FUNCTIONS

INTRODUCTION

- Functions are like building blocks
- They allow complicated programs to be divided into manageable pieces
- Some advantages of functions:
  - A programmer can focus on just that part of the program and construct it, debug it, and perfect it
  - Different people can work on different functions simultaneously
  - Can be re-used (even in different programs)
  - Enhance program readability

INTRODUCTION (CONTINUED)

- Functions
  - Called modules
  - Like miniature programs
  - Can be put together to form a larger program

FUNCTIONS IN C++

- Experience has shown that the best way to develop and maintain large programs is to construct it from smaller pieces(Modules)
- This technique Called "**Divide and Conquer**"

Bad Development Approach

```
main()
{
    -----
    -----
    -----
    -----
    .
    .
    .
    ----
    -----
    -----
Return 0;
}
```

- Easer To
- ✔Design
- ✔Build
- ✔Debug
- ✔Extend
- ✔Modify
- ✔Understand
- ✔Reuse
- ✔Better Organization

Wise Development Approach

```
main()
{
  -----
    ----
}
```

```
function f1()
{
    ---
    ---
}
```

```
function f2()
{
    ---
    ---
}
```

PREDEFINED FUNCTIONS

- In algebra, a function is defined as a rule or correspondence between values, called the function's arguments, and the unique value of the function associated with the arguments
  - If `f(x) = 2x + 5`, then `f(1) = 7`, `f(2) = 9`, and `f(3) = 11`
    - `1`, `2`, and `3` are arguments
    - `7`, `9`, and `11` are the corresponding values

PREDEFINED FUNCTIONS (CONTINUED)

- Some of the predefined mathematical functions are:

```
sqrt(x)
pow(x, y)
floor(x)
```

- Predefined functions are organized into separate libraries
- I/O functions are in `iostream` header
- Math functions are in `cmath` header

PREDEFINED FUNCTIONS (CONTINUED)

- `pow(x,y)` calculates x^y
  - `pow(2, 3) = 8.0`
  - Returns a value of type `double`
  - `x` and `y` are the parameters (or arguments)
    - The function has two parameters
- `sqrt(x)` calculates the nonnegative square root of `x`, for `x >= 0.0`
  - `sqrt(2.25)` is `1.5`
  - Type `double`

PREDEFINED FUNCTIONS (CONTINUED)

- The `floor` function `floor(x)` calculates largest whole number not greater than `x`
  - `floor(48.79)` is `48.0`
  - Type `double`
  - Has only one parameter

PREDEFINED FUNCTIONS CONTINUED

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cstdlib>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int`	`int`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than `x`: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle `x`: `cos(0.0) = 1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where `e = 2.718`: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`

Function	Header File	Purpose	Parameter(s) Type	Result
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than x: `floor(45.67) = 45.00`	`double`	`double`
`pow(x, y)`	`<cmath>`	Returns x^y; If `x` is negative, `y` must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of `x` if `x` is uppercase; otherwise, returns `x`	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of `x` if `x` is lowercase; otherwise, returns `x`	`int`	`int`

C++ USER-DEFINED FUNCTION TYPES

- Function with no argument and no return value
- Function with no argument but return value
- Function with argument but no return value
- Function with argument and return value

FUNCTION DEFINITION

- Function definition format

```
functionType functionName(formal parameter list)
{
    statements
}
```

  - Function-name: any valid identifier
  - Return-value-type: data type of the result (default **`int`**)
    - **`void`** – indicates that the function returns nothing
  - Parameter-list: comma separated list, declares parameters
    - A type must be listed explicitly for each parameter unless, the parameter is of type **`int`**

FUNCTION PROTOTYPE

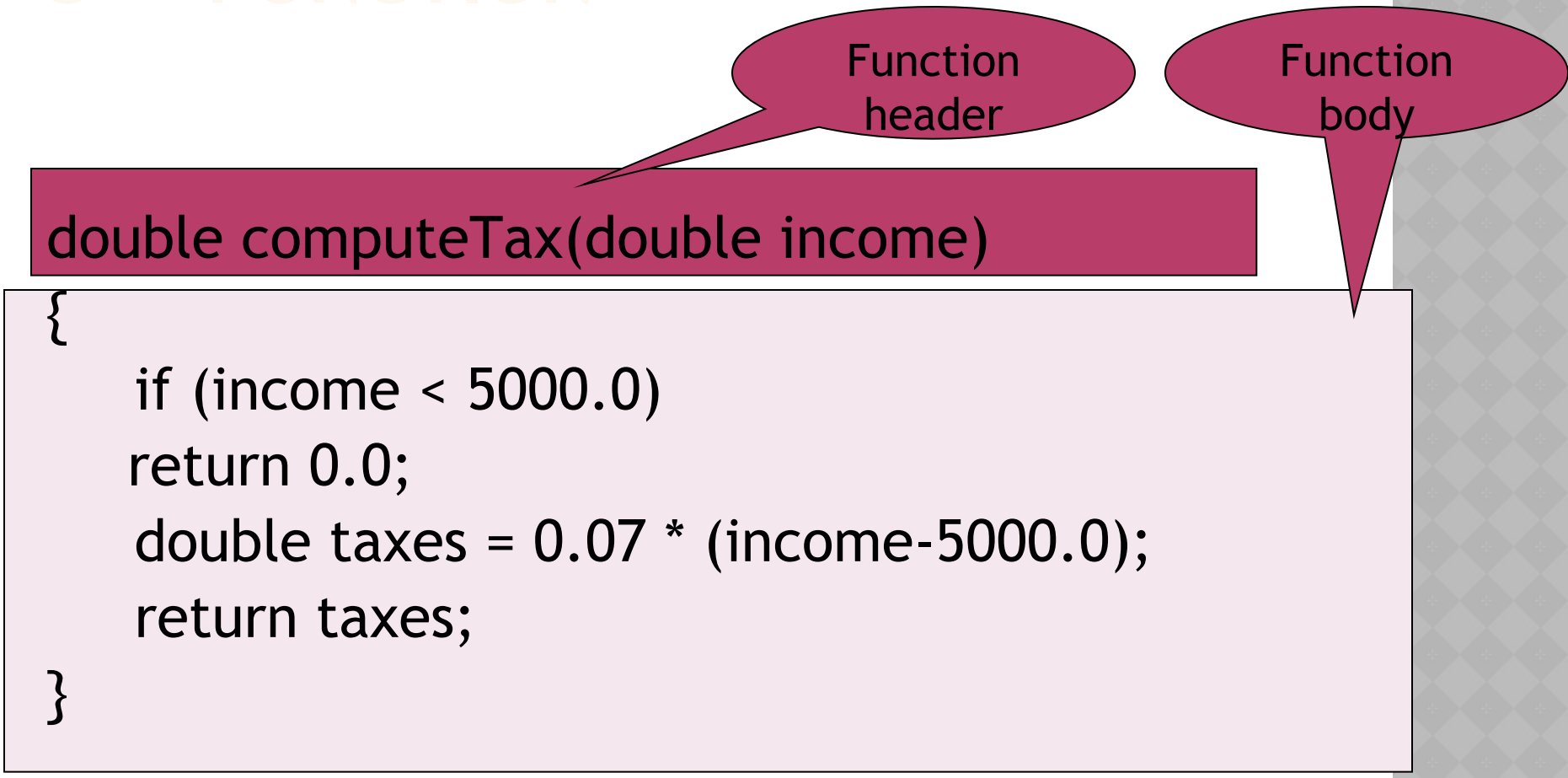
- The function prototype declares the input and output parameters of the function.

- The function prototype has the following syntax:

```
<type> <function name>(<type list>);
```

- Example: A function that returns the absolute value of an integer is: `int absolute(int);`

EXAMPLE OF USER-DEFINED C++ FUNCTION



- Functions
  - Modularize a program
  - Software reusability
    - Call function multiple times
- Local variables
  - Known only in the function in which they are defined
  - All variables declared in function definitions are local variables
- Parameters
  - Local variables passed to function when called
  - Provide outside information

FUNCTION CALLING

- Functions called by writing
  - functionName (argument);
  - or
  - functionName(argument1, argument2, …);
- Example
  - `cout << sqrt( 900.0 );`
  - sqrt (square root) function
  - The preceding statement would print 30
  - All functions in math library return a **double**
- Function Arguments can be:
  - Constant `sqrt(9);`
  - Variable `sqrt(x);`
  - Expression `sqrt( x*9 + y) ;`

```
sqrt( sqrt(x) ) ;
```

FACTORIAL FUNCTION

- Write a function to compute n!

```
int factorial( int n)
{
    int product=1;
    for (int i=1; i<=n; i++)
       product *= i;
    return product;
}
```

```
double larger(double x, double y)
{
    double max;

    if (x >= y)
        max = x;
    else
        max = y;

    return max;
}
```

You can also write this function as follows:

```
double larger(double x, double y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

```
double larger(double x, double y)
{
    if (x >= y)
        return x;

    return y;
}
```

VOID FUNCTION TAKES ARGUMENTS

If the Function does not RETURN result, it is called void Function

```
#include<iostream>
using namespace std;
void add2Nums(int,int);
main()
{   int a, b;
    cout<<"enter two Number:";
    cin >>a >> b;
    add2Nums(a, b);

}
void add2Nums(int x, int y)
{
    cout<< x<< "+" << y << "=" << x+y;
}
```

VOID FUNCTION TAKE NO ARGUMENTS

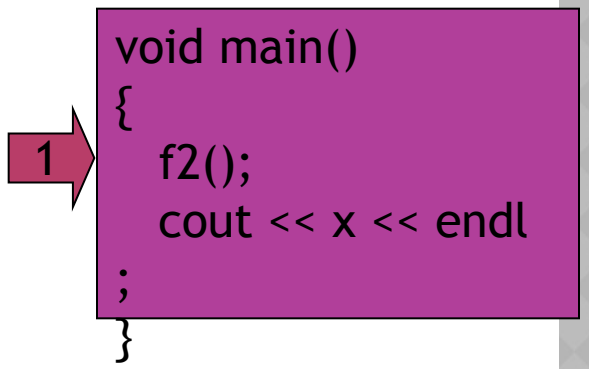
If the function Does Not Take Arguments specify this with EMPTY-LIST OR write `void` inside

```
#include<iostream>
using namespace std;
    void funA();
    void funB(void);
    main()
    {
       funA();
       funB();

