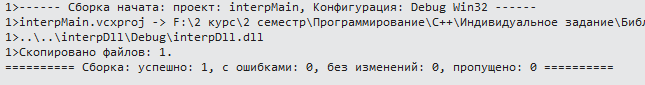
cout << s << endl;

system("pause");

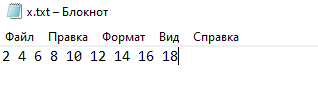
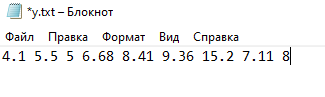
return 0;

}

Сборка прошла успешно



Остался **последний шаг**: программа интерполяции функции работает с текстовыми файлами, поэтому создаём в папке *\\interpMain\\Debug* 2 текстовых файла «x.txt» и «y.txt», которые заполняем нашими любыми данными.



В итоге, результат работы программы следующий:

