C/C++ Program Design

LAB 6

CONTENTS

- Master how to declare, define, and call a user-defined function

2 Knowledge Points

2.1 User-Defined Function

2.2 Recursive function

2.3 Pointers to functions

2.1 User-Defined Function

Syntax of defining a function:

function header

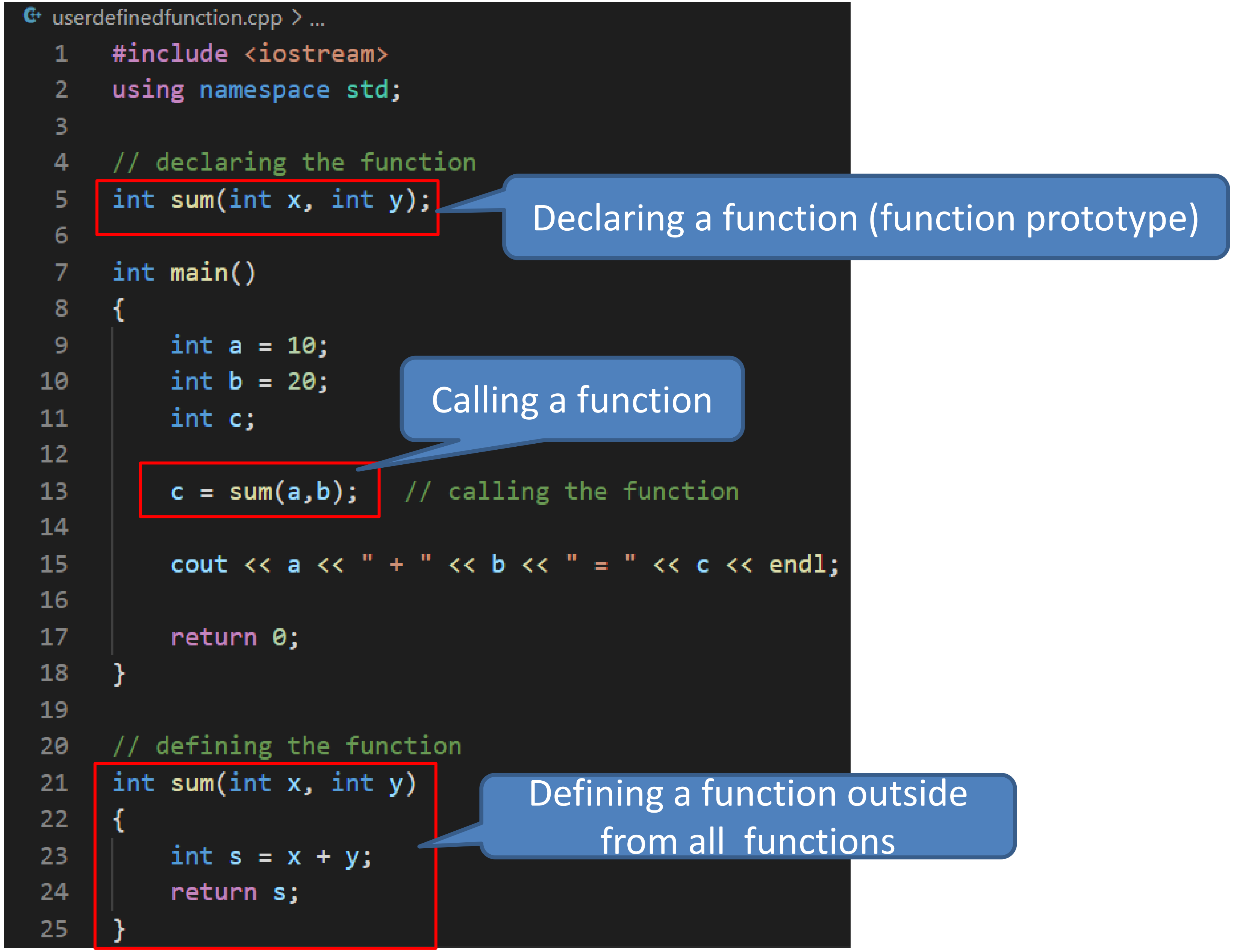
```
return_type function_name (datatype parameter1, datatype parameter2, …)
{
        // function body
}
```

- **return type:** suggests what type the function will return. It can be int,char, string, pointer or even a class object. If a function does not return anything, it is mentioned with void.
- **function name:** is the name of the function, using a legal identifier.
- **parameters:** are variables to hold values of arguments passed while function is called. A function may not contain parameter list, give **void** in the parentheses.

Function prototype:

The simplest way to get a prototype is to copy the **function header** and add a **semicolon**.

Example: Declaring, Defining and Calling a function



userdefinedfunction.cpp > ...

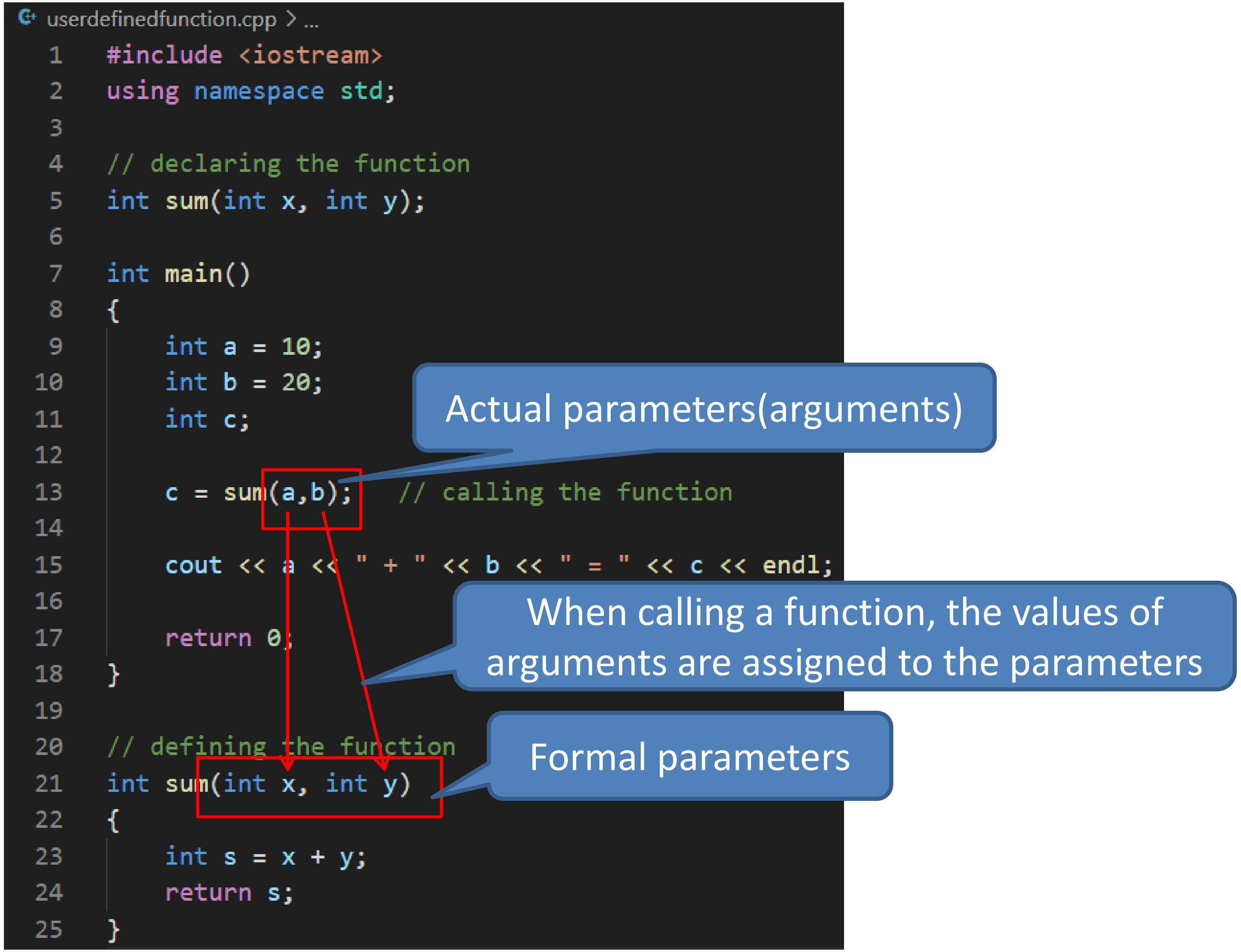
```cpp
#include <iostream>
using namespace std;

// declaring the function
int sum(int x, int y);

int main()
{
    int a = 10;
    int b = 20;
    int c;

    c = sum(a,b);   // calling the function

    cout << a << " + " << b << " = " << c << endl;

    return 0;
}

// defining the function
int sum(int x, int y)
{
    int s = x + y;
    return s;
}
```

Declaring a function (function prototype)

Calling a function

Defining a function outside from all functions

Actual parameter and Formal parameter

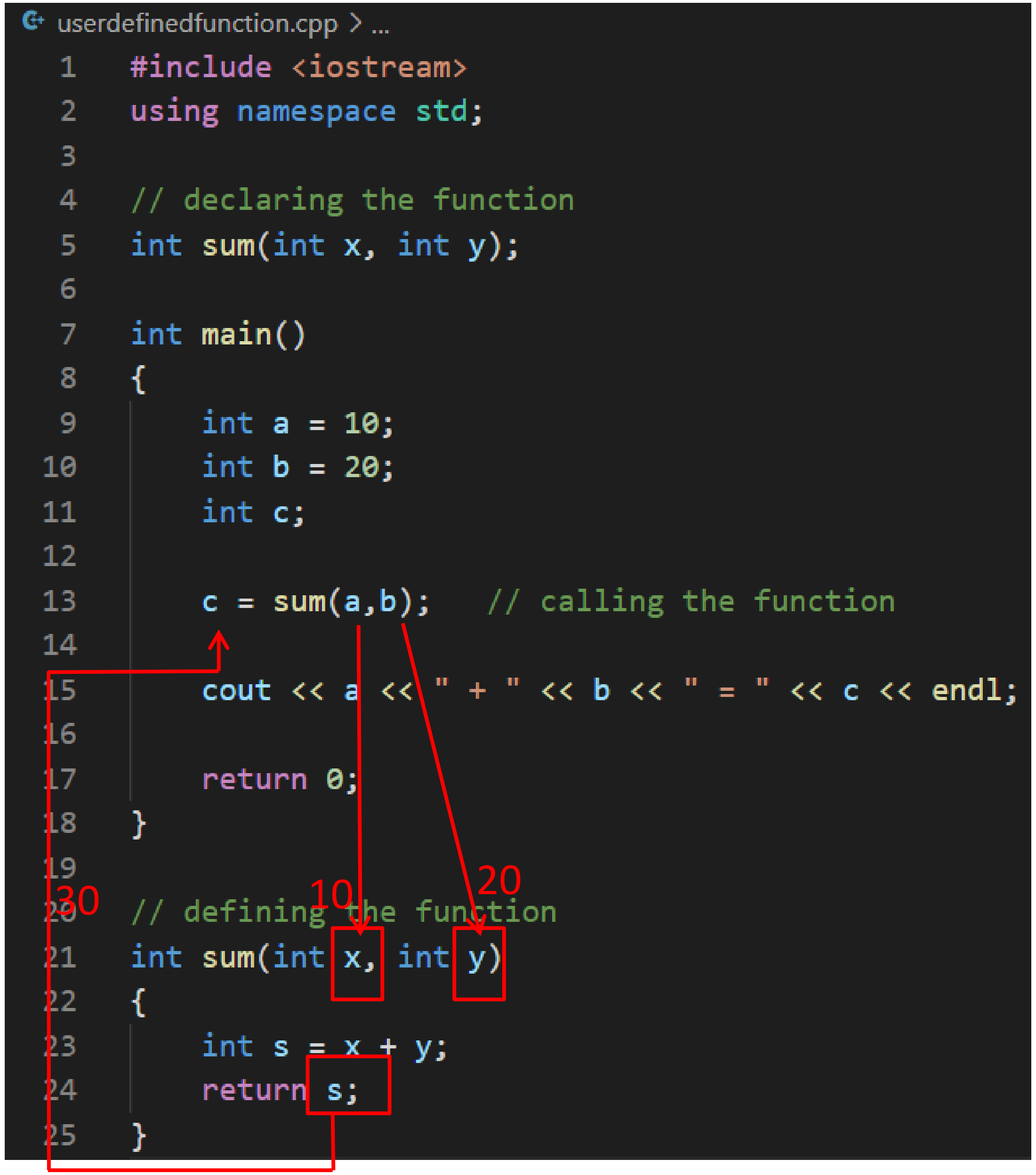


```cpp
#include <iostream>
using namespace std;

// declaring the function
int sum(int x, int y);

int main()
{
    int a = 10;
    int b = 20;
    int c;

    c = sum(a,b);    // calling the function

    cout << a << " + " << b << " = " << c << endl;

    return 0;
}

// defining the function
int sum(int x, int y)
{
    int s = x + y;
    return s;
}
```

Actual parameters(arguments)

When calling a function, the values of arguments are assigned to the parameters

Formal parameters



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

// declaring the function
int sum(int x, int y);

int main()
{
    int a = 10;
    int b = 20;
    int c;

    c = sum(a,b);   // calling the function

    cout << a << " + " << b << " = " << c << endl;

    return 0;
}

// defining the function
int sum(int x, int y)
{
    int s = x + y;
    return s;
}
```

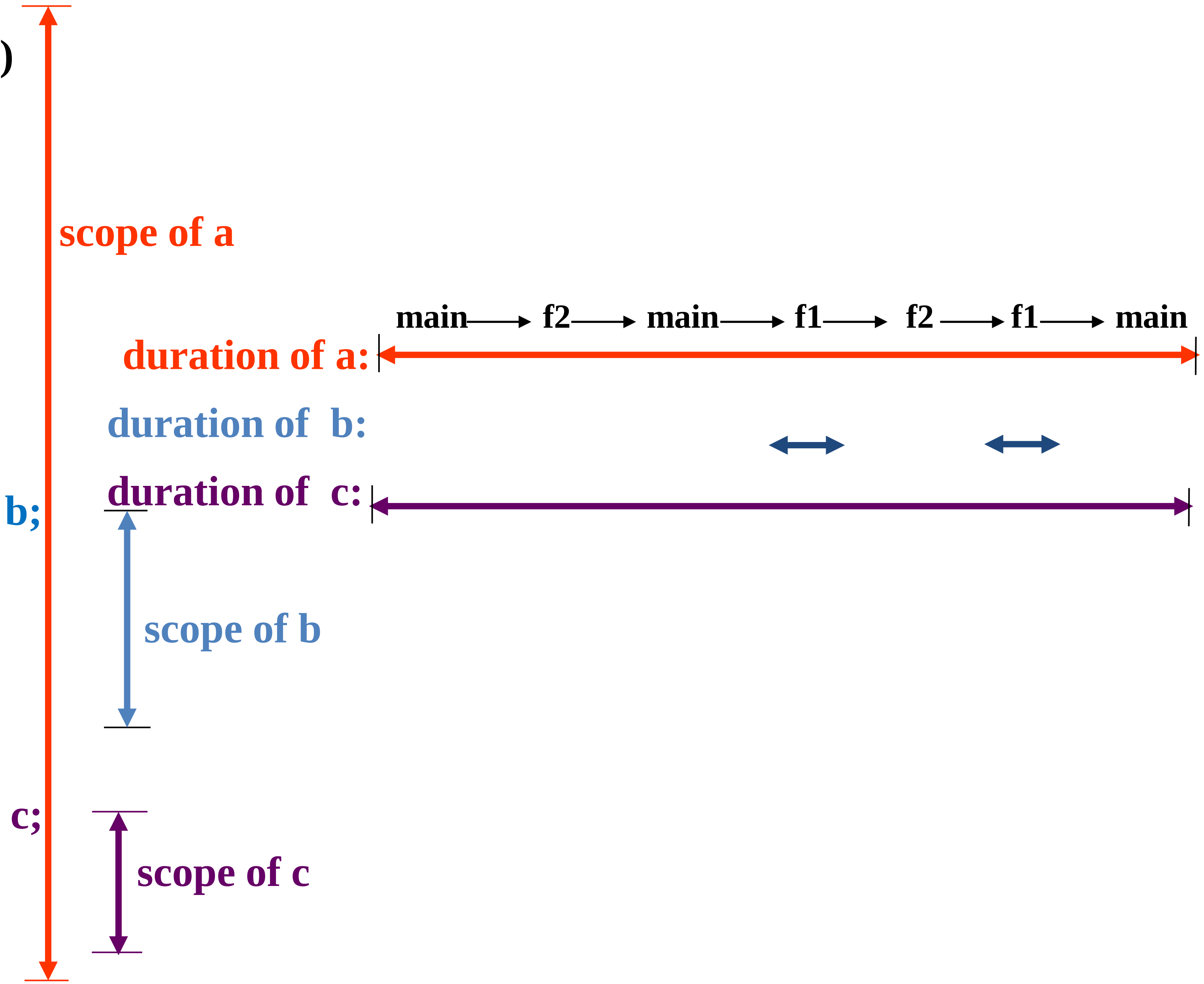
Process of the calling a function:

- The values of arguments are assigned to the those of parameters by the sequence of their definition from left to right one by one.
- The control flows into the function body and executes the statements inside the body.
- When it encounters the return statement, the control flow returns back to the calling function with a return value.

Scope and duration of a variable

- An variable’s **scope** is where the variable can be referenced in a program. Some identifiers can be referenced throughout a program, others from only portions of a program.
- A variable defined inside a function is referred to as a **local variable**. A **global variable** is defined outside functions.
- The **scope of a local variable** is from where it is defined to the end of the block which it is included or the end of the function.
- The **scope of a global variable** is from where it is defined to the end of the file(or the program).
- An variable’s **storage duration** is the period during which that variable exists in memory.