    }
   void  funA()
   {
       cout << "Function-A takes no arguments\n";
    }
    void funB()
    {
       cout << "Also Function-B takes No arguments\n";
    }
```

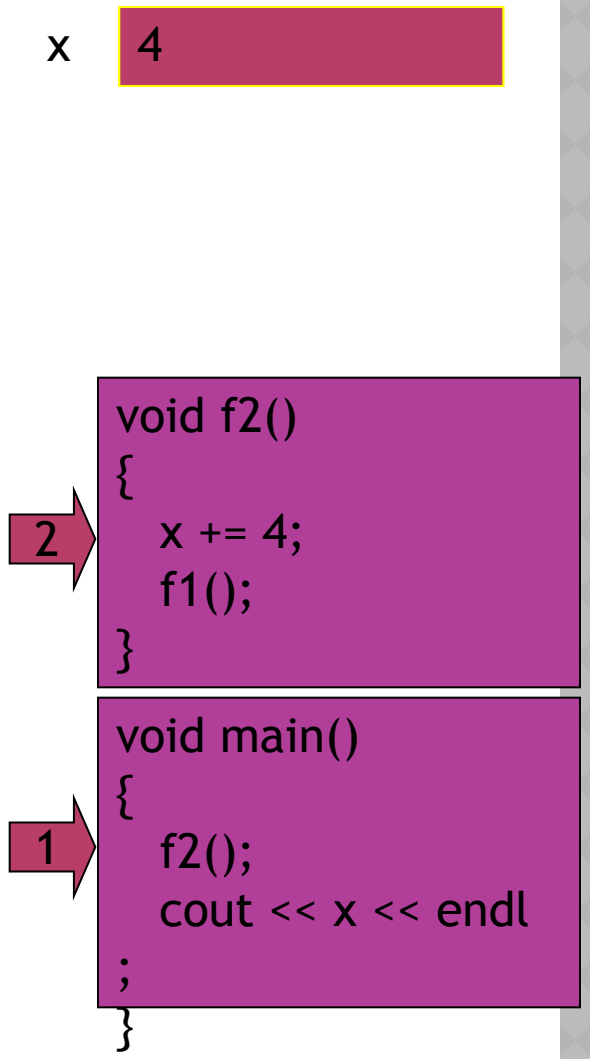
USING GLOBAL VARIABLES

```
#include<iostream>
int x = 0;
void f1() {  x++; }
void f2() { x+=4;
   f1(); }
void main()
{
   f2();
   cout << x << endl;
}
```



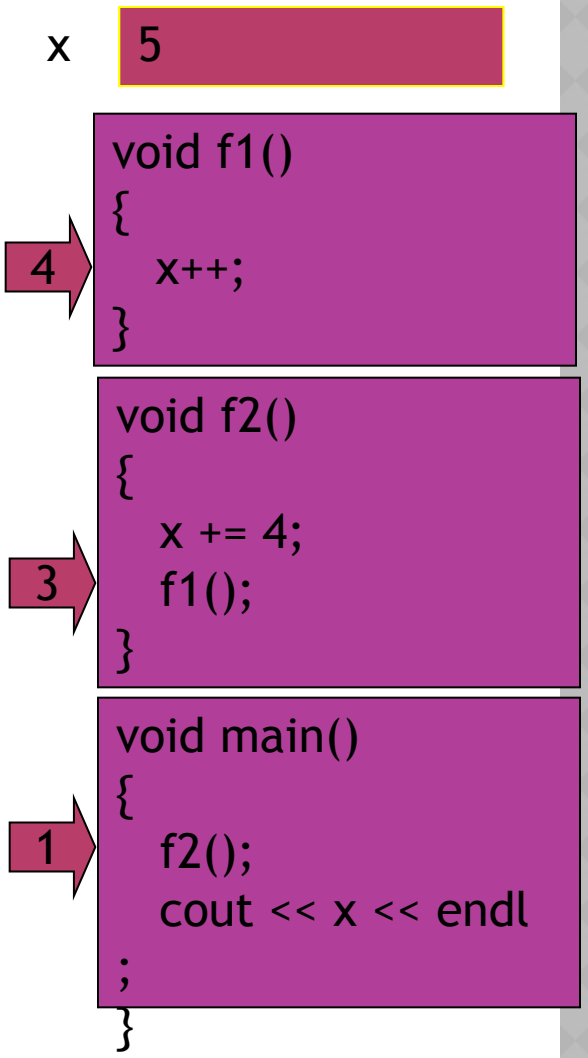
USING GLOBAL VARIABLES

