

Модули

# Причины создания модулей

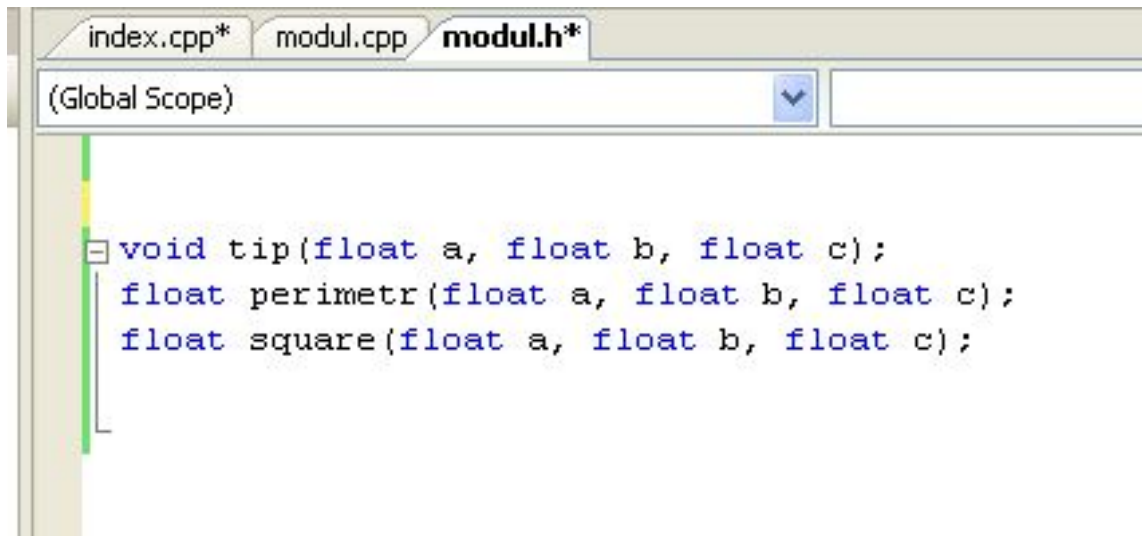
- Длинный программный код
- После исправления ошибки необходимо перекомпилировать программу заново.  
Для больших программ – время большое.

# Решение проблемы

- Разбиение решения сложной задачи на подзадачи.
- Реализация решения каждой подзадачи в виде функции.
- Выделение однотипных подзадач в рамках разных содержательных задач.
- Создание библиотек стандартных задач

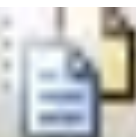
# Создание модулей

- В отдельном файле собирают объявления функций



The screenshot shows a code editor window with three tabs: index.cpp\*, modul.cpp, and modul.h\*. The active tab is modul.h\*. The editor displays the following C++ code:

```
(Global Scope)
void tip(float a, float b, float c);
float perimetr(float a, float b, float c);
float square(float a, float b, float c);
```



Solution 'modul' (1 project)



**modul**



Header Files



modul.h



Resource Files

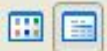
Add New Item - modul



Categories:

- Visual C++
  - UI
  - Code**
  - Data
  - Resource
  - Web
  - Utility
  - Property Sheets

Templates:



**Visual Studio installed templates**

- C++ File (.cpp)
- Header File (.h)
- Module-Definition File (.def)
- Midl File (.idl)
- Component Class
- Installer Class

**My Templates**

- Search Online Templates...

Creates a C++ header file

Name:

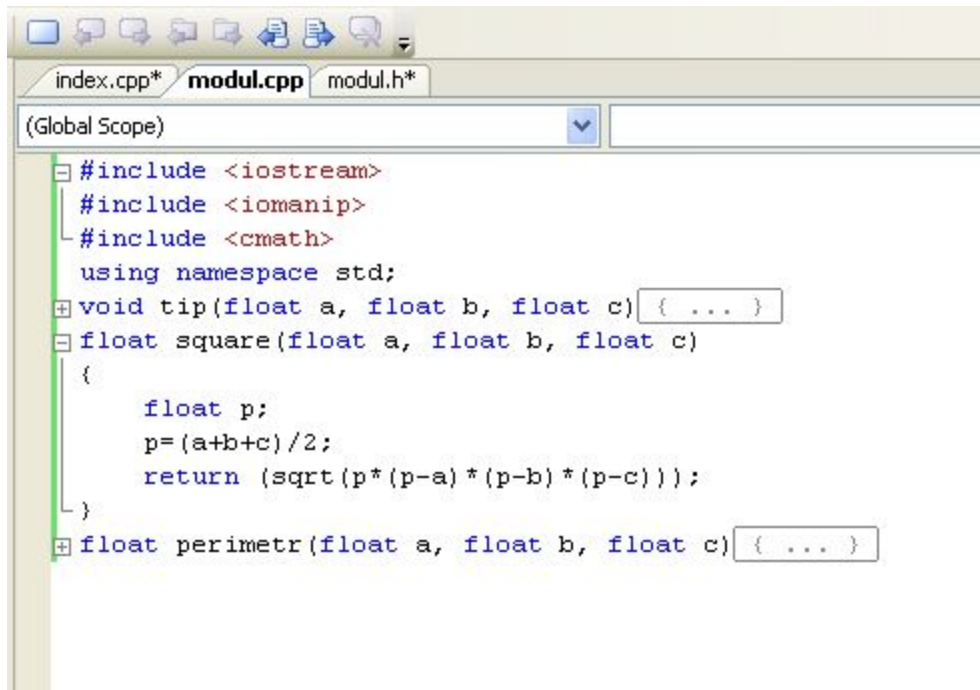
modul

Location:

d:\temp\modul\modul

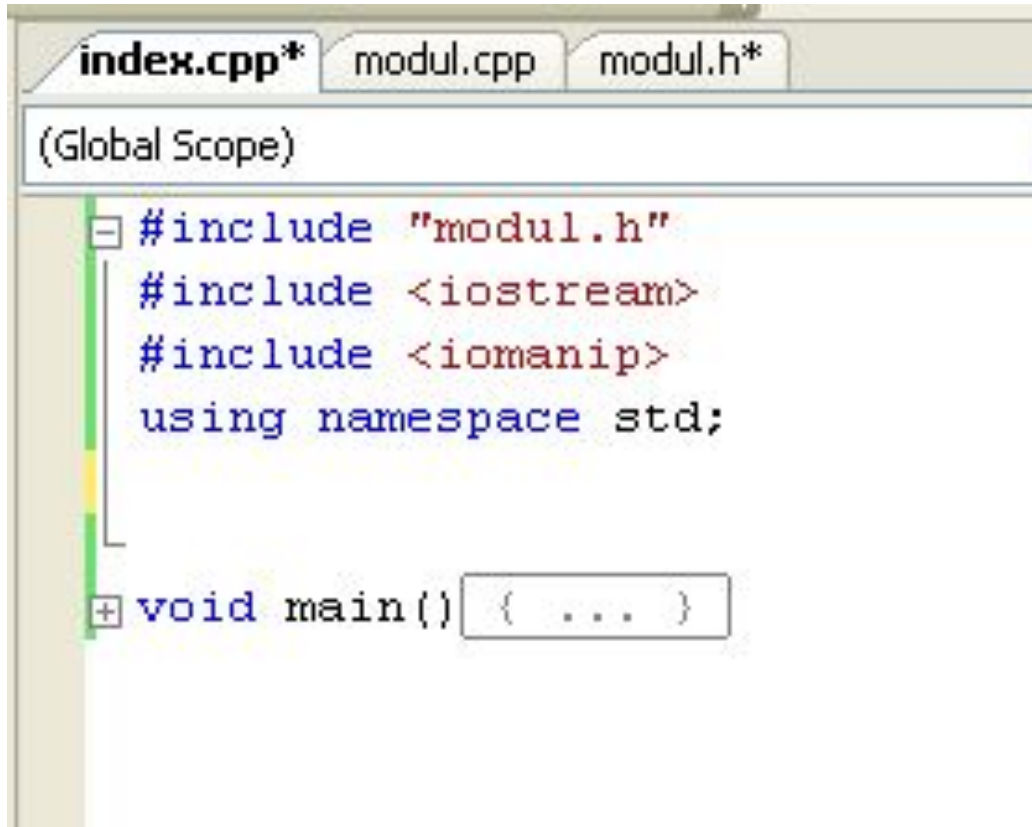
Browse...

- В отдельном файле собираем описание функций



```
index.cpp* modul.cpp modul.h*
(Global Scope)
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
void tip(float a, float b, float c) { ... }
float square(float a, float b, float c)
{
    float p;
    p=(a+b+c)/2;
    return (sqrt(p*(p-a)*(p-b)*(p-c)));
}
float perimetr(float a, float b, float c) { ... }
```

- Создаем главную функцию



The image shows a code editor window with three tabs: `index.cpp*`, `modul.cpp`, and `modul.h*`. The active tab is `index.cpp*`. Below the tabs, the text "(Global Scope)" is displayed. The code content is as follows:

```
#include "modul.h"  
#include <iostream>  
#include <iomanip>  
using namespace std;  
  
void main() { ... }
```



C:\WINDOWS\system32\cmd.exe

Введите стороны треугольника

3 4 5

Выберите номер пункта:

1 . Периметр

2 . Площадь

3 . Тип

4 . Выход

3

Прямоугольный

Выберите номер пункта:

1 . Периметр

2 . Площадь

3 . Тип

4 . Выход

2

Площадь = 6 . 00

Выберите номер пункта:

1 . Периметр

2 . Площадь

3 . Тип

4 . Выход

The image shows a screenshot of the Visual Studio IDE. The top toolbar includes icons for file operations and a 'Debug' button. Below the toolbar is the Solution Explorer, which displays a solution named 'modul' containing one project. The project 'modul' has three folders: 'Header Files' containing 'modul.h', 'Resource Files', and 'Source Files' containing 'index.cpp' and 'modul.cpp'. The 'modul.h' and 'modul.cpp' files are underlined in red. The main editor window shows the code for 'index.cpp' with the following content:

```
(Global Scope)

#include "modul.h"
#include <iostream>
#include <iomanip>
using namespace std;

void main() { ... }
```

# Что будет выдаваться на экран?

```
#include <iostream>;
using namespace std;
void privet()
{
    cout<<"Hi!";
    privet();
}
void main()
{
    privet();
}
```





# Типы рекурсии

- ***Рекурсия*** (от латинского *recursio* – возвращение) — это такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения ее операторов обращается сама к себе.