```
int a;
void main( )
{     .......
      .......
        f2;
      .......
        f1;
      .......
}
f1( )
{   auto int b;
     .........
     f2;
      ........
}
f2( )
{  static int c;
     .........
}
```



scope of a

main → f2 → main → f1 → f2 → f1 → main

duration of a:

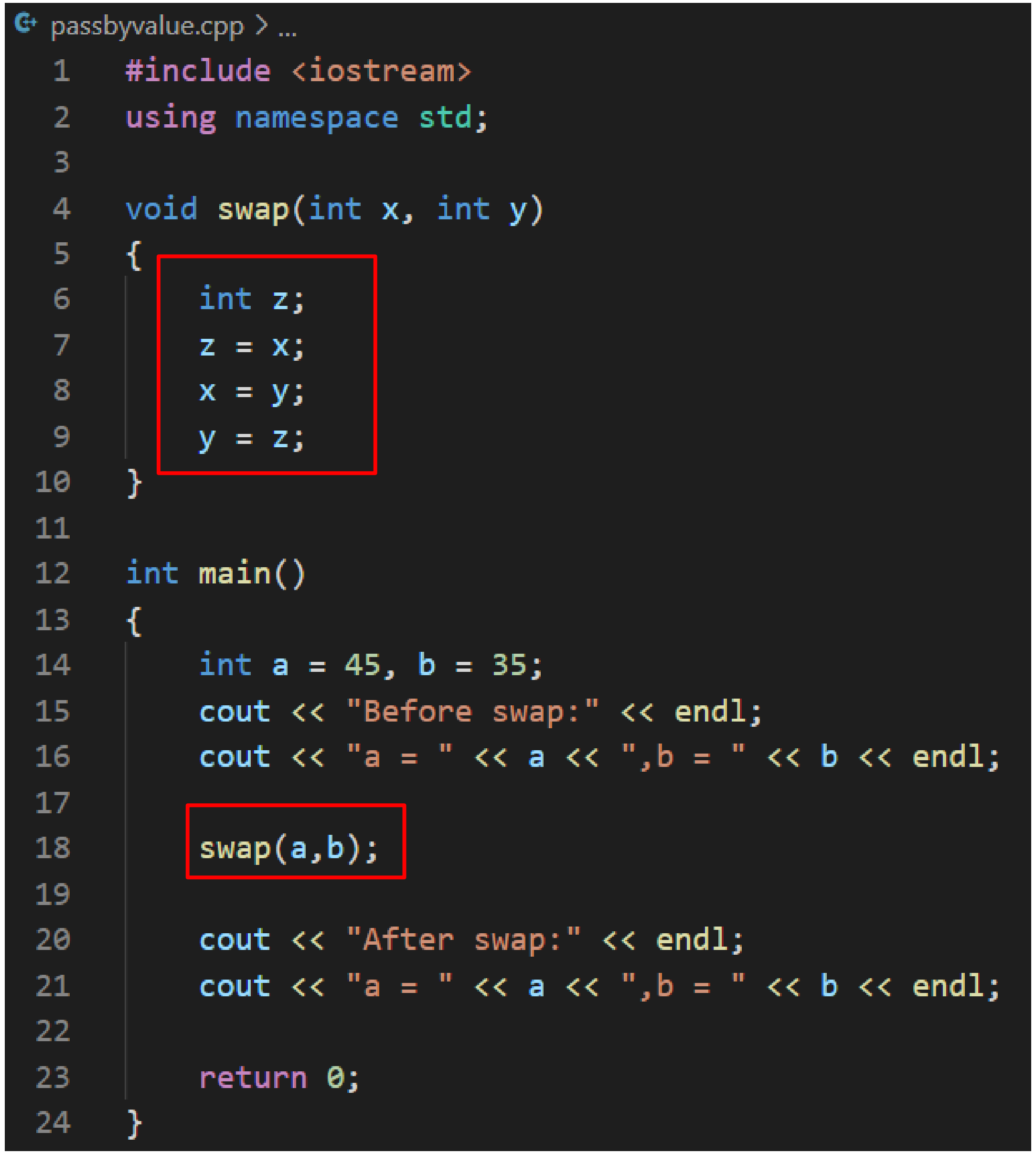
duration of b:

duration of c:

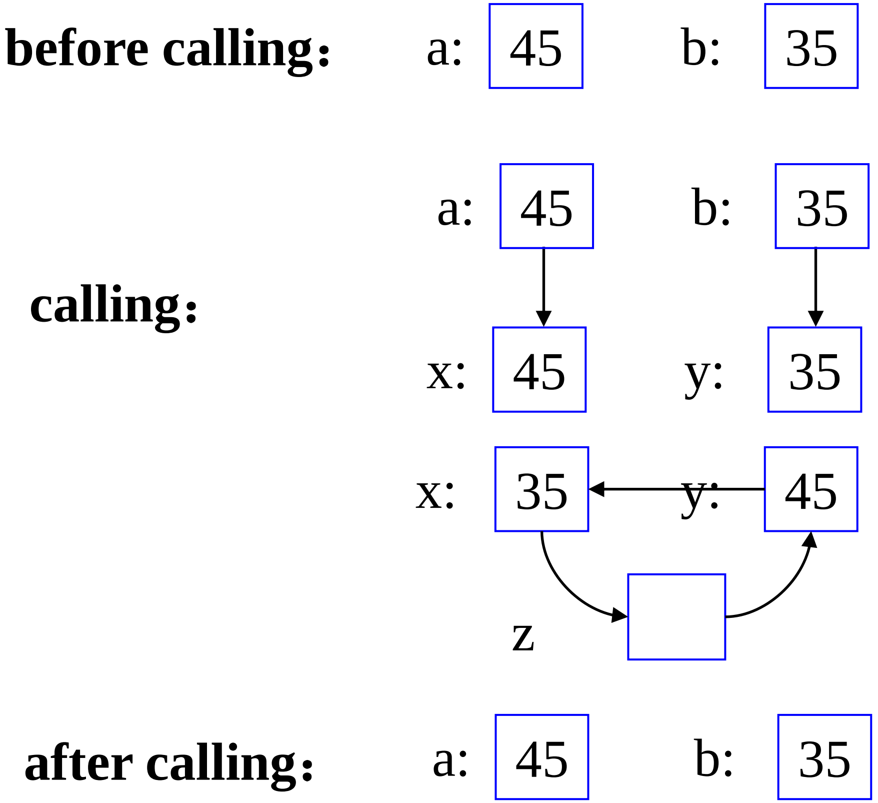
scope of b

scope of c

1. Passing arguments to a function by value

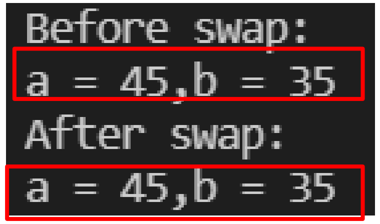


```cpp
#include <iostream>
using namespace std;

void swap(int x, int y)
{
    int z;
    z = x;
    x = y;
    y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before swap:" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    swap(a,b);

    cout << "After swap:" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```


before calling:

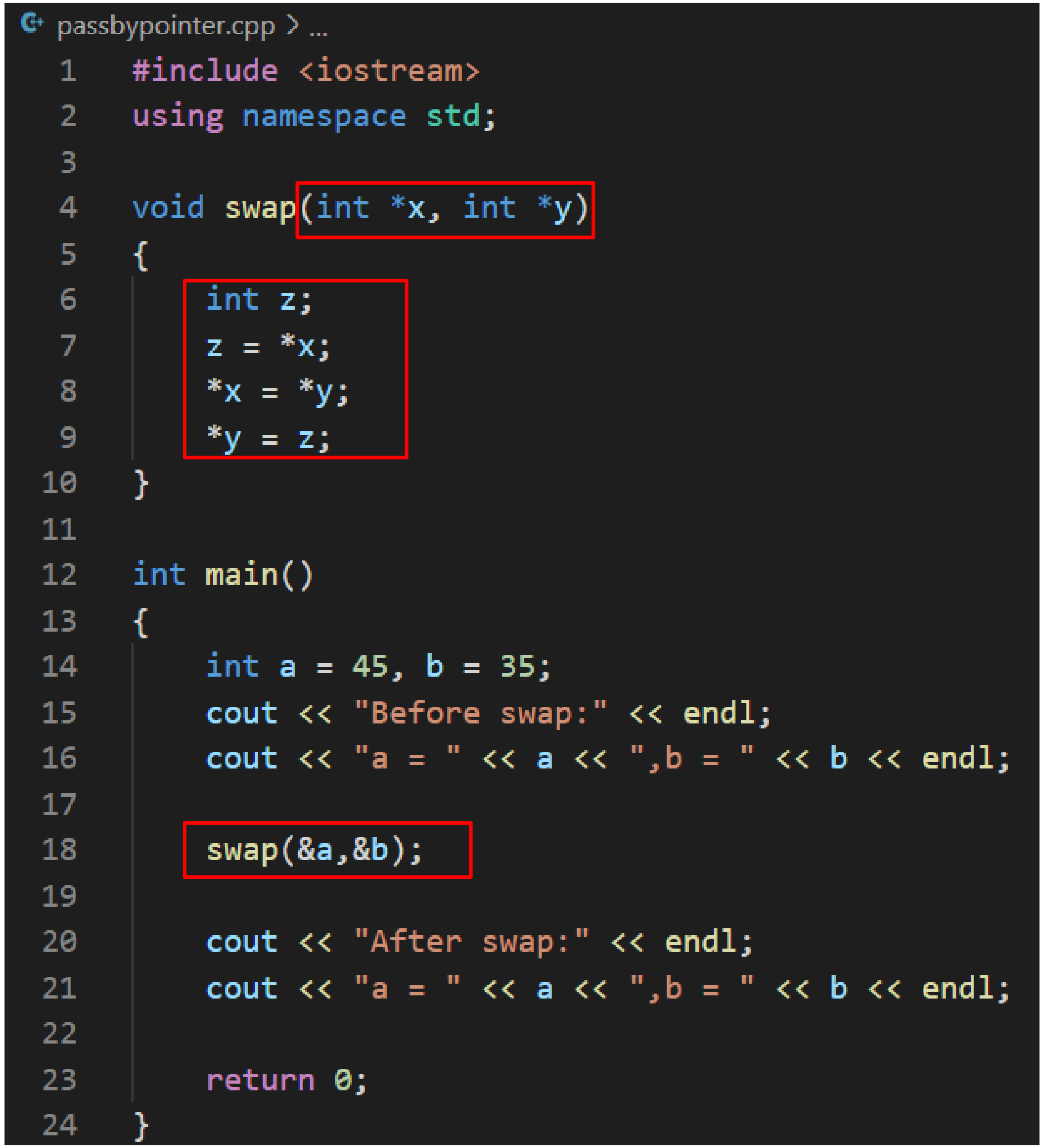
calling:

after calling:



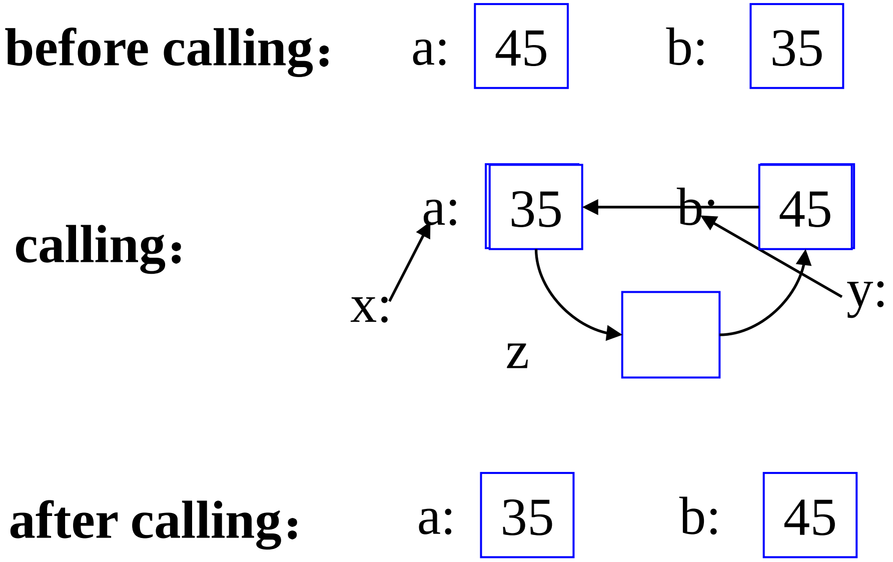
```
Before swap:
a = 45,b = 35
After swap:
a = 45,b = 35
```

2. Passing arguments to a function by pointer



passbypointer.cpp > ...

