Раздельная компиляция

Важной возможностью языка C++ является раздельная компиляция.

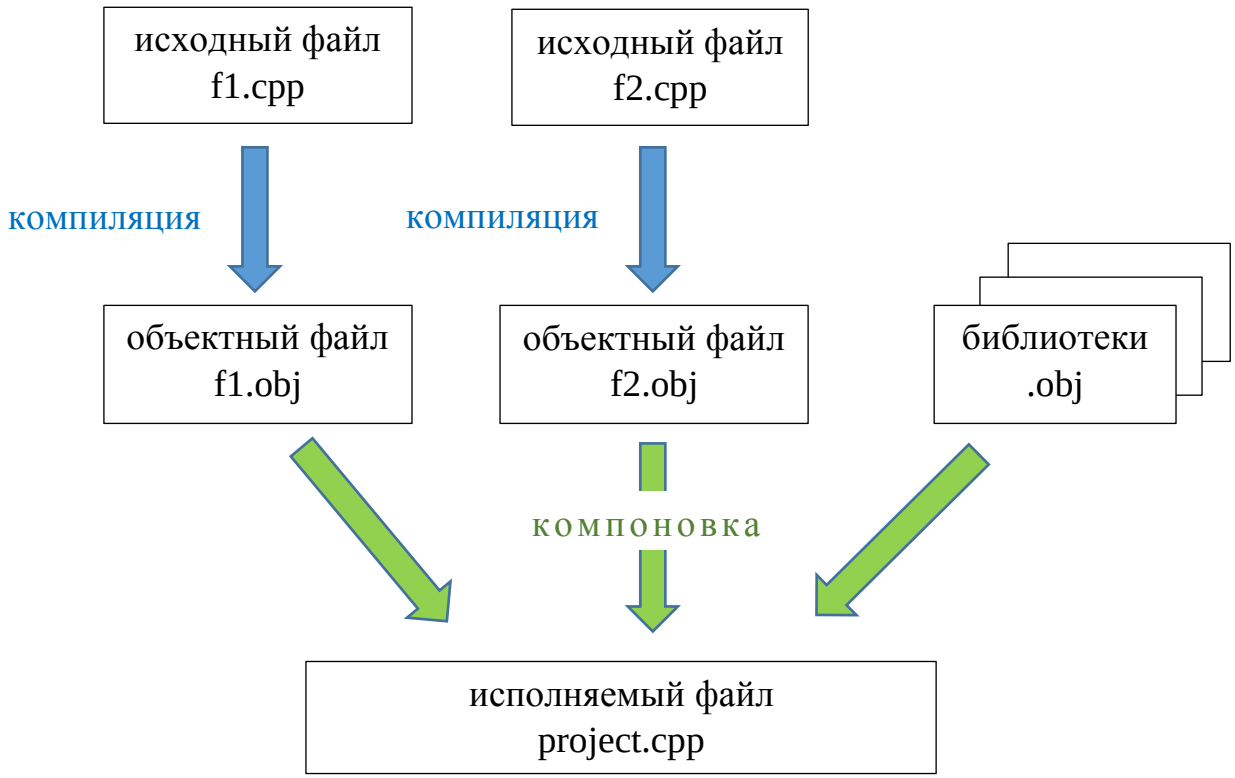
Небольшие программы обычно размещают в одном исходном файле. ***Большие*** же ***программы*** разбивают по смыслу на несколько частей, которые размещают в разных файлах. Для компиляции больших программ используется механизм раздельной компиляции.

Механизм раздельной компиляции состоит в том, что процесс получения программы на машинном языке осуществляется в ***два этапа.***

Первый этап – это компиляция. Отдельные исходные файлы компилируются независимо друг от друга. Результат компиляции одного исходного файла называется объектным модулем. В ОС Windows объектный модуль – это файл с расширением «.obj».

Второй этап называется компоновкой (по-английски linking). Он состоит в сборке всех объектных модулей в готовую программу на машинном языке. Кроме объектных модулей, полученных из исходных файлов программы пользователя, на этом этапе требуются дополнительные файлы, называемые *библиотеками*. Библиотеки содержат машинный код стандартных функций, которые используются в программе пользователя, например, математические, функции ввода-вывода.

Библиотеки обычно представляют собой наборы объектных модулей, объединенных в один файл (под Windows они обычно имеют расширение .lib).



Раздельная компиляция удобна при *небольших модификациях* больших программ, когда имеется много отдельных файлов, и из них изменяется лишь небольшое количество. Тогда нет необходимости перекомпилировать все файлы, а можно перекомпилировать *только те файлы*, которые были *изменены*.