- При каждом рекурсивном вызове информация о нем сохраняется в специальной области памяти, называемой ***стеком***.
- В стеке записываются значения локальных переменных, параметров функции и адрес точки возврата.
- Какой-либо локальной переменной *A* на разных уровнях рекурсии будут соответствовать разные ячейки памяти, которые могут иметь разные значения.



# Основные понятия

- Максимальное количество вызовов рекурсивной подпрограммы, которое одновременно может находиться в памяти компьютера, называется ***глубиной рекурсии***.

1) Выполнение действий *на рекурсивном спуске*.

тип **рес(параметры)**

{

*<действия на входе в рекурсию>;*

**If** *<проверка условия>* **рес(параметры);**

**[else S;]**

}

```
#include <iostream>
#include <iomanip>
using namespace std;
int s=1;
int fact(int k)
{
if (k>=1)
{
s=s*k;
fact(k-1);
}
return s;
}
```

```
void main()
{
int n;
cin>>n;
s=fact(n);
cout<<s;
}
```

# Ввели 4

Уровень	Спуск			Возврат
	<i>K</i>	<i>S</i>	<i>Fact</i>	
0	4	1	Fact(4)	24
1	4	4	Fact(3)	24
2	3	12	Fact(2)	24
3	2	24	Fact(1)	24
4	1	24	Fact(0)	24
5	0			

2) Выполнение действий *на рекурсивном возврате*.

тип **Rec(параметры);**

**{**

**If** *<проверка условия>* **Rec(параметры);**

**[else S1];**

*<действия на выходе из рекурсии>;*

**}**

```
#include <iostream>
#include <iomanip>
using namespace std;
int fact(int k)
{
    if (k>=1)
        return (fact(k-1)*k);
    else return (1);
}
```

```
void main()
{
    int n;
    cin>>n;
    cout<<fact(n);
}
```

<i>Уровень</i>	<i>Спуск</i>		<i>Возврат</i>
	<i>K</i>	<i>Fact</i>	
0	4	Fact(4)	24
1	4	Fact(3)*4	24
2	3	Fact(2)*3	12
3	2	Fact(1)*2	4
4	1	Fact(0)*2	2
5	0	1	

3) Выполнение действий *на рекурсивном спуске и на рекурсивном возврате.*

тип Рес (параметры);

```
{  
  <действия на входе в рекурсию>;  
  If <условие> Рес(параметры);  
  <действия на выходе из рекурсии>  
}
```

или

тип Рес(параметры);

```
{  
  If <условие>  
  {  
    <действия на входе в рекурсию>;  
    Рес;  
    <действия на выходе из рекурсии>  
  }  
}
```



# Написать рекурсивную функцию нахождения $n$ -го числа Фибонначи

```
#include <iostream>
using namespace std;
int Fib(int n);
int main()
{
    int n ;
    cin >> n;
    int y = Fib(n);
    cout << "Fib(" << n << ")="
    << y<< endl;
    return 0;
}
```

```
int Fib(int n)
{
    if(n <= 2)
        return 1;
    else
        return Fib(n-1) +
        Fib(n-2);
}
```

Написать рекурсивную функцию  
нахождения цифр числа.

```
#include <iostream>
using namespace std;
int cifra(int n)
{
    int s;
    if (n==0) s=0;
    else
s=cifra(n/10)+n%10;
    return s;
}
```

```
int main()
{
    int n ;
    cin >> n;
    int y = cifra(n);
    cout << "Sum="<< y<<
endl;
    return 0;
}
```

# Косвенная рекурсия

```
#include <iostream>
using namespace std;
int A;
void Rec2 (int& Y);
void Rec1 (int& X)
{
    X= X-1;
    if (X>0)
    {
        cout<<X<<"\n";
        Rec2(X);
    }
}
```

```
void Rec2 (int& Y)
{
    Y= Y /2;
    if (Y>2)
    {
        cout<<Y<<"\n";
        Rec1(Y);
    }
}
```

```
void main()
{
    A= 15;
    Rec1(A);
    cout<<A<<"\n";
}
```

$$X=14$$

$$Y=7$$

$$X=6$$

$$Y=3$$

$$X=2$$

$$A=1$$

- Рекурсивные версии программ, как правило, гораздо короче и нагляднее.
- Использование рекурсии увеличивает время исполнения программы и зачастую требует значительного объёма памяти для хранения копий подпрограммы на рекурсивном спуске.
- Разумно заменять рекурсивные алгоритмы на итеративные.
- Любой рекурсивный алгоритм можно преобразовать в эквивалентный итеративный (то есть использующий циклические конструкции).