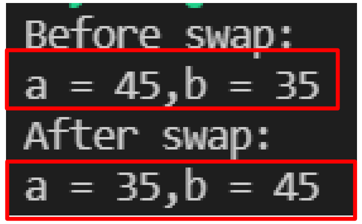
```cpp
#include <iostream>
using namespace std;

void swap(int *x, int *y)
{
    int z;
    z = *x;
    *x = *y;
    *y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before swap:" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    swap(&a,&b);

    cout << "After swap:" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```



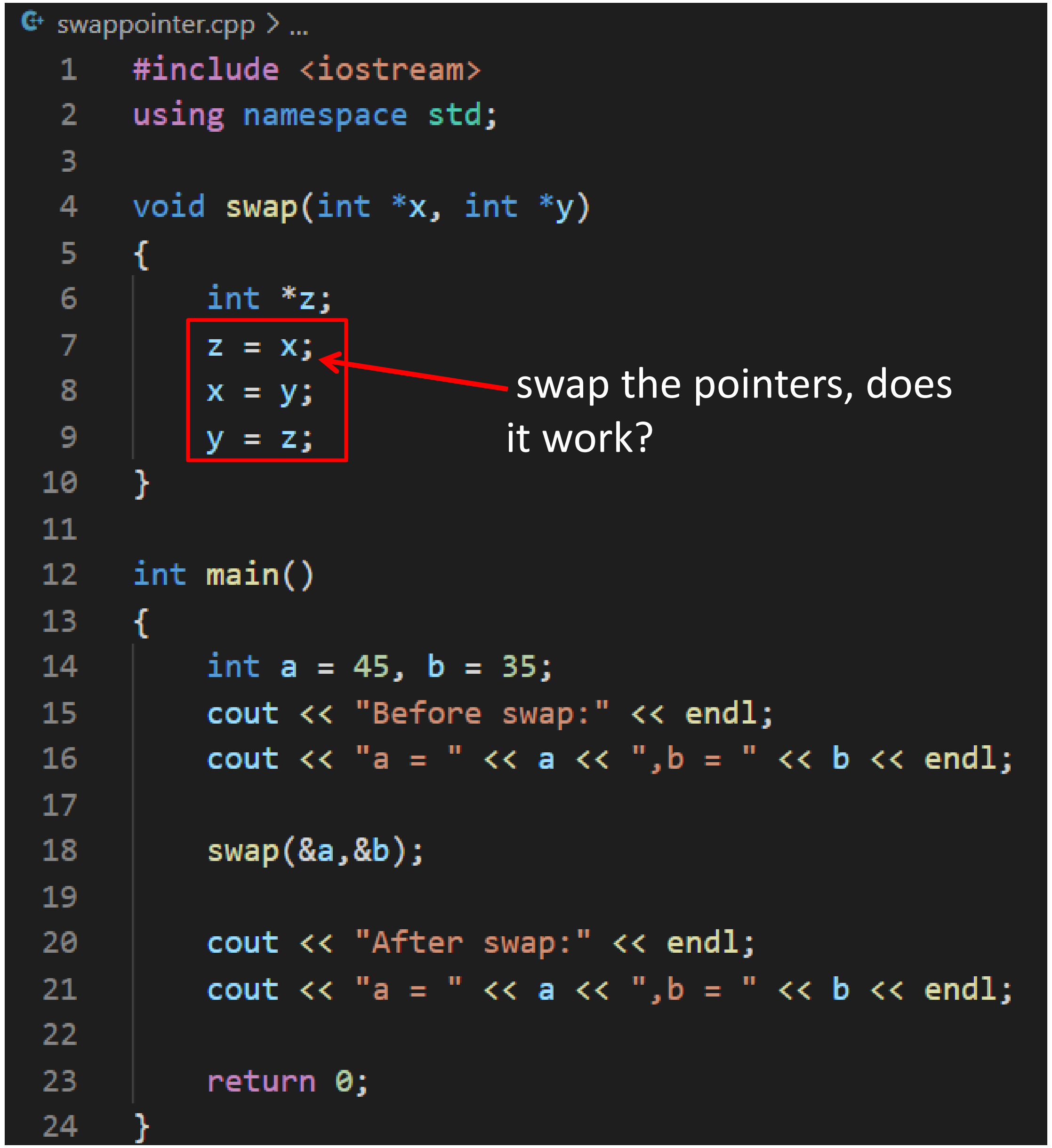
before calling: a: 45 b: 35

calling: a: 35 b: 45 x: y: z

after calling: a: 35 b: 45



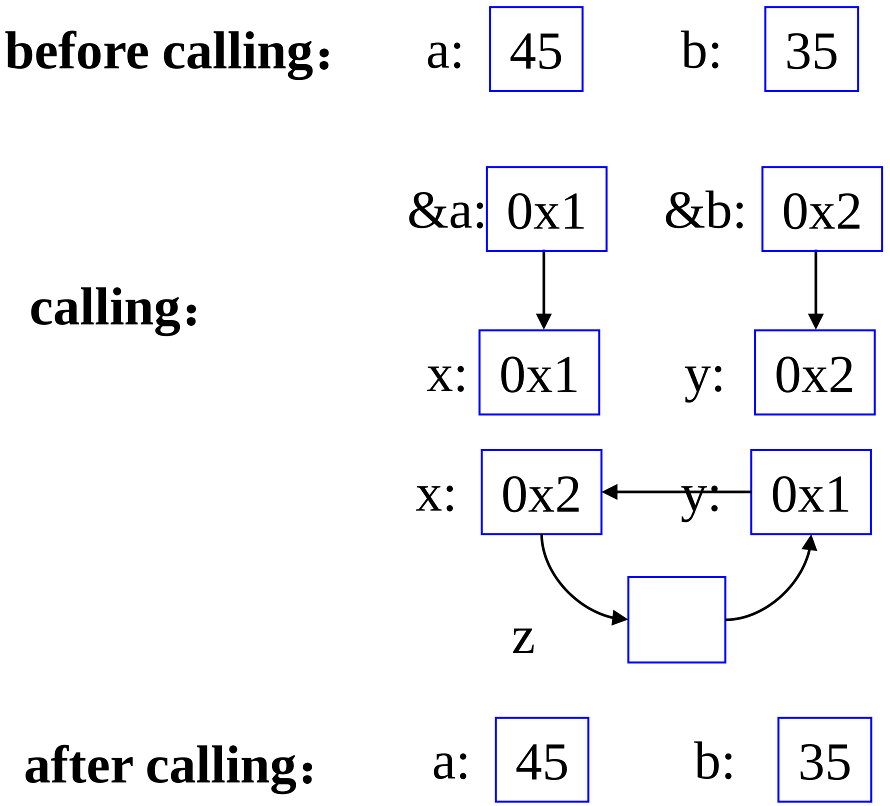
```
Before swap:
a = 45,b = 35
After swap:
a = 35,b = 45
```



swappointer.cpp > ...

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

void swap(int *x, int *y)
{
    int *z;
    z = x;
    x = y;
    y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before swap:" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    swap(&a,&b);

    cout << "After swap:" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```

swap the pointers, does it work?



before calling: a: 45 b: 35

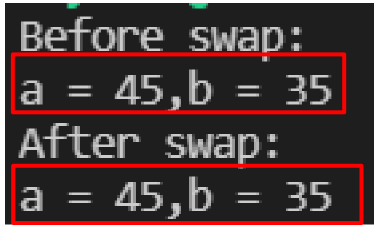
calling: &a: 0x1 &b: 0x2

x: 0x1 y: 0x2

x: 0x2 y: 0x1

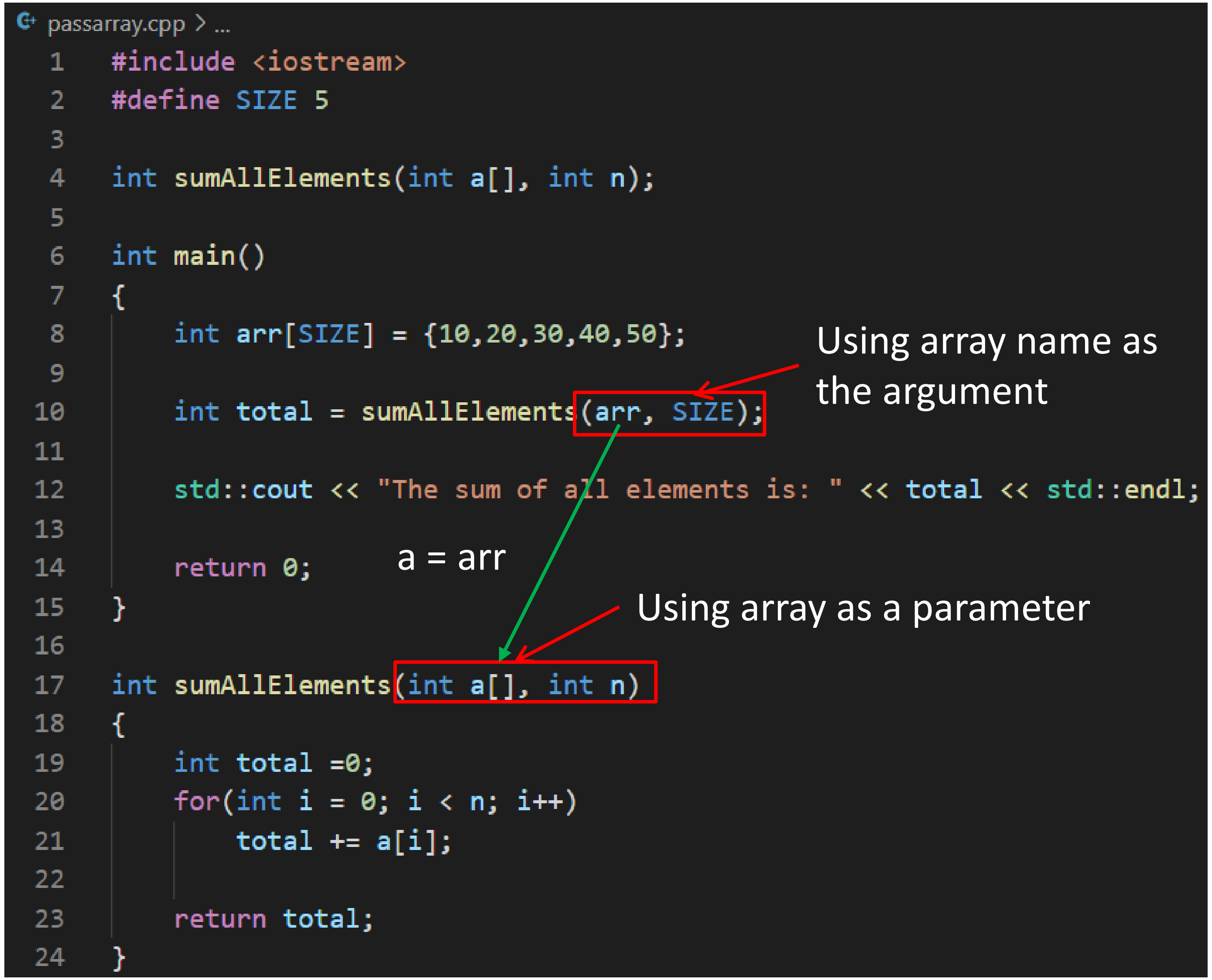
z

after calling: a: 45 b: 35



```
Before swap:
a = 45,b = 35
After swap:
a = 45,b = 35
```

3. Passing arrays to a function
(array as parameters and arguments)



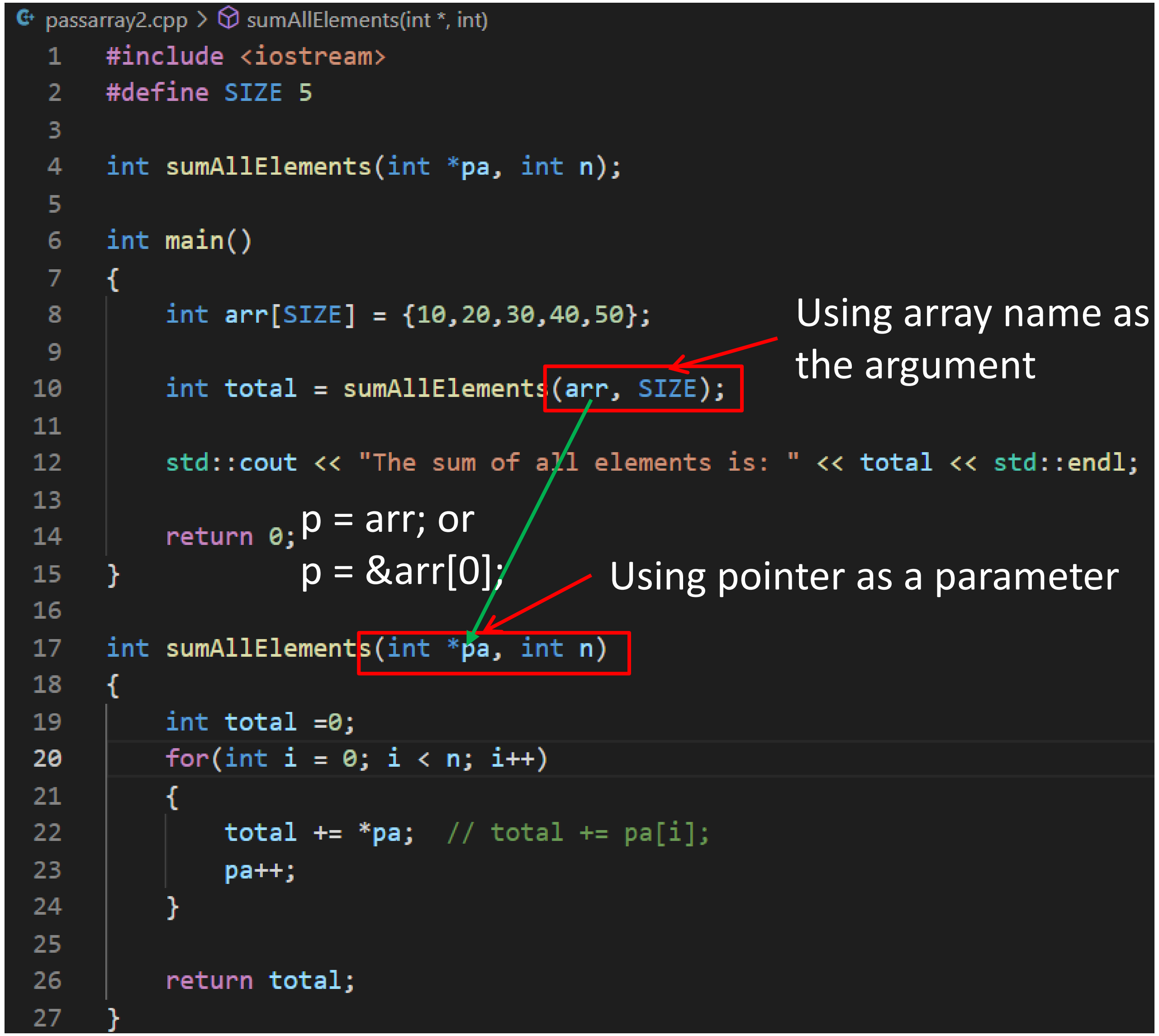
```cpp
#include <iostream>
#define SIZE 5

int sumAllElements(int a[], int n);

int main()
{
    int arr[SIZE] = {10,20,30,40,50};

    int total = sumAllElements(arr, SIZE);

    std::cout << "The sum of all elements is: " << total << std::endl;

    return 0;
}

int sumAllElements(int a[], int n)
{
    int total =0;
    for(int i = 0; i < n; i++)
        total += a[i];

    return total;
}
```

Using array name as the argument

a = arr

Using array as a parameter

3. Passing arrays to a function
(pointers as parameters and array name as arguments)



```cpp
#include <iostream>
#define SIZE 5

int sumAllElements(int *pa, int n);

int main()
{
    int arr[SIZE] = {10,20,30,40,50};

    int total = sumAllElements(arr, SIZE);

    std::cout << "The sum of all elements is: " << total << std::endl;

    return 0;
}

int sumAllElements(int *pa, int n)
{
    int total =0;
    for(int i = 0; i < n; i++)
    {
        total += *pa;  // total += pa[i];
        pa++;
    }

    return total;
}
```

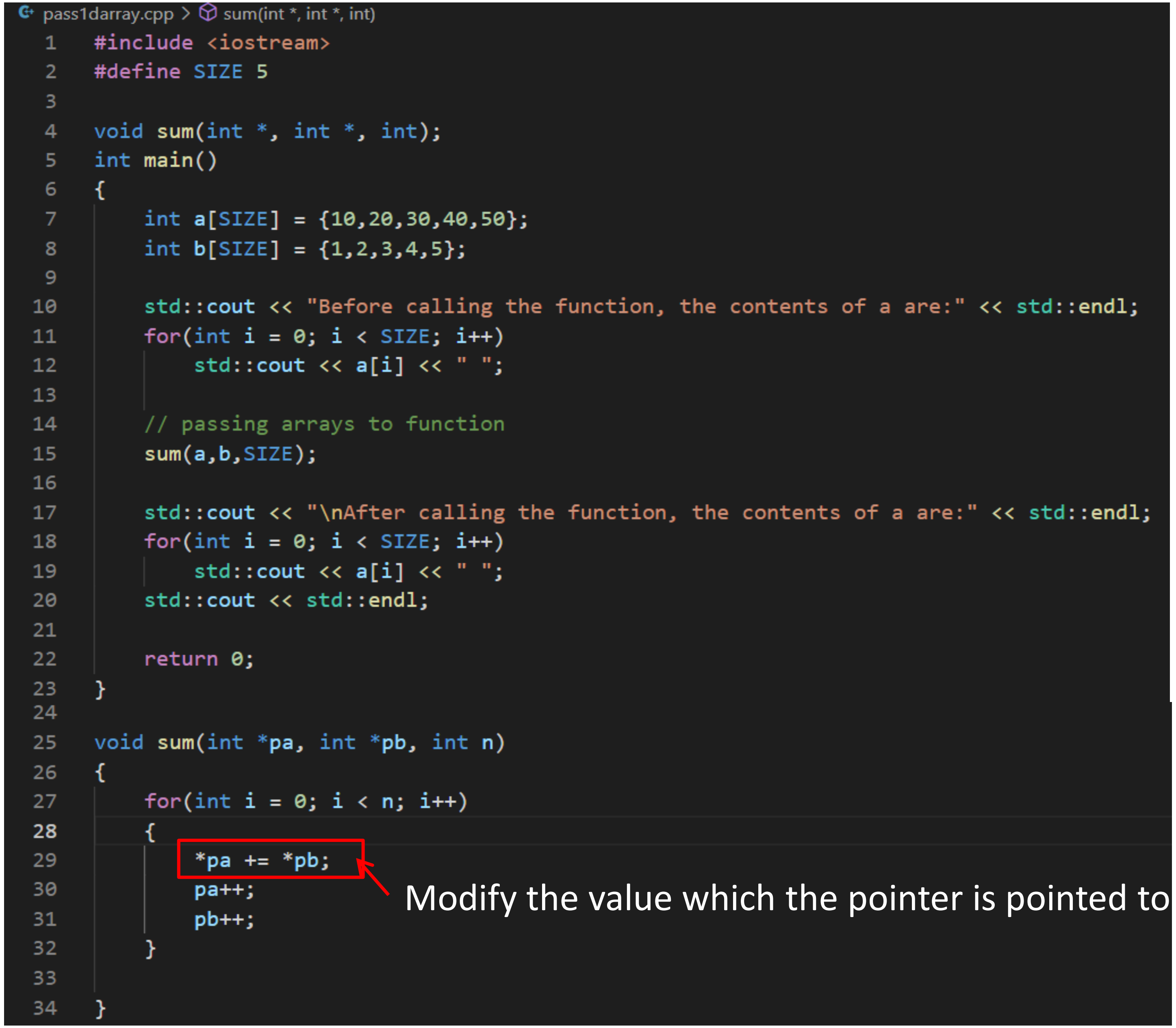
Using array name as the argument

p = arr; or

p = &arr[0];

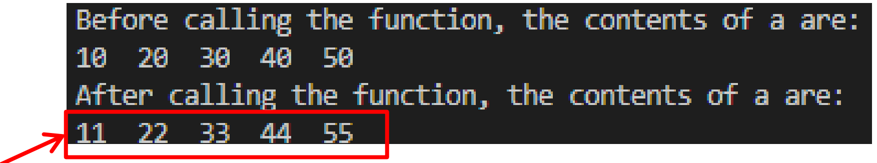
Using pointer as a parameter

3. Passing arrays to a function
(The values in an array can be modified inside the function body)



```cpp
#include <iostream>
#define SIZE 5

void sum(int *, int *, int);
int main()
{
    int a[SIZE] = {10,20,30,40,50};
    int b[SIZE] = {1,2,3,4,5};

    std::cout << "Before calling the function, the contents of a are:" << std::endl;
    for(int i = 0; i < SIZE; i++)
        std::cout << a[i] << " ";

    // passing arrays to function
    sum(a,b,SIZE);

    std::cout << "\nAfter calling the function, the contents of a are:" << std::endl;
    for(int i = 0; i < SIZE; i++)
        std::cout << a[i] << " ";
    std::cout << std::endl;

    return 0;
}

void sum(int *pa, int *pb, int n)
{
    for(int i = 0; i < n; i++)
    {
        *pa += *pb;
        pa++;
        pb++;
    }