```
#include
   <iostream.h>
int x = 0;
void f1() {  x++; }
void f2() { x+=4;
   f1(); }
void main()
{
   f2();
   cout << x << endl;
}
```



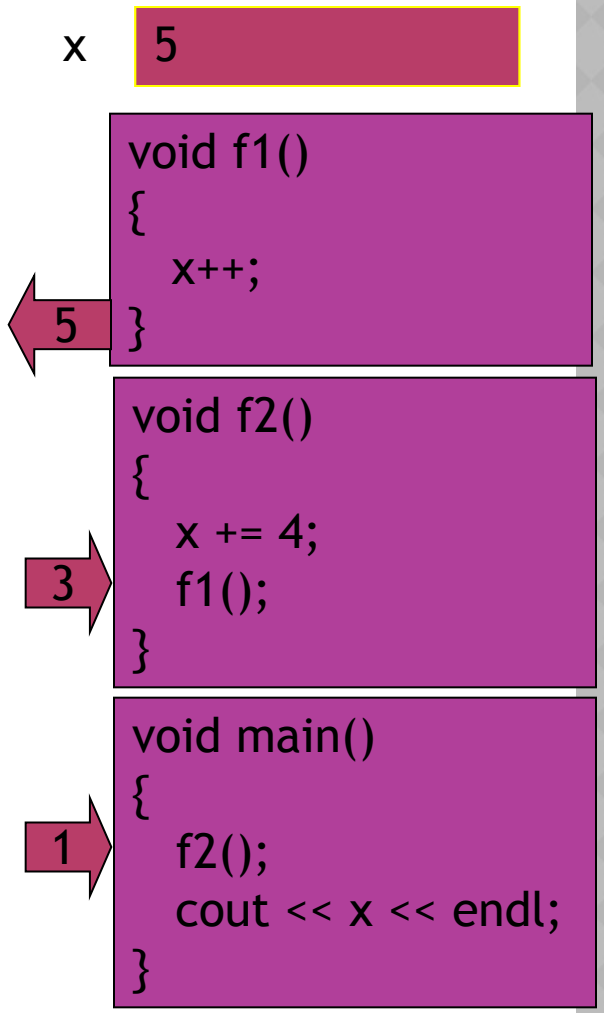
I. USING GLOBAL VARIABLES

```
#include <iostream>
int x = 0;
void f1() {  x++; }
void f2() { x+=4;
   f1(); }
void main()
{
   f2();
   cout << x << endl;
}
```



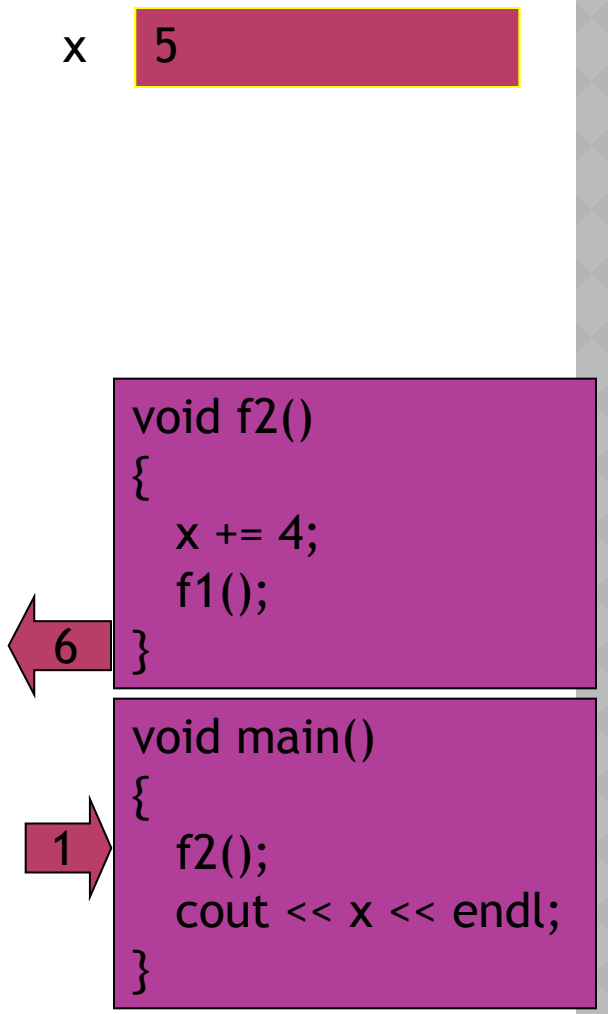
USING GLOBAL VARIABLES