Однако, перекомпоновывать придется все. Смысл объединения файлов программы в проекты, например, в Visual Studio, в том и состоит, что перекомпилироваться будут только изменившиеся файлы, и каждый файл компилируется отдельно.

Пример.

На простом примере рассмотрим, как можно построить проект, состоящий из нескольких файлов. Изначально данная программа, вычисляющая определитель матрицы 2x2, размещена в одном исходном файле:

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

float det (float a, float b, float c, float d)
```

```
{
	return  a*d - b*c;
}

int main(){

	cout << det (1, 2, 3, 4);
	return EXIT_SUCCESS;
}
```

Щелкнув правой кнопкой мыши на имени проекта в обозревателе решений, добавим еще один файл d.cpp, в который переместим определение функции det:

```
float det (float a, float b, float c, float d)
{
	return  a*d - b*c;
}
```

Аналогичным образом добавим в проект заголовочный файл d.h, в котором будет находиться так называемый ***прототип*** функции det (заголовок функции, который заканчивается точкой с запятой):

```
float det (float a, float b, float c, float d);
```

Оставшуюся часть программы, содержащую функцию main, переименуем в main.cpp, добавив в нее заголовочный файл d.h:

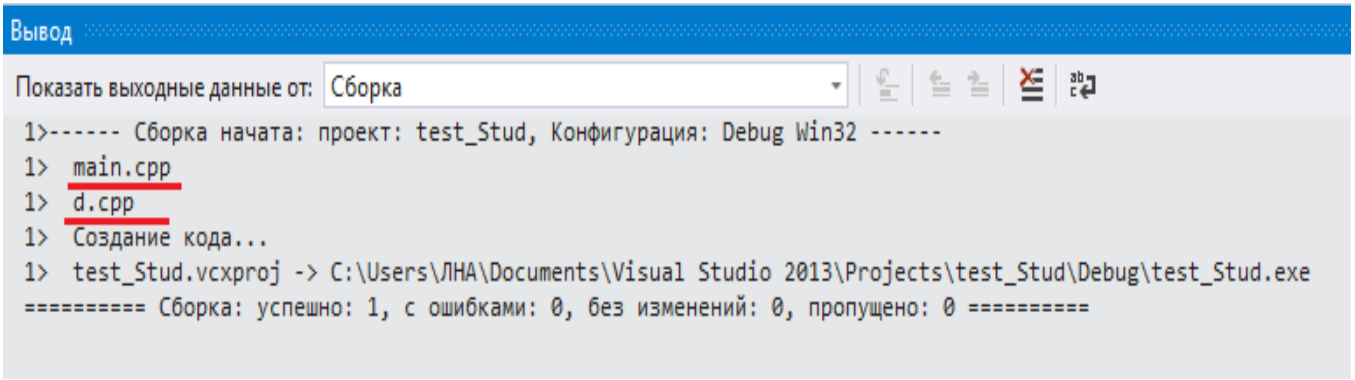
```
#include <iostream>
// <iostream>  -- системный заголовочный файл заключается в угловые скобки

#include "d.h"
// "d.h" -- пользовательский заголовочный файл заключается в двойные кавычки

using namespace std;

 int main()
 {
      cout << det (1, 2, 3, 4);
      return EXIT_SUCCESS;
 }
```

Теперь можно скомпилировать проект. В окне вывода можно увидеть процесс сборки программы: компиляцию файлов main.cpp и d.cpp.



Далее, убедимся в том, что при изменении одного из файлов не требуется перекомпиляция всей программы. Изменим, например, файл main.cpp, добавив новую строчку:

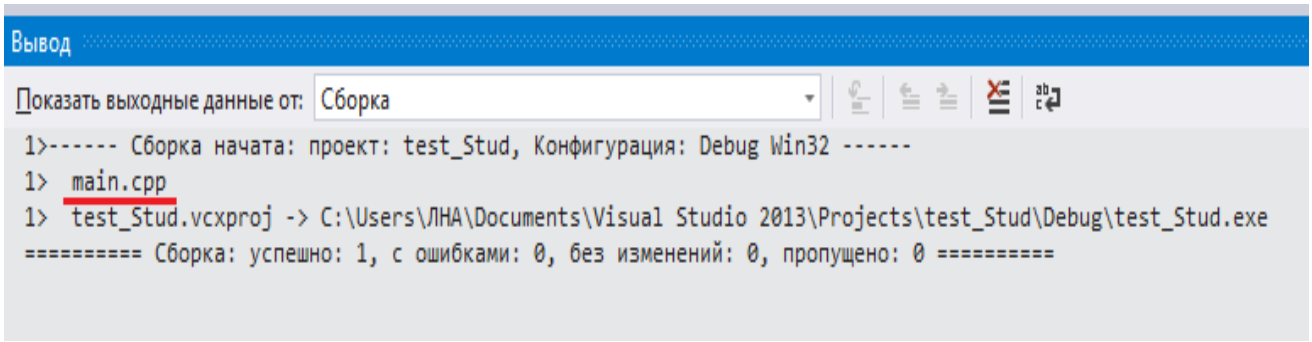
```cpp
#include <iostream>
#include "d.h"