}
```

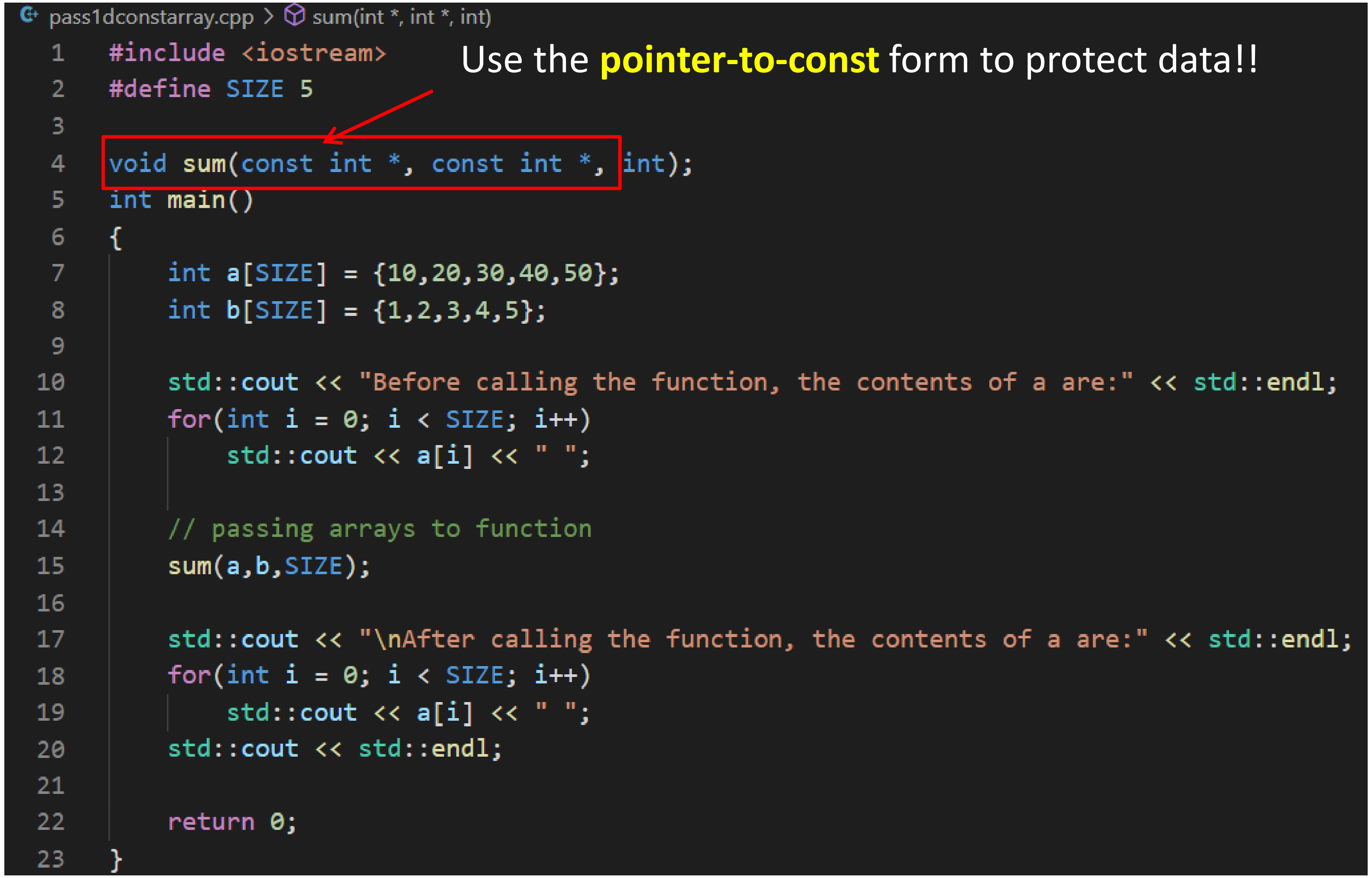
Modify the value which the pointer is pointed to



```
Before calling the function, the contents of a are:
10  20  30  40  50
After calling the function, the contents of a are:
11  22  33  44  55
```

The values of elements in array **a** are changed.

3. Passing arrays to a function
(protect the value of the argument from modifying, please use const)

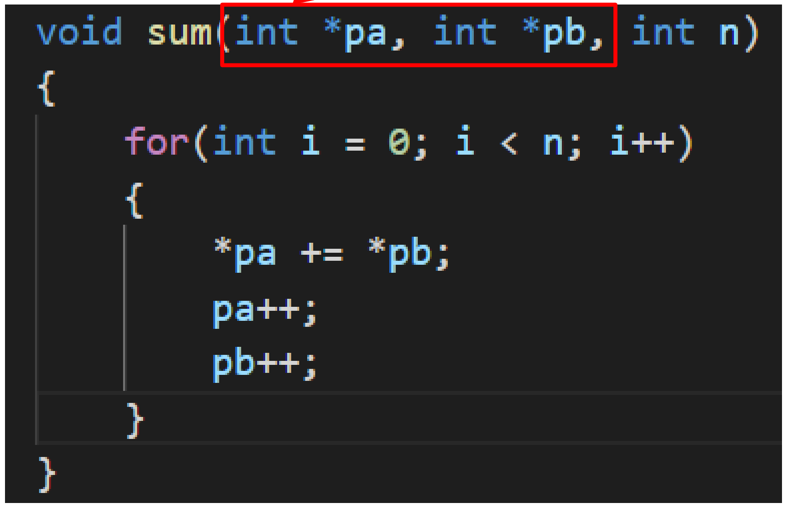


pass1dconstarray.cpp > sum(int *, int *, int)

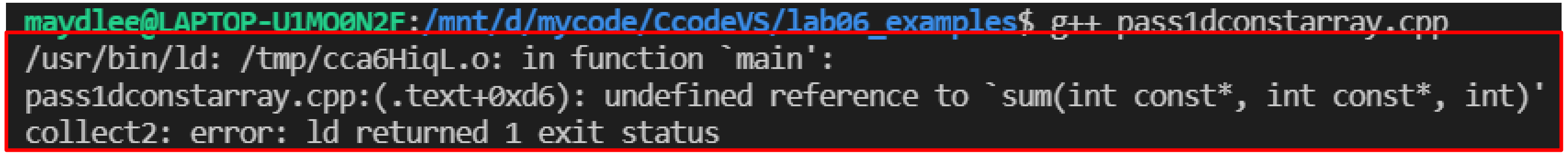
Use the **pointer-to-const** form to protect data!!

```cpp
#include <iostream>
#define SIZE 5

void sum(const int *, const int *, int);
int main()
{
    int a[SIZE] = {10,20,30,40,50};
    int b[SIZE] = {1,2,3,4,5};

    std::cout << "Before calling the function, the contents of a are:" << std::endl;
    for(int i = 0; i < SIZE; i++)
        std::cout << a[i] << " ";

    // passing arrays to function
    sum(a,b,SIZE);

    std::cout << "\nAfter calling the function, the contents of a are:" << std::endl;
    for(int i = 0; i < SIZE; i++)
        std::cout << a[i] << " ";
    std::cout << std::endl;

    return 0;
}
```

In definition, if the const is omitted, it will cause compiling error.

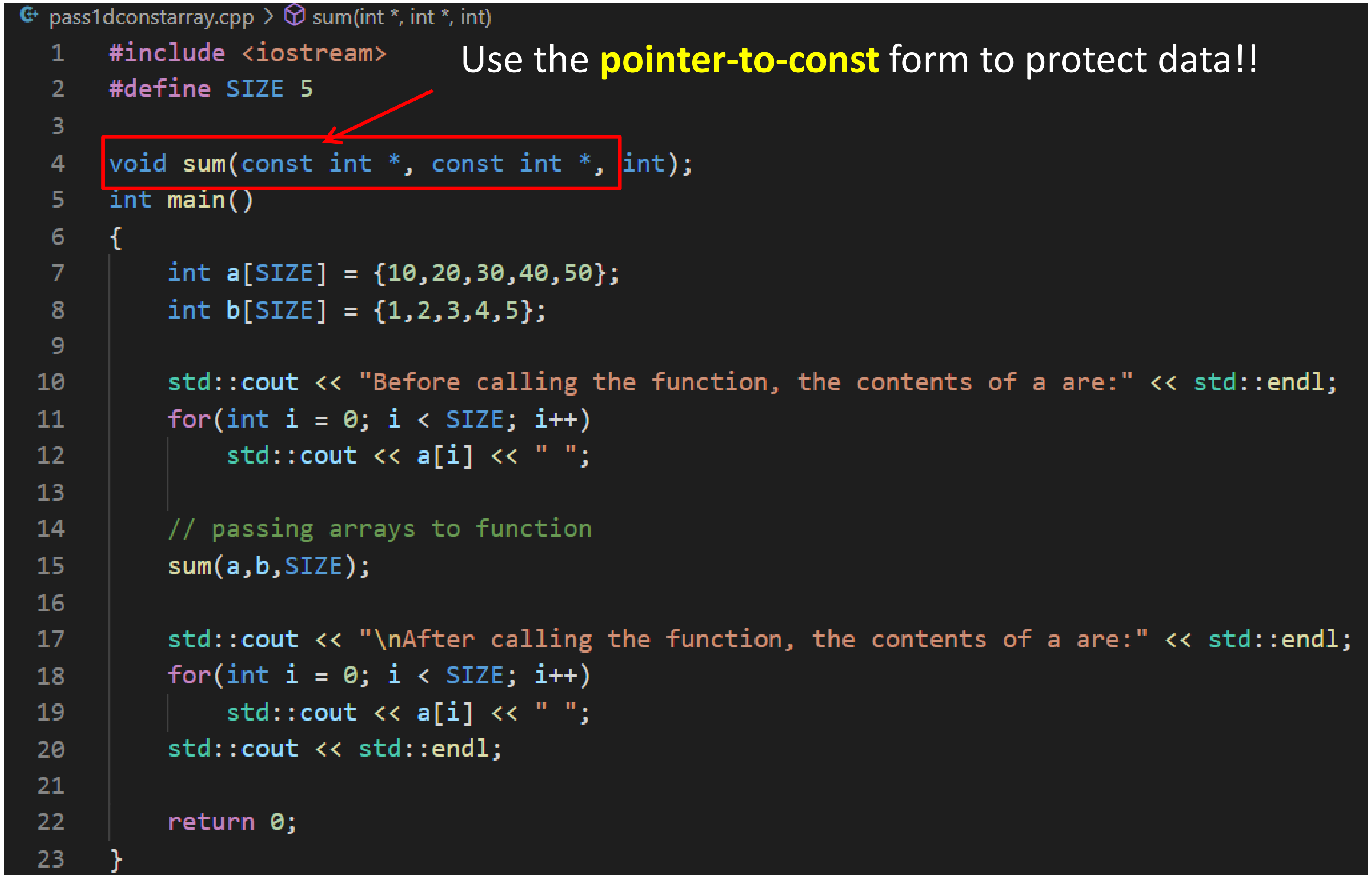


```cpp
void sum(int *pa, int *pb, int n)
{
    for(int i = 0; i < n; i++)
    {
        *pa += *pb;
        pa++;
        pb++;
    }
}
```



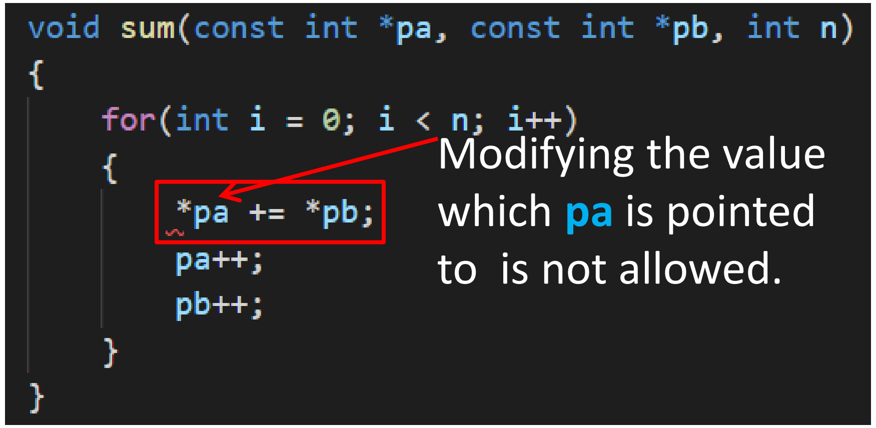
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab06_examples$ g++ pass1dconstarray.cpp
/usr/bin/ld: /tmp/cca6HiqL.o: in function `main':
pass1dconstarray.cpp:(.text+0xd6): undefined reference to `sum(int const*, int const*, int)'
collect2: error: ld returned 1 exit status
```

3. Passing arrays to a function
(protect the value of the argument from modifying, please use const)



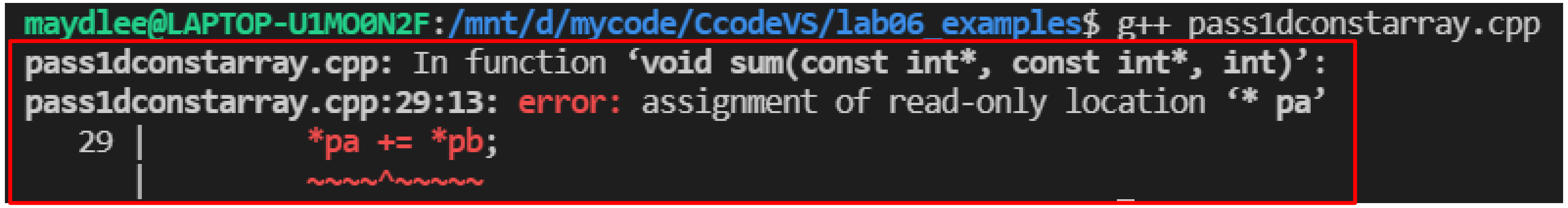
Use the **pointer-to-const** form to protect data!!

```cpp
#include <iostream>
#define SIZE 5

void sum(const int *, const int *, int);
int main()
{
    int a[SIZE] = {10,20,30,40,50};
    int b[SIZE] = {1,2,3,4,5};

    std::cout << "Before calling the function, the contents of a are:" << std::endl;
    for(int i = 0; i < SIZE; i++)
        std::cout << a[i] << " ";

    // passing arrays to function
    sum(a,b,SIZE);

    std::cout << "\nAfter calling the function, the contents of a are:" << std::endl;
    for(int i = 0; i < SIZE; i++)
        std::cout << a[i] << " ";
    std::cout << std::endl;

    return 0;
}
```



```cpp
void sum(const int *pa, const int *pb, int n)
{
    for(int i = 0; i < n; i++)
    {
        *pa += *pb;
        pa++;
        pb++;
    }
}
```

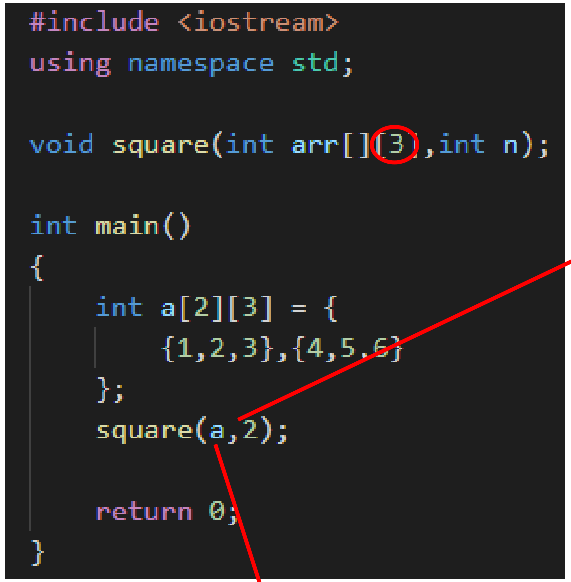
Modifying the value which **pa** is pointed to is not allowed.



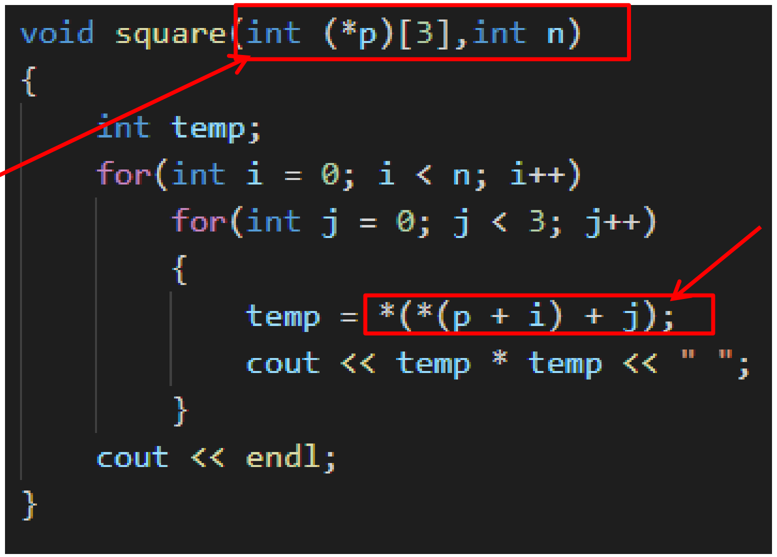
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab06_examples$ g++ pass1dconstarray.cpp
pass1dconstarray.cpp: In function ‘void sum(const int*, const int*, int)’:
pass1dconstarray.cpp:29:13: error: assignment of read-only location ‘* pa’
   29 |         *pa += *pb;
      |         ~~~~^~~~~~
```

4. Passing multidimensional array to a function

Passing two-dimensional array as a parameter, the length of column can not be omitted.



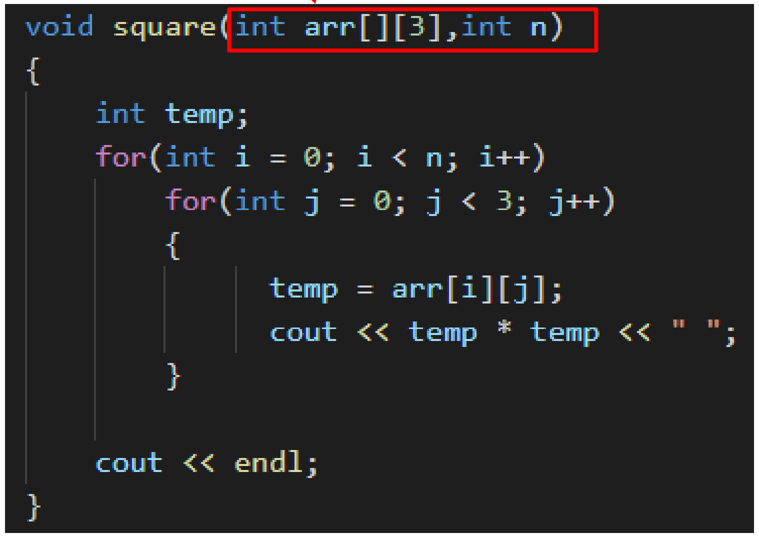
```cpp
#include <iostream>
using namespace std;

void square(int arr[][3],int n);

int main()
{
    int a[2][3] = {
        {1,2,3},{4,5,6}
    };
    square(a,2);

    return 0;
}
```



```cpp
void square(int (*p)[3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = *(*(p + i) + j);
            cout << temp * temp << " ";
        }
    cout << endl;
}
```

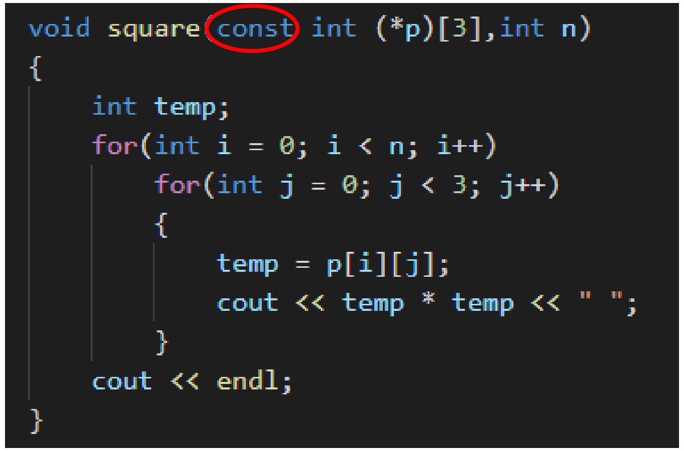
the same as

p[i][j]

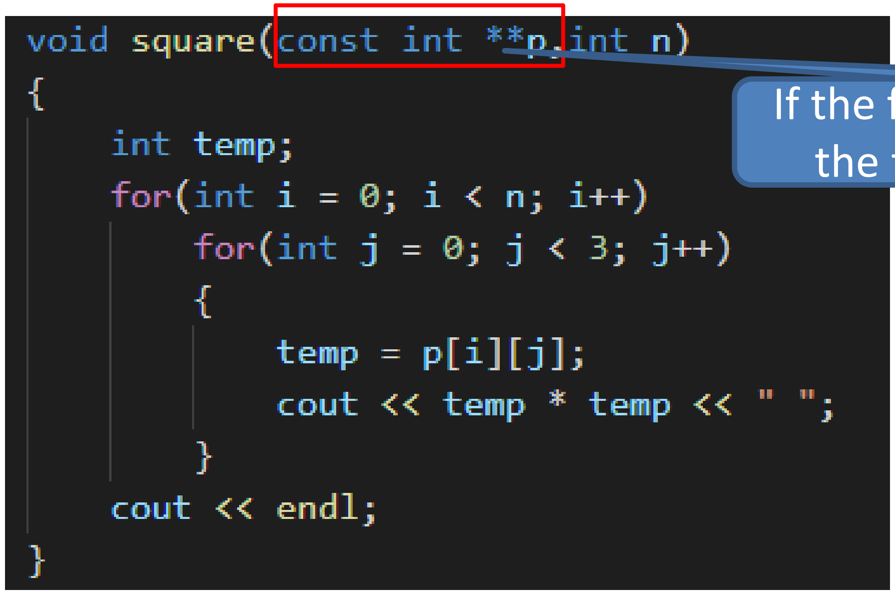