```
#include
   <iostream.h>
int x = 0;
void f1() {  x++; }
void f2() { x+=4;
   f1(); }
void main()
{
   f2();
   cout << x << endl;
}
```



USING GLOBAL VARIABLES

```
#include
   <iostream.h>
int x = 0;
void f1() {  x++; }
void f2() { x+=4;
   f1(); }
void main()
{
   f2();
   cout << x << endl;
}
```



USING GLOBAL VARIABLES

```
#include <iostream.h>
int x = 0;
void f1() {  x++; }
void f2() { x+=4; f1(); }
void main()
{
   f2();
   cout << x << endl;
}
```

x 5

7

```
void main()
{
   f2();
   cout << x << endl;
}
```

USING GLOBAL VARIABLES

```
#include <iostream.h>
int x = 0;
void f1() {  x++; }
void f2() { x+=4; f1(); }
void main()
{
   f2();
   cout << x << endl;
}
```

x 5

8

```
void main()
{
   f2();
   cout << x << endl;
}
```

USING GLOBAL VARIABLES

```
#include <iostream.h>
int x = 0;
void f1() {  x++; }
void f2() { x+=4; f1(); }
void main()
{
   f2();
   cout << x << endl;
}
```

LOCAL VS GLOBAL VARIABLES

```
#include<iostream>
Using namespace std;
int x,y;  //Global Variables
int add2(int, int); //prototype
main()
{  int s;
   x = 11;
   y = 22;
   cout << "global x=" << x << endl;
   cout << "Global y=" << y << endl;
   s = add2(x, y);
   cout << x << "+" << y << "=" << s;
   cout<<endl;
   cout<<"\n---end of output---\n";
   return 0;
}
int add2(int x1,int y1)
{   int x; //local variables
   x=44;
   cout << "\nLocal x=" << x << endl;
   return x1+y1;
}
```

global x=11
global y=22
Local x=44
11+22=33
---end of output---

FUNCTION CALL METHODS

- Call by value
  - A copy of the value is passed
- Call by reference
  - The caller passes the address of the value
- **Call by value**

- Up to this point all the calls we have seen are call-by-value, a copy of the value (known) is passed from the caller-function to the called-function
- Any change to the copy does not affect the original value in the caller function
- Advantages, prevents side effect, resulting in reliable software

FUNCTION CALL METHODS

Call By Reference

- We introduce reference-parameter, to perform call by reference. The caller gives the called function the ability to directly access the caller's value, and to modify it.
- A reference parameter is an alias for it's corresponding argument, it is stated in c++ by "flow the parameter's type" in the function prototype by an ampersand(&) also in the function definition-header.
- Advantage: performance issue

```
void  function_name (type &);// prototype

main()
{
   -----
   ------
}
void function_name(type &parameter_name)
```

FUNCTION CALL EXAMPLE

```
#include<iostream >
using namespace std;
void passbyref(int &cz);
void passybyval(int a);
int main()
{
   int x=2;
   cout<<x<<"Before calling"<<endl;
   passybyval(x);
   cout<<x<<"after calling"<<endl;
   cout<<x<<"Before calling"<<endl;
   passbyref(x);
   cout<<x<<"after calling"<<endl;

   return 0;
}
void passybyval(int a)
{
    a++;
}
void passbyref(int &cz)
{
   cz++;
}
```

SWAPPING

```
#include<iostream >
using namespace std;
void swap(int &a, int &b)
{
	int c=a;
	a=b;
	b=c;
}
int main()
{
	int a, b;
	cin>>a>>b;
	cout<<a<<" "<<b<<endl;
	swap(a,b);
	cout<<a<<" "<<b<<endl;

}
```