using namespace std;

int main()
{

    cout << det(1, 2, 3, 4);
    cout << endl;      // добавили строчку
    return EXIT_SUCCESS;
}
```

Запустим программу заново. В окне вывода видим, что скомпилировался только файл main.cpp, который был изменен. Файл d.cpp заново компиляции не подвергался:



Аналогичным образом, можно внести изменения в файл d.cpp и увидеть, что только этот файл и был скомпилирован вновь еще раз.

Задание.

0. Проделать все указанные действия для разобранного выше примера.

1. Используя следующую программу

```cpp
#include <iostream>
using namespace std;

int min(int a, int b)
```

```
{
     return a < b ? a : b;
}

int min(int a, int b, int c)
{
     return min (min(a,b), c);
}


int main()
{
     cout << min(2, 4, 3);
     return  EXIT_SUCCESS;
}
```

постройте проект, состоящий из трех файлов. Скомпилируйте проект. Последовательно измените каждый из трех файлов, анализируя результат компиляции в окне вывода.

2. Преобразовать программу вычисления в комплексных числах, выделив все функции кроме main в отдельный файл “complex.cpp”, а прототипы этих функций в файл “complex.h”, который надо будет включить в файл “main.cpp”. Скомпилировать проект и убедиться в работоспособности данного подхода.

```
#include <iostream>
#include <cmath>

using namespace std;

void print(double r, double i)
{
     cout << r << "+" << i << "i";
     cout << endl;
}

void input(double &r, double &i)
{
     cin >> r >> i;
}

double abs(double r, double i)
{
     return sqrt(r*r + i*i);
}
```

```
void add(double r1, double i1,
     double r2, double i2,
     double &r, double &i)
{
     r = r1 + r2;
     i = i1 + i2;
}

void mul(double r1, double i1,
     double r2, double i2,
     double &r, double &i)
{
     r = r1*r2 - i1*i2;
     i = r1*i2 + r2*i1;
}

void conj(double r1, double i1, double &r, double &i)
{
     r = r1;
     i = -i1;
}

void div(double r1, double i1, double t, double &r, double
&i)
{
     r = r1 / t;
     i = i1 / t;
}

void div(double r1, double i1,
     double r2, double i2,
     double &r, double &i)
{
     double tr, ti, nr, ni, dr, di;
     conj(r2, i2, tr, ti);
     mul(r1, i1, tr, ti, nr, ni);
     mul(r2, i2, tr, ti, dr, di);
     div(nr, ni, dr, r, i);
}


int main()
   {
```

```
    setlocale(LC_ALL, "Russian");
    double ra, ia, rb, ib, r, i;

    cout << "Введите комплексные числа: ";
    input(ra, ia);
    input(rb, ib);

    cout << "Сложение: ";
    add(ra, ia, rb, ib, r, i);
    print(r, i);

    cout << "Произведение: ";
    mul(ra, ia, rb, ib, r, i);
    print(r, i);

    cout << "Деление: ";
    div(ra, ia, rb, ib, r, i);
    print(r, i);

    return EXIT_SUCCESS;
}
```

3. Написать функцию f(x) = $\sin^2 x$, вычисляющую приближенное значение определенного интеграла от нее методом средних прямоугольников (по формуле

$$\int_a^b f(x)dx \approx \frac{b-a}{n} \sum_{i=1}^{n} f\left(a + \left(i - \frac{1}{2}\right)\frac{b-a}{n}\right)$$

а также функцию main, выводящую на экран приближенное значение интеграла и разместить их в трех разных файлах. Скомпилировать проект и посмотреть, как влияет рост n (натуральный параметр) на точность вычислений при a=b=1. Изменить функцию f(x) на tg x и убедиться, что перекомпилируется только файл, содержащий определение f. Изменить в процедуре вычисления интеграла формулу на формулу трапеций

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\left[\frac{f(a)}{2} + \sum_{i=1}^{n-1} f(a+ih) + \frac{f(b)}{2}\right]$$

и убедиться, что перекомпилируется только файл, содержащий функцию приближенного вычисления интеграла.