```cpp
void square(int arr[][3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = arr[i][j];
            cout << temp * temp << " ";
        }

    cout << endl;
}
```

If the values in the array can not be modified, use **const** in the prototype and definition.
void square(**const** int arr[][3],int n);

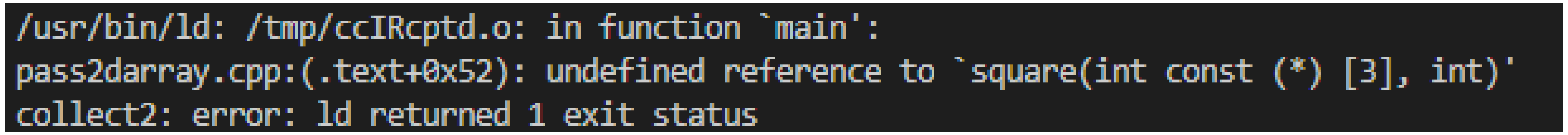


```cpp
void square(const int (*p)[3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = p[i][j];
            cout << temp * temp << " ";
        }
    cout << endl;
}
```



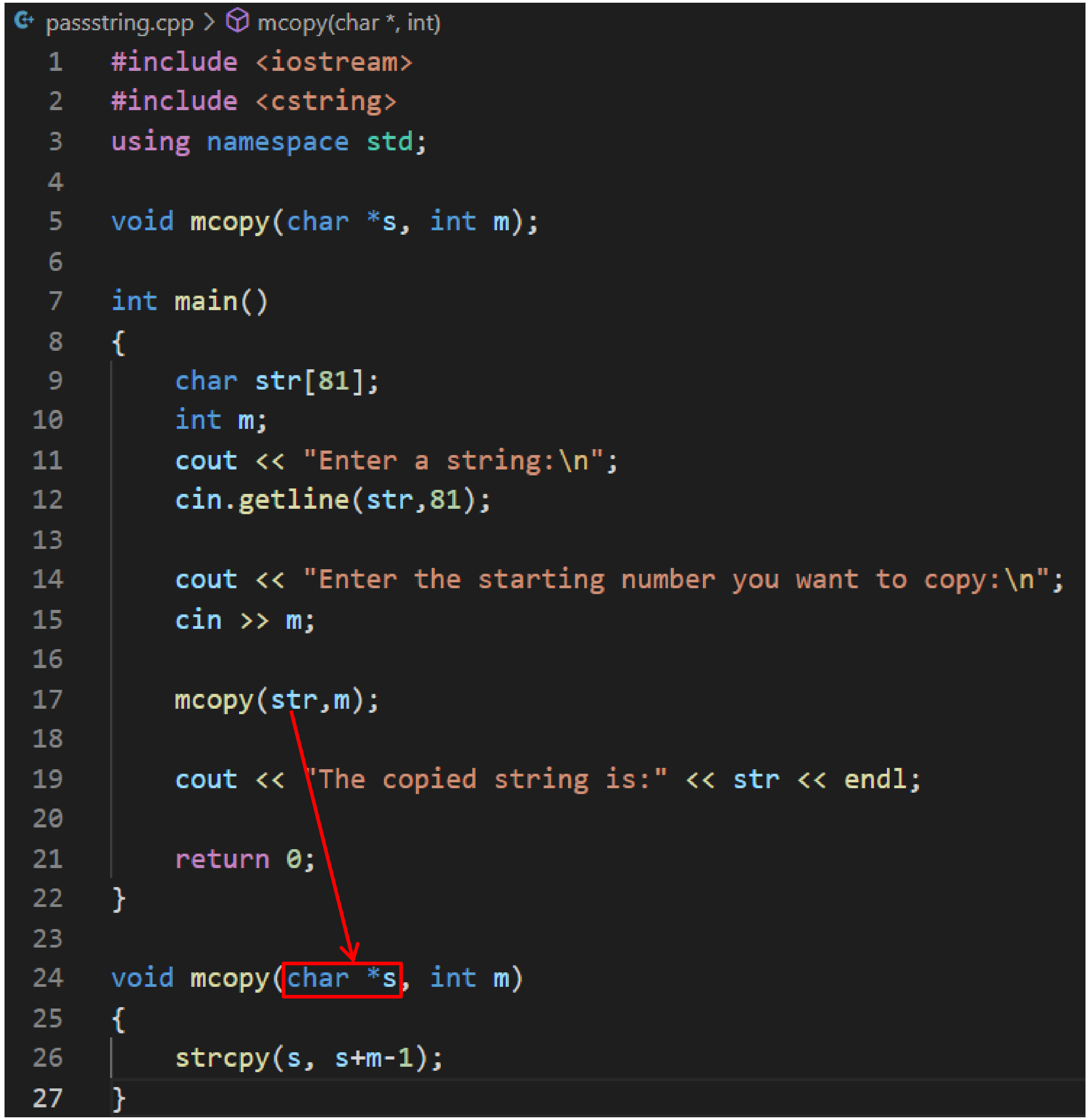
```
void square(const int **p,int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = p[i][j];
            cout << temp * temp << " ";
        }
    cout << endl;
}
```

If the function definition is like this, can we invoke the function by two-dimensional array name?



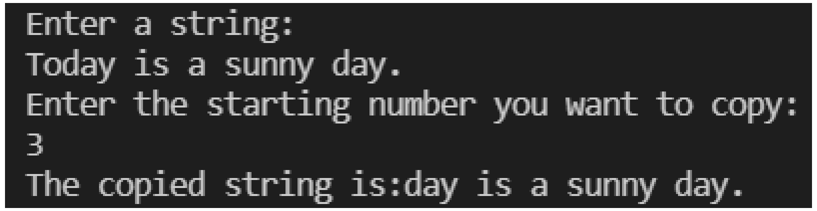
```
/usr/bin/ld: /tmp/ccIRcptd.o: in function `main':
pass2darray.cpp:(.text+0x52): undefined reference to `square(int const (*) [3], int)'
collect2: error: ld returned 1 exit status
```

5. Passing C-style string to a function



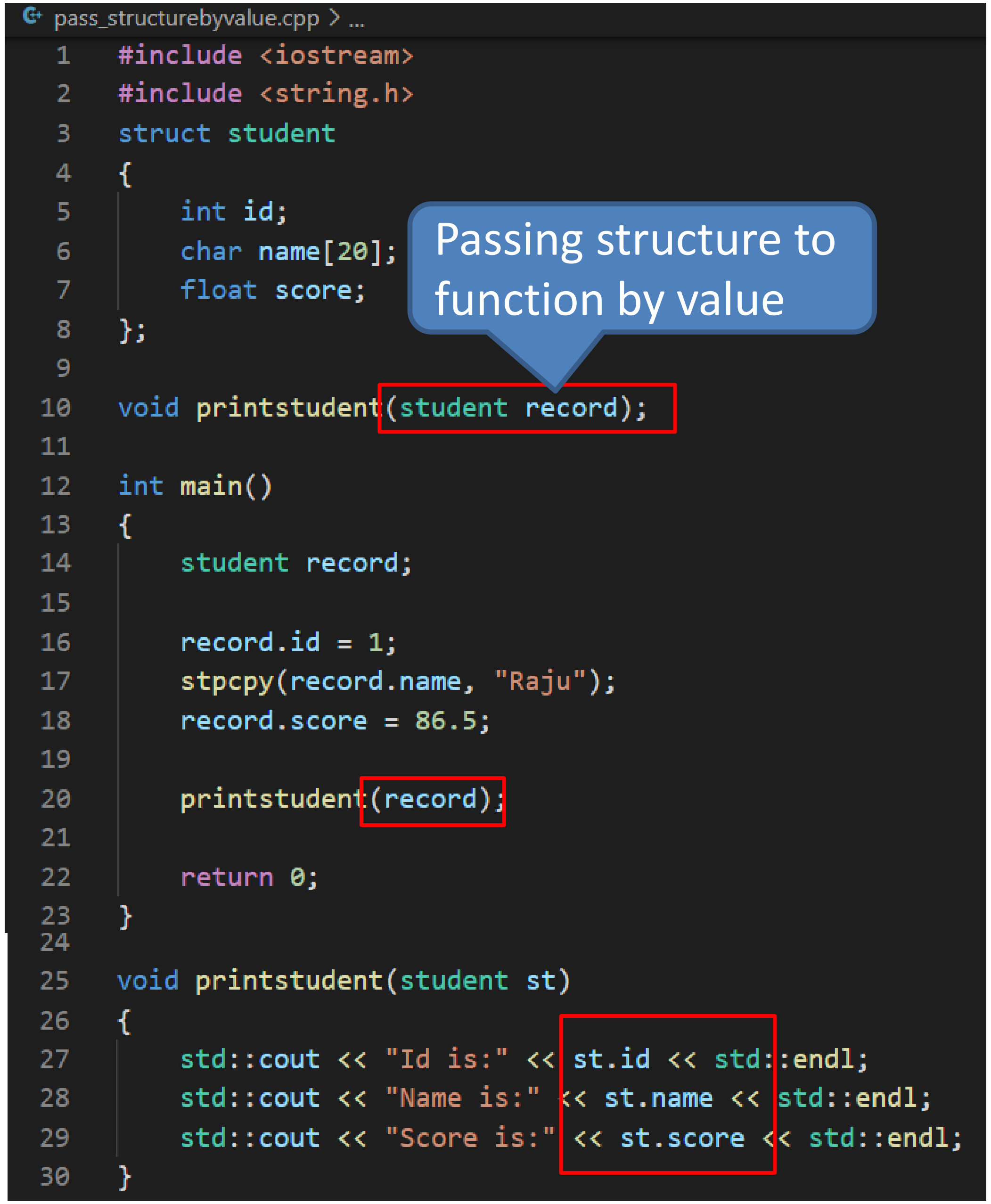
```cpp
#include <iostream>
#include <cstring>
using namespace std;

void mcopy(char *s, int m);

int main()
{
    char str[81];
    int m;
    cout << "Enter a string:\n";
    cin.getline(str,81);

    cout << "Enter the starting number you want to copy:\n";
    cin >> m;

    mcopy(str,m);

    cout << "The copied string is:" << str << endl;

    return 0;
}

void mcopy(char *s, int m)
{
    strcpy(s, s+m-1);
}
```

You can use **character array** or **pointer-to-char** as a parameter.



```
Enter a string:
Today is a sunny day.
Enter the starting number you want to copy:
3
The copied string is:day is a sunny day.
```

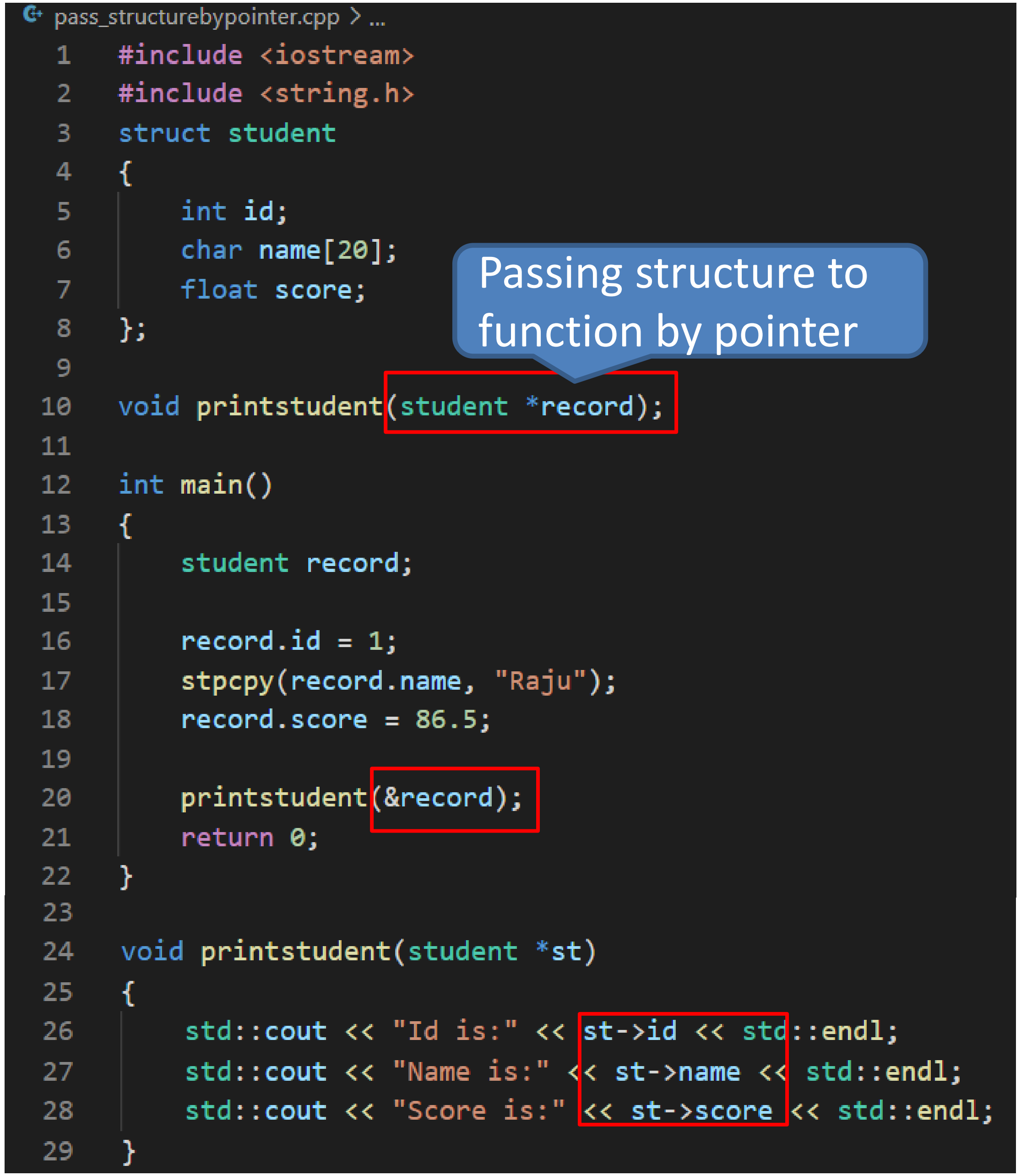
6. Passing structure to a function



pass_structurebyvalue.cpp > ...

```cpp
#include <iostream>
#include <string.h>
struct student
{
    int id;
    char name[20];
    float score;
};

void printstudent(student record);

int main()
{
    student record;

    record.id = 1;
    stpcpy(record.name, "Raju");
    record.score = 86.5;

    printstudent(record);

    return 0;
}

void printstudent(student st)
{
    std::cout << "Id is:" << st.id << std::endl;
    std::cout << "Name is:" << st.name << std::endl;
    std::cout << "Score is:" << st.score << std::endl;
}
```

Passing structure to function by value

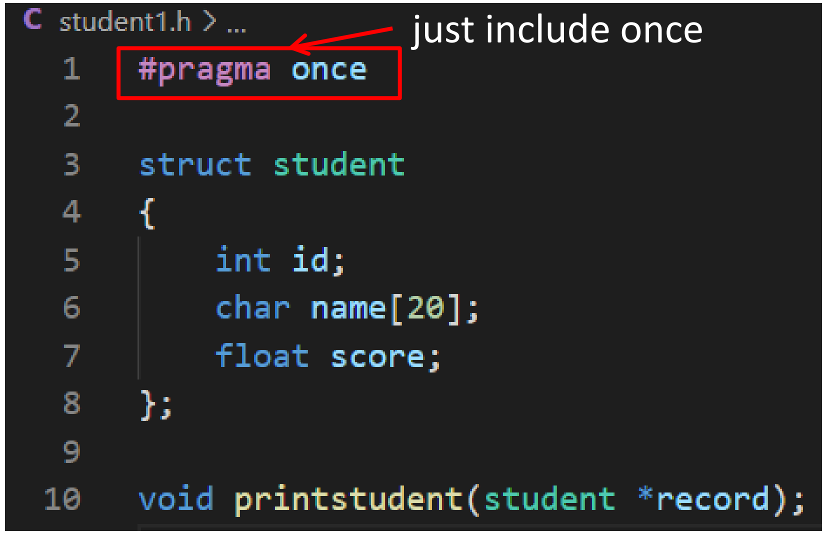


pass_structurebypointer.cpp > ...

```cpp
#include <iostream>
#include <string.h>
struct student
{
    int id;
    char name[20];
    float score;
};

void printstudent(student *record);

int main()
{
    student record;

    record.id = 1;
    stpcpy(record.name, "Raju");
    record.score = 86.5;

    printstudent(&record);
    return 0;
}

void printstudent(student *st)
{
    std::cout << "Id is:" << st->id << std::endl;
    std::cout << "Name is:" << st->name << std::endl;
    std::cout << "Score is:" << st->score << std::endl;
}
```

Passing structure to function by pointer

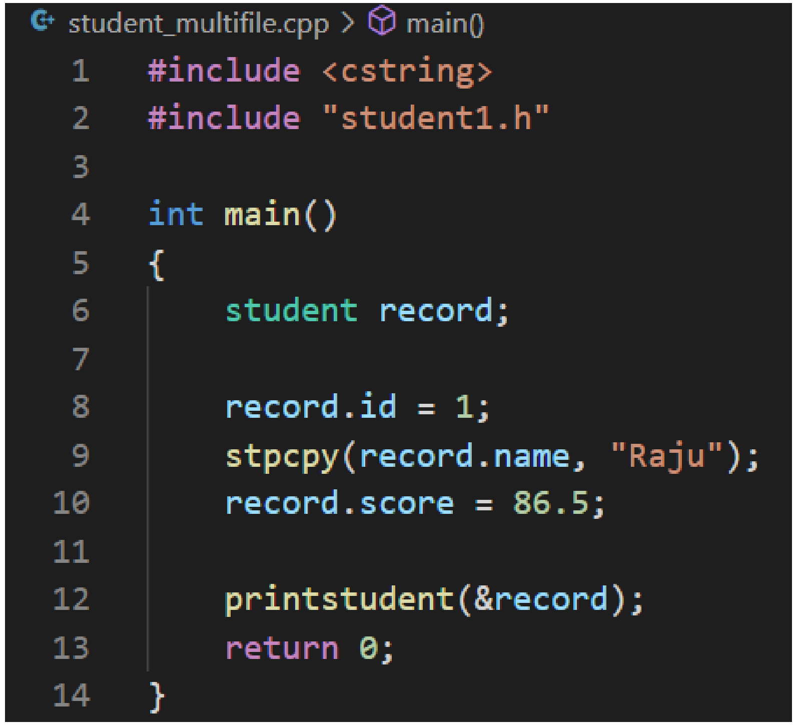
Multiple files



student1.h > ...

just include once

```cpp
#pragma once

struct student
{
    int id;
    char name[20];
    float score;
};

void printstudent(student *record);
```



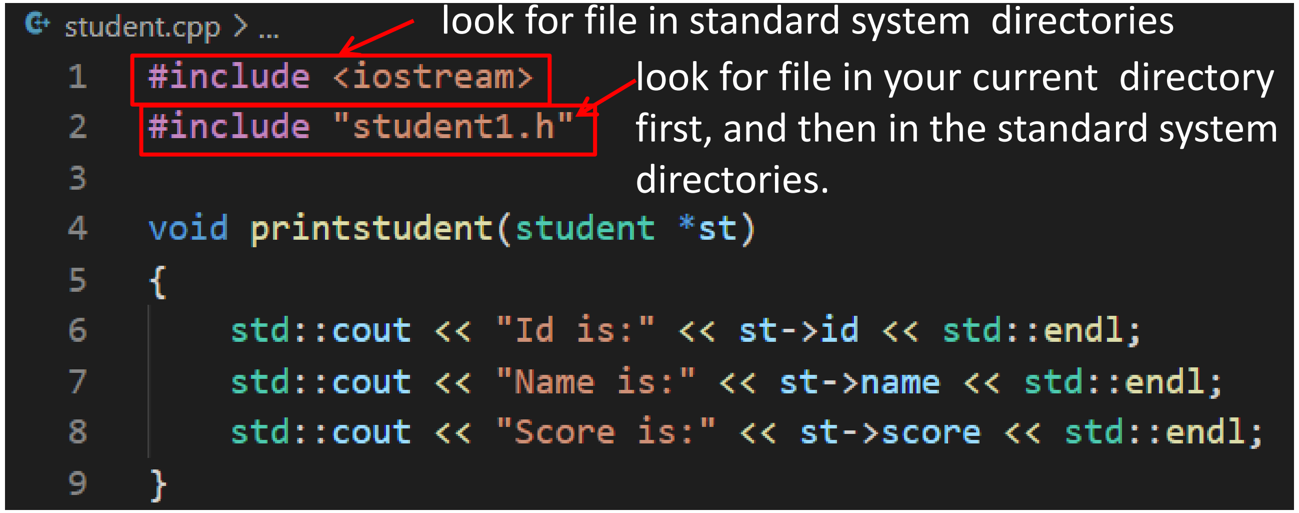
student_multifile.cpp > main()

```cpp
#include <cstring>
#include "student1.h"

int main()
{
    student record;

    record.id = 1;
    stpcpy(record.name, "Raju");
    record.score = 86.5;

    printstudent(&record);
    return 0;
}
```

Header file:

- const variable or macro definition
- structure declaration
- function prototype

When the preprocessor spots an **#include** directive, it looks for the following filename and includes the contents of that file within the current file.



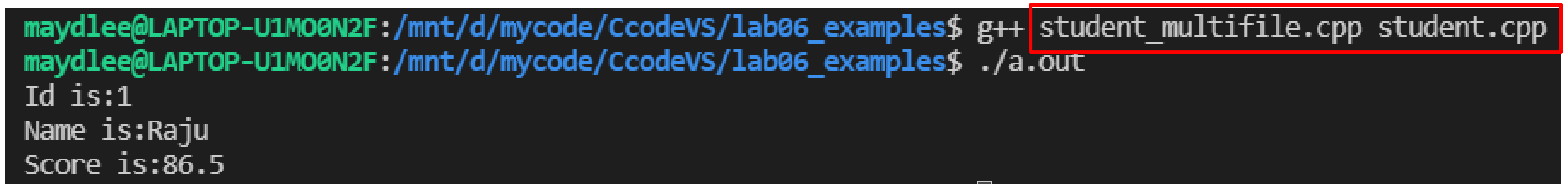
student.cpp > ...

look for file in standard system directories

look for file in your current directory first, and then in the standard system directories.

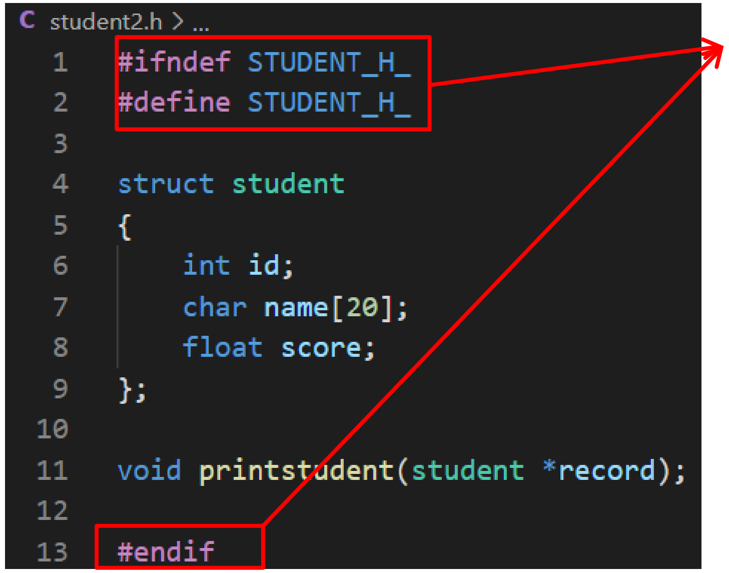
```cpp
#include <iostream>
#include "student1.h"

void printstudent(student *st)
{
    std::cout << "Id is:" << st->id << std::endl;
    std::cout << "Name is:" << st->name << std::endl;
    std::cout << "Score is:" << st->score << std::endl;
}
```

compile all the source files, with default executable name



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab06_examples$ g++ student_multifile.cpp student.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab06_examples$ ./a.out
Id is:1
Name is:Raju
Score is:86.5
```

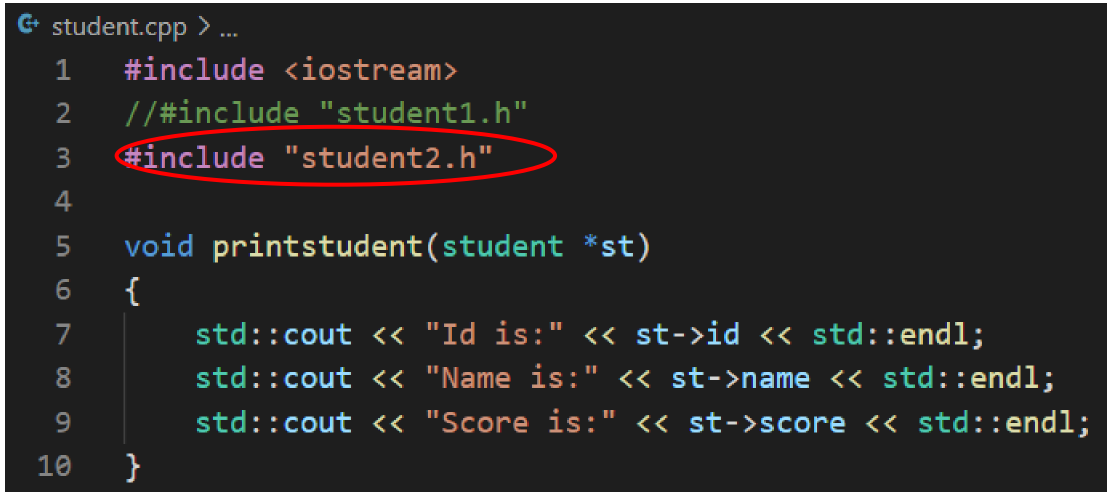
Multiple files



student2.h > ...

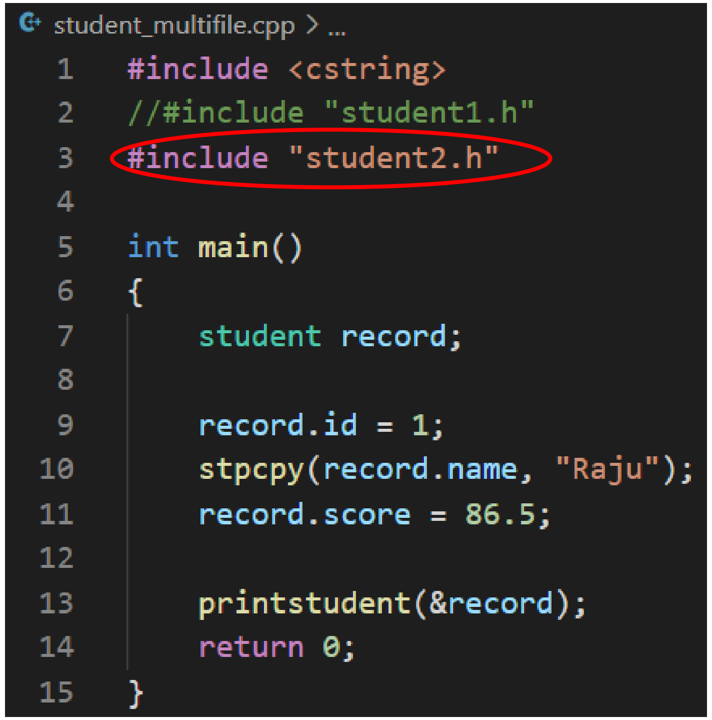
```
#ifndef STUDENT_H_
#define STUDENT_H_

struct student
{
    int id;
    char name[20];
    float score;
};

void printstudent(student *record);

#endif
```

Using conditional compilation directives to avoid duplicate including.



student.cpp > ...

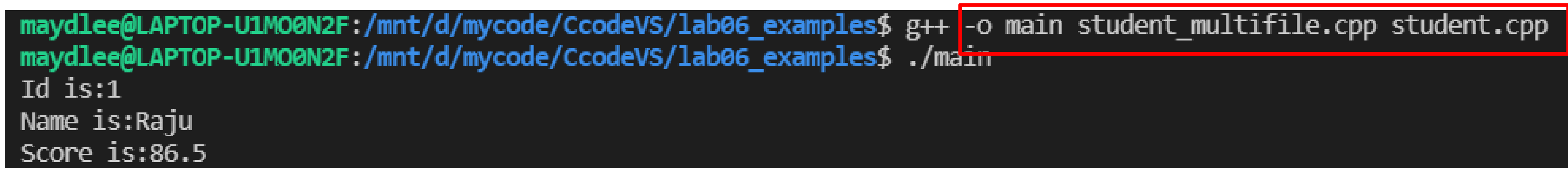
```
#include <iostream>
//#include "student1.h"
#include "student2.h"

void printstudent(student *st)
{
    std::cout << "Id is:" << st->id << std::endl;
    std::cout << "Name is:" << st->name << std::endl;
    std::cout << "Score is:" << st->score << std::endl;
}
```



student_multifile.cpp > ...

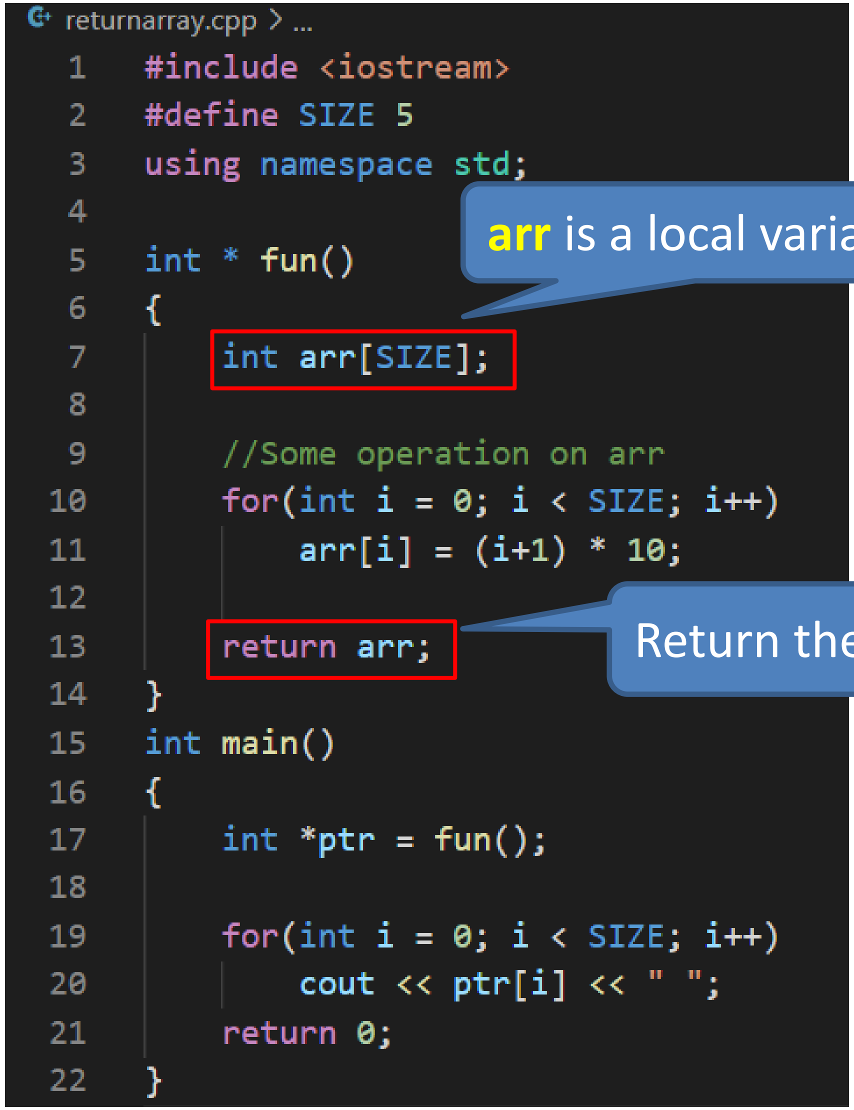
```
#include <cstring>
//#include "student1.h"
#include "student2.h"

int main()
{
    student record;

    record.id = 1;
    stpcpy(record.name, "Raju");
    record.score = 86.5;

    printstudent(&record);
    return 0;
}
```

compile all the source files, with a given executable name



```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab06_examples$ g++ -o main student_multifile.cpp student.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab06_examples$ ./main
Id is:1
Name is:Raju
Score is:86.5
```

7. Return an array (or a pointer) from a function

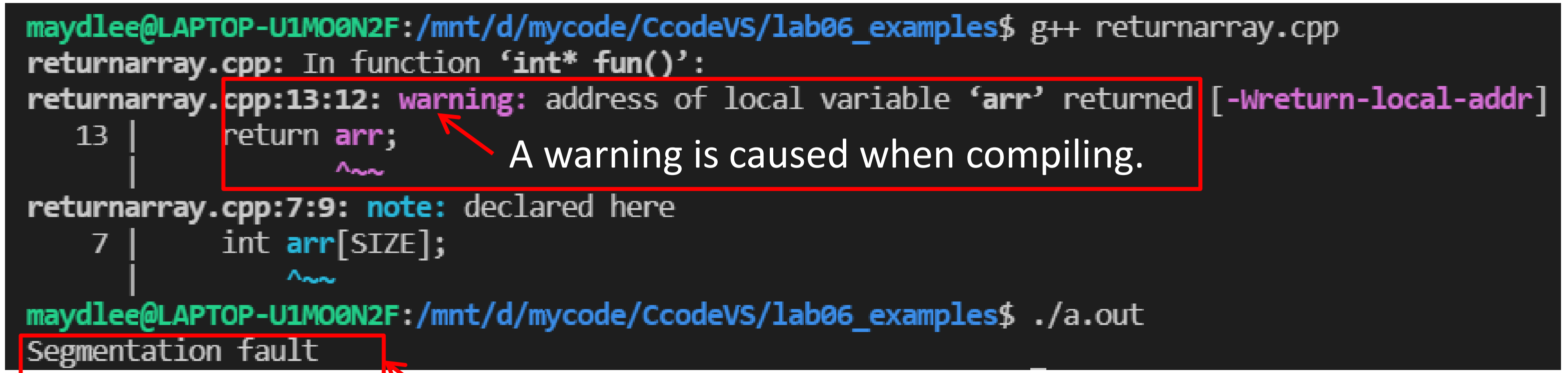


returnarray.cpp > ...

```cpp
#include <iostream>
#define SIZE 5
using namespace std;

int * fun()
{
    int arr[SIZE];

    //Some operation on arr
    for(int i = 0; i < SIZE; i++)
        arr[i] = (i+1) * 10;

    return arr;
}
int main()
{
    int *ptr = fun();

    for(int i = 0; i < SIZE; i++)
        cout << ptr[i] << " ";
    return 0;
}
```

arr is a local variable

Return the address of a local variable is wrong.



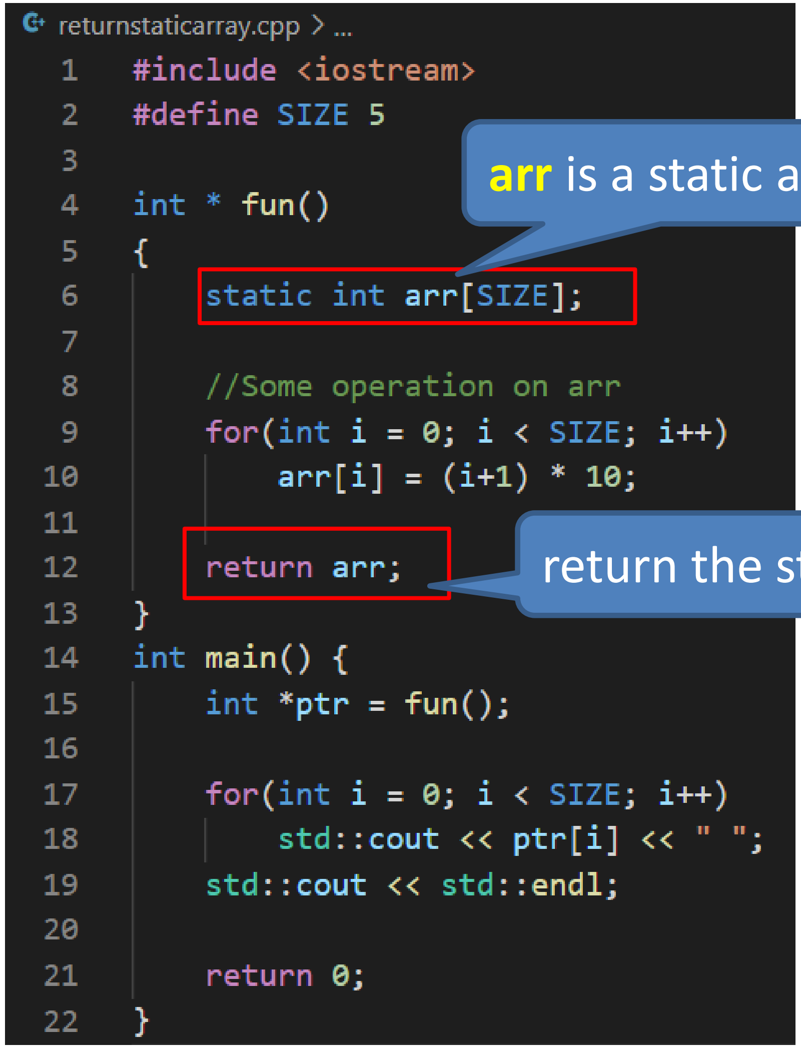
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab06_examples$ g++ returnarray.cpp
returnarray.cpp: In function ‘int* fun()’:
returnarray.cpp:13:12: warning: address of local variable ‘arr’ returned [-Wreturn-local-addr]
   13 |     return arr;
      |            ^~~
returnarray.cpp:7:9: note: declared here
    7 |     int arr[SIZE];
      |         ^~~
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab06_examples$ ./a.out
Segmentation fault
```

A warning is caused when compiling.

The program can not be executed.

Three correct ways of returning an array (or a pointer):

- Return a static array
- Return a dynamically allocated array (or a pointer)
- Return a parameter pointer

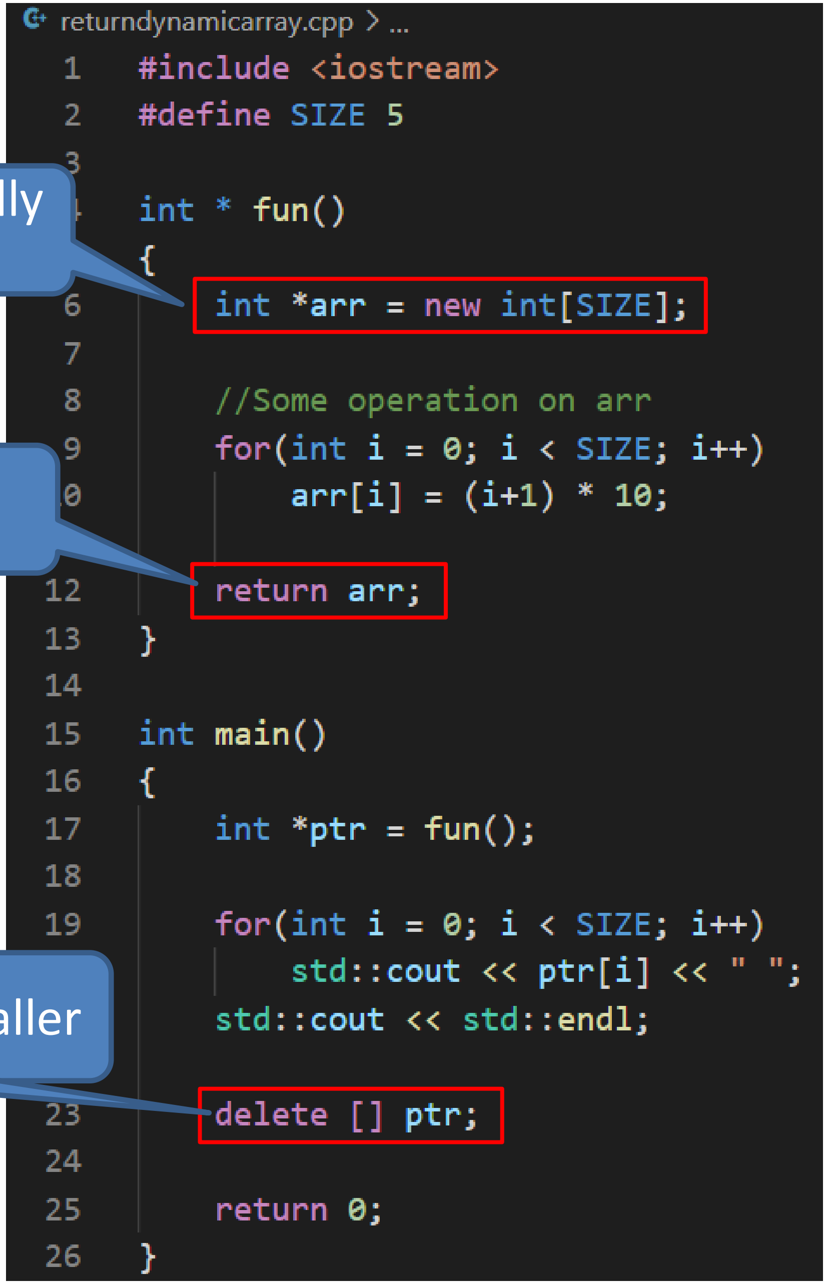


returnstaticarray.cpp > ...

```cpp
#include <iostream>
#define SIZE 5

int * fun()
{
    static int arr[SIZE];

    //Some operation on arr
    for(int i = 0; i < SIZE; i++)
        arr[i] = (i+1) * 10;

    return arr;
}
int main() {
    int *ptr = fun();

    for(int i = 0; i < SIZE; i++)
        std::cout << ptr[i] << " ";
    std::cout << std::endl;

    return 0;
}
```

arr is a static array

return the static arr



returndynamicarray.cpp > ...

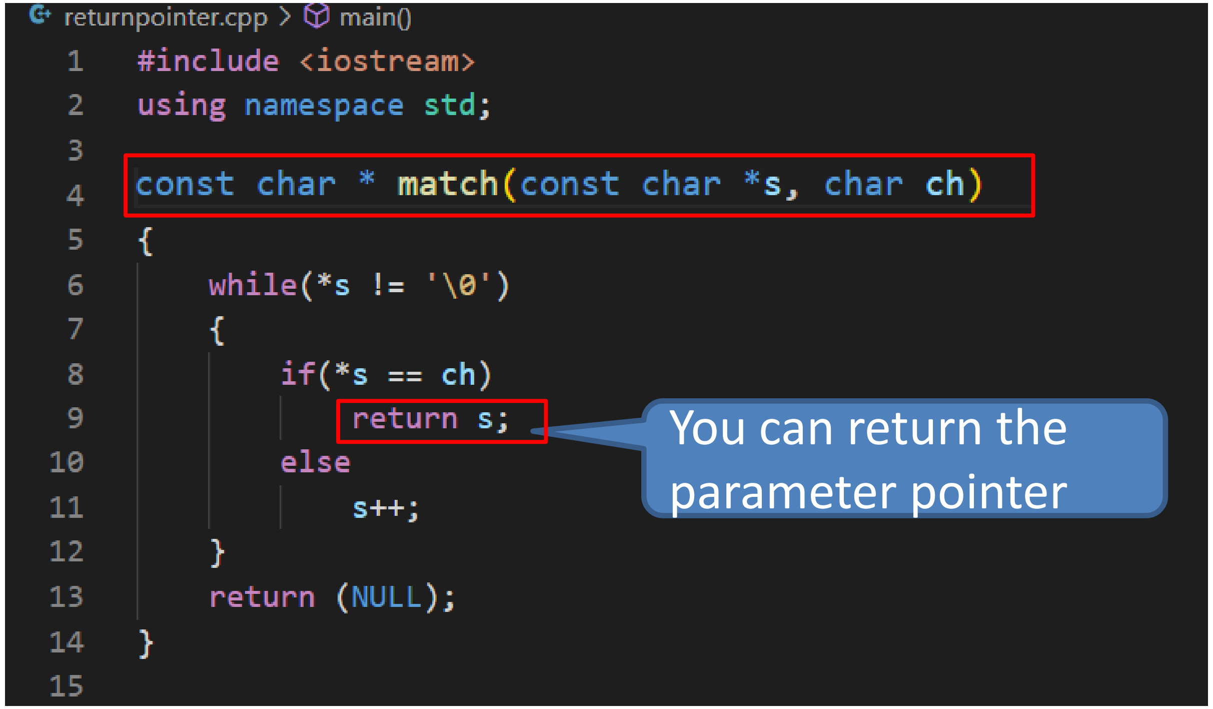
```cpp
#include <iostream>
#define SIZE 5

int * fun()
{
    int *arr = new int[SIZE];

    //Some operation on arr
    for(int i = 0; i < SIZE; i++)
        arr[i] = (i+1) * 10;

    return arr;
}

int main()
{
    int *ptr = fun();

    for(int i = 0; i < SIZE; i++)
        std::cout << ptr[i] << " ";
    std::cout << std::endl;

    delete [] ptr;

    return 0;
}
```

arr is a dynamically allocated array

return the dynamically allocated array arr

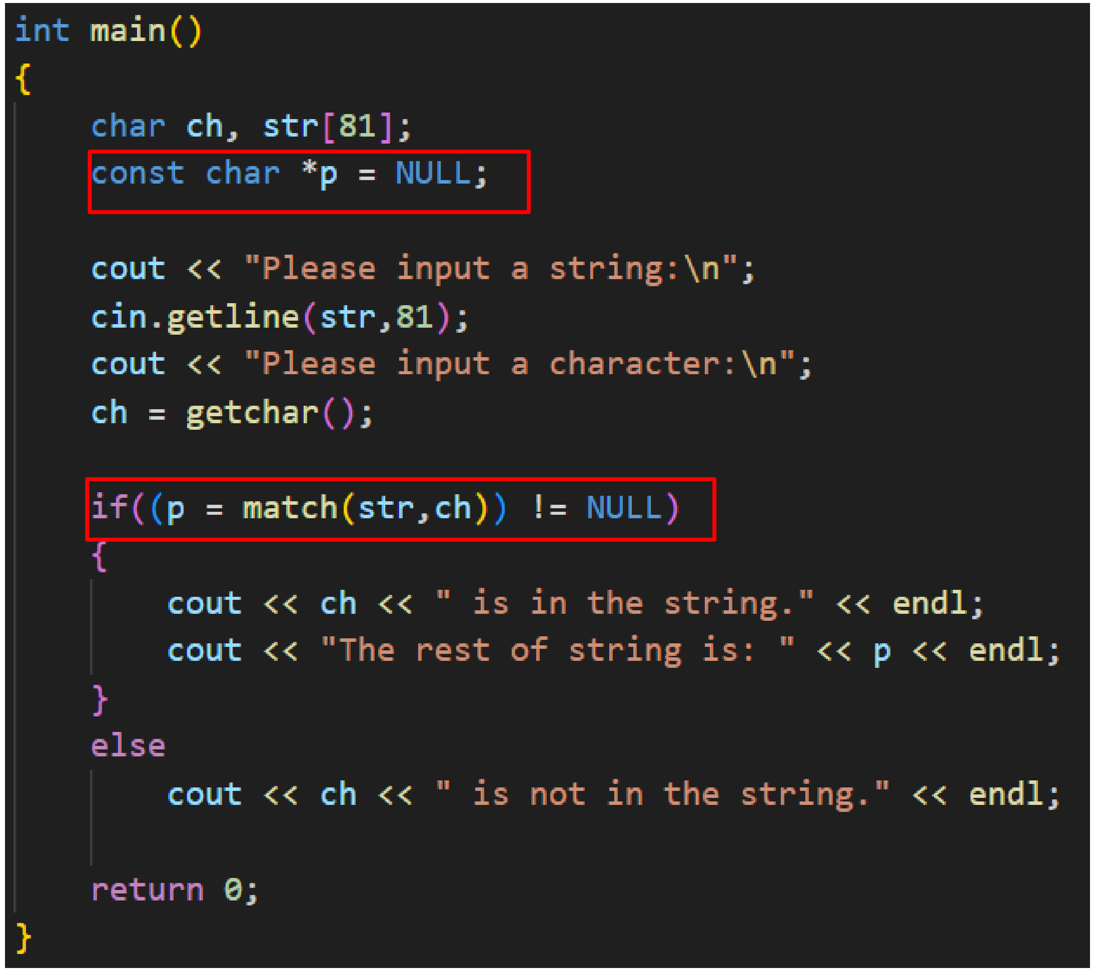
release the memory in caller

Return a parameter pointer

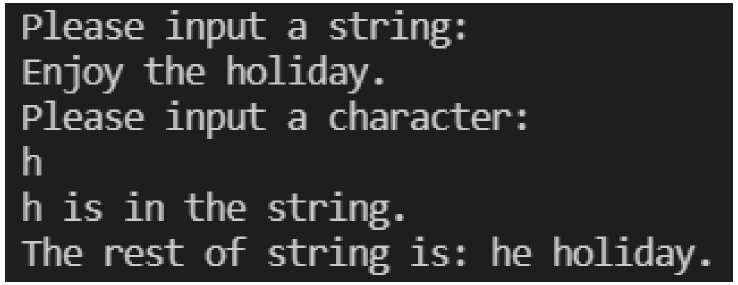


```cpp
#include <iostream>
using namespace std;

const char * match(const char *s, char ch)
{
    while(*s != '\0')
    {
        if(*s == ch)
            return s;
        else
            s++;
    }
    return (NULL);
}

```

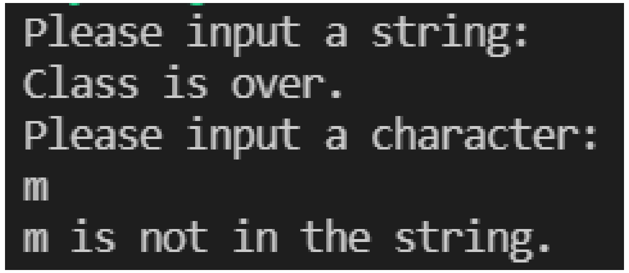
You can return the parameter pointer



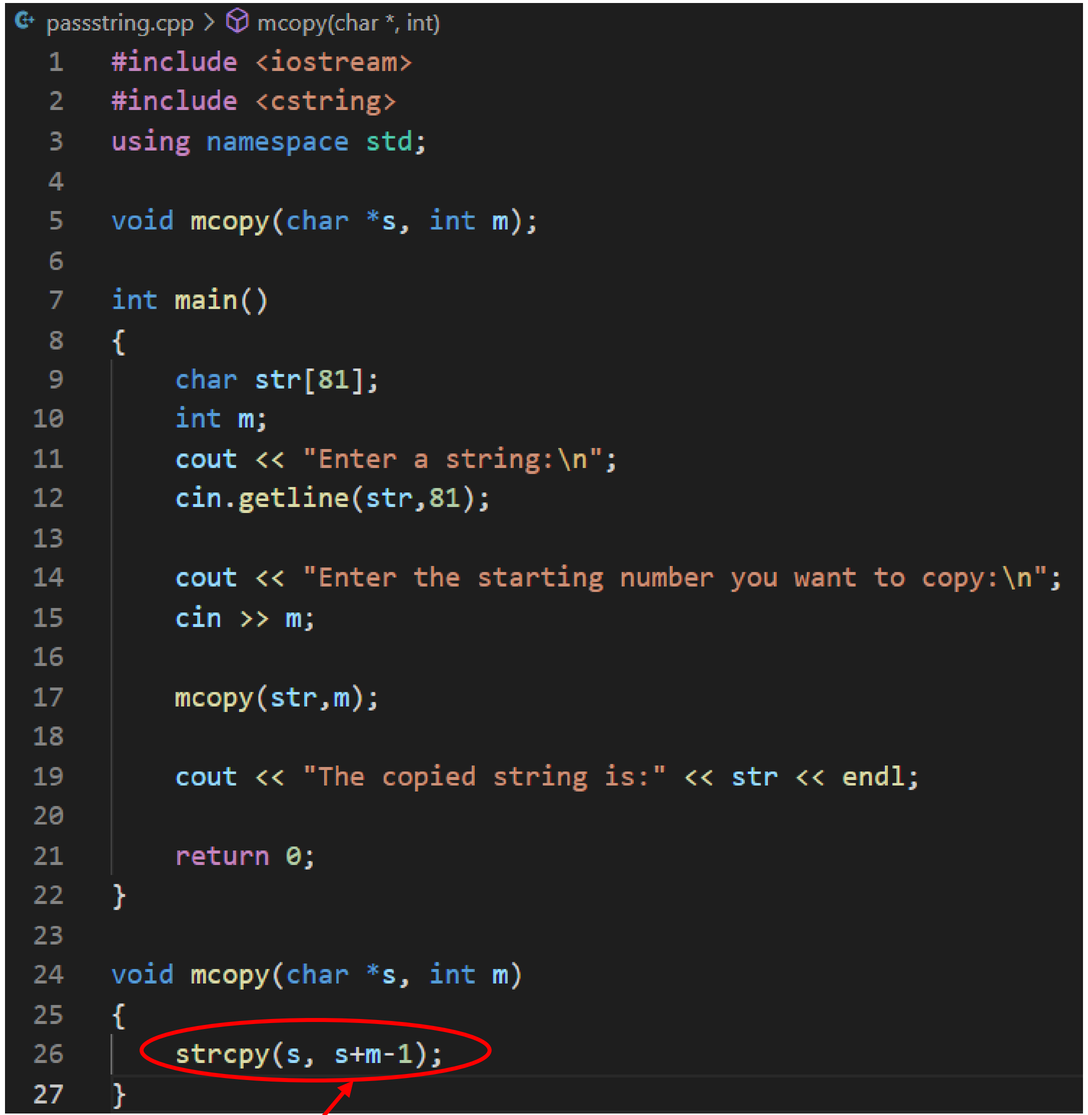
```cpp
int main()
{
    char ch, str[81];
    const char *p = NULL;

    cout << "Please input a string:\n";
    cin.getline(str,81);
    cout << "Please input a character:\n";
    ch = getchar();

    if((p = match(str,ch)) != NULL)
    {
        cout << ch << " is in the string." << endl;
        cout << "The rest of string is: " << p << endl;
    }
    else
        cout << ch << " is not in the string." << endl;

    return 0;
}
```



```
Please input a string:
Enjoy the holiday.
Please input a character:
h
h is in the string.
The rest of string is: he holiday.
```

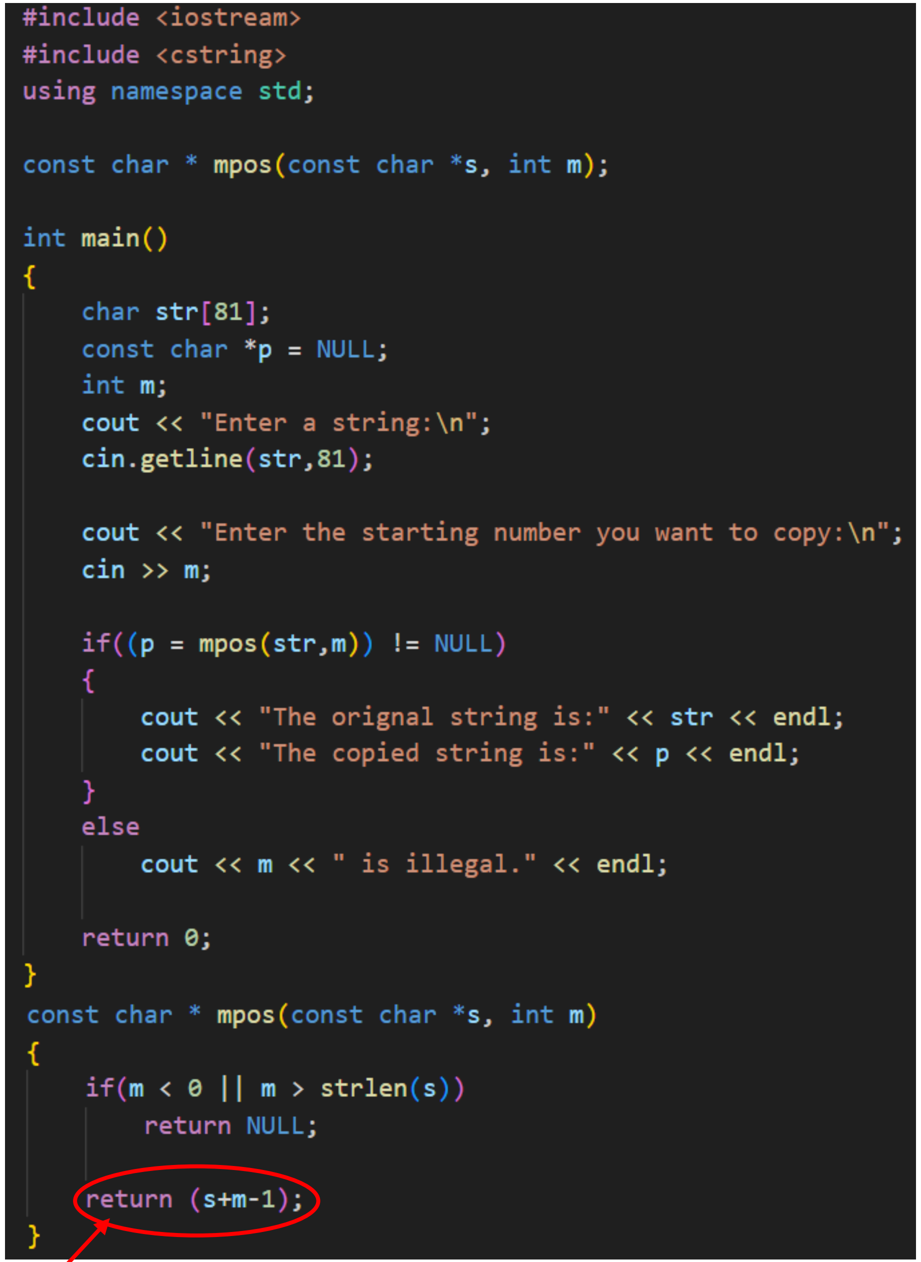


```
Please input a string:
Class is over.
Please input a character:
m
m is not in the string.
```



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

void mcopy(char *s, int m);

int main()
{
    char str[81];
    int m;
    cout << "Enter a string:\n";
    cin.getline(str,81);

    cout << "Enter the starting number you want to copy:\n";
    cin >> m;

    mcopy(str,m);

    cout << "The copied string is:" << str << endl;

    return 0;
}

void mcopy(char *s, int m)
{
    strcpy(s, s+m-1);
}
```

Modify the contents of the array, need not return value.

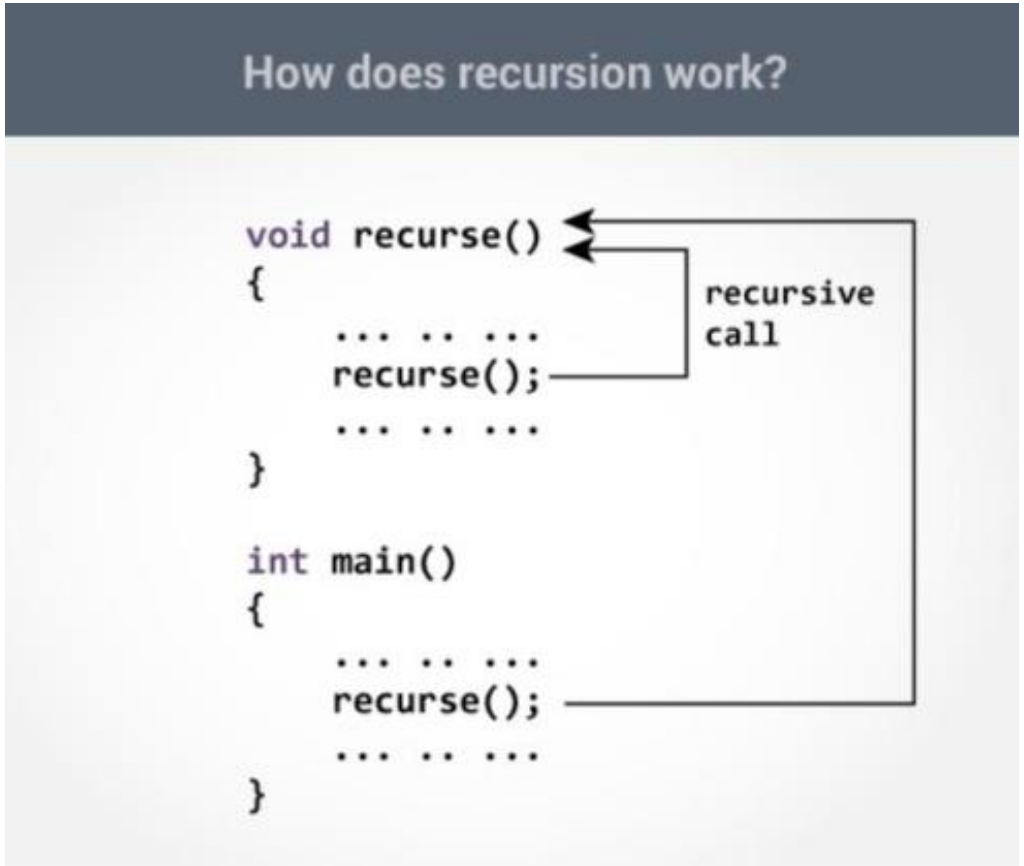


```cpp
#include <iostream>
#include <cstring>
using namespace std;

const char * mpos(const char *s, int m);

int main()
{
    char str[81];
    const char *p = NULL;
    int m;
    cout << "Enter a string:\n";
    cin.getline(str,81);

    cout << "Enter the starting number you want to copy:\n";
    cin >> m;

    if((p = mpos(str,m)) != NULL)
    {
        cout << "The orignal string is:" << str << endl;
        cout << "The copied string is:" << p << endl;
    }
    else
        cout << m << " is illegal." << endl;

    return 0;
}
const char * mpos(const char *s, int m)
{
    if(m < 0 || m > strlen(s))
        return NULL;

    return (s+m-1);
}
```

Don not modify the contents of the array, return the proper position(pointer).

2.2 Recursive function

A function that **calls itself** is known as **recursive function**. And, this technique is known as **recursion**.



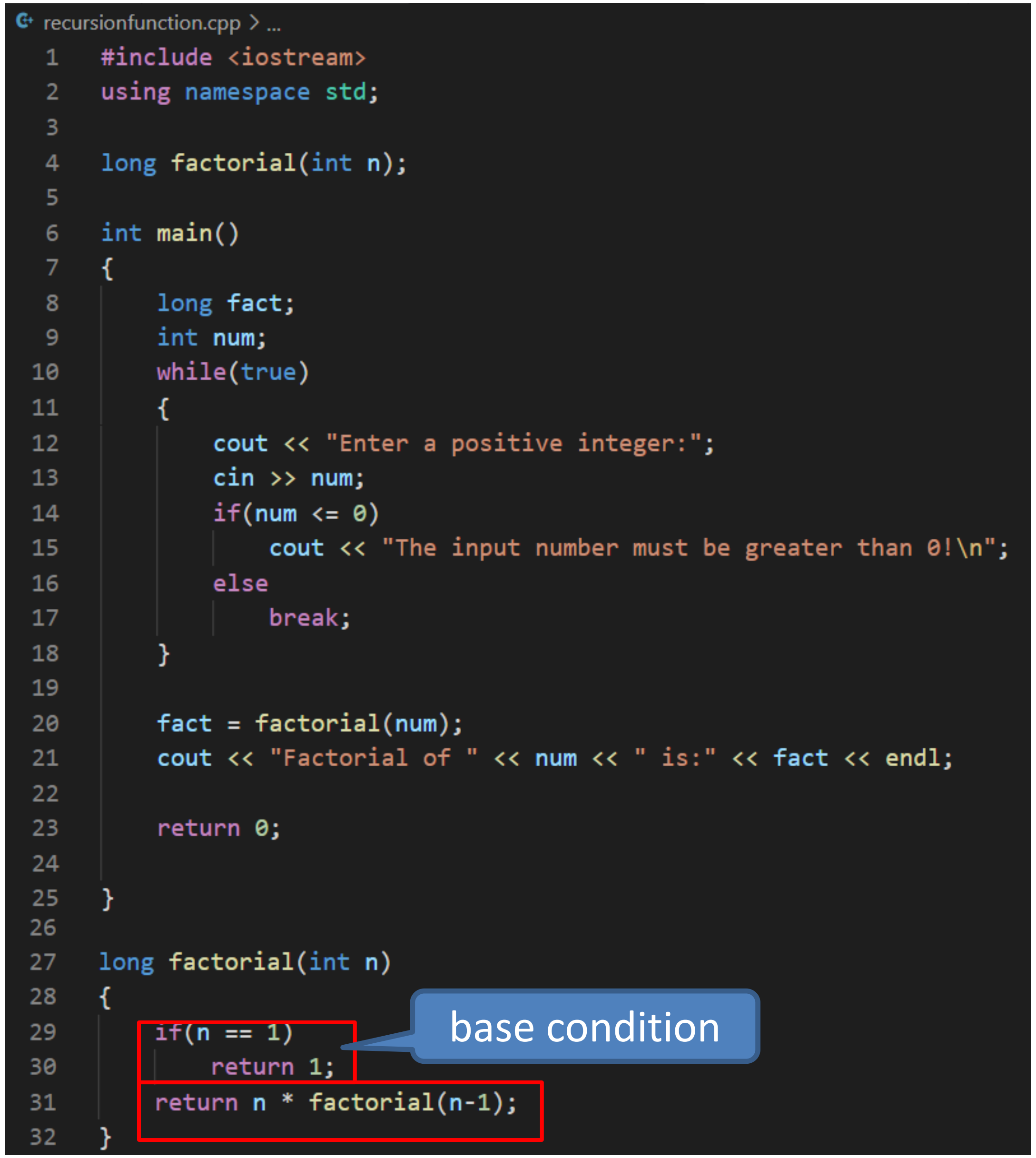
How does recursion work?

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

Recursion is used to solve various mathematical problems by dividing it into smaller problems.

Example: compute factorial with recursive function

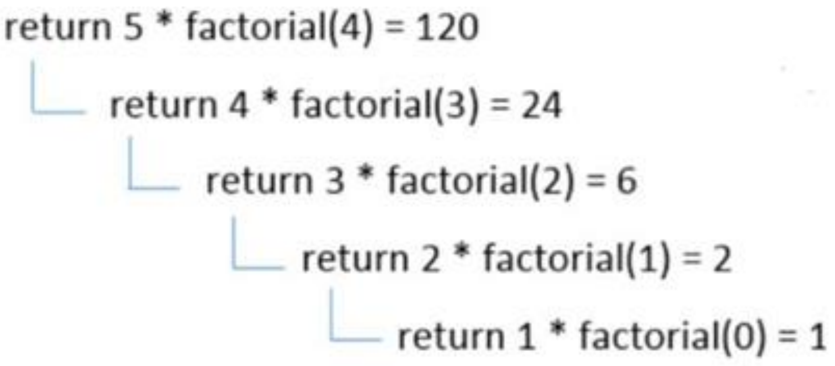
Compute factorial of a number Factorial of n = 1*2*3…*n



recursionfunction.cpp > ...

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

long factorial(int n);

int main()
{
    long fact;
    int num;
    while(true)
    {
        cout << "Enter a positive integer:";
        cin >> num;
        if(num <= 0)
            cout << "The input number must be greater than 0!\n";
        else
            break;
    }

    fact = factorial(num);
    cout << "Factorial of " << num << " is:" << fact << endl;

    return 0;

}

long factorial(int n)
{
    if(n == 1)
        return 1;
    return n * factorial(n-1);
}
```

- Factorial function: f(n) = n*f(n-1),
- base condition: if n<=1 then f(n) = 1



return 5 * factorial(4) = 120

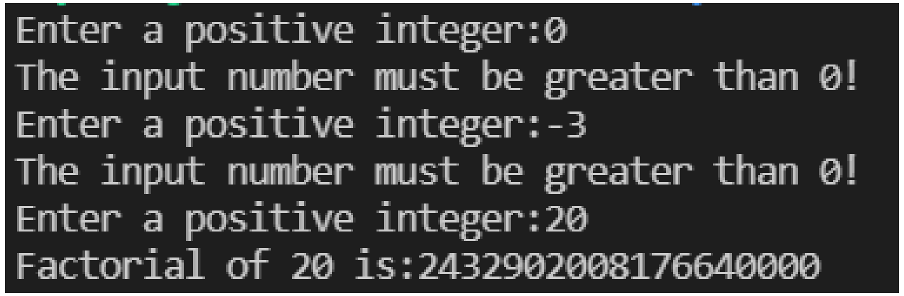
return 4 * factorial(3) = 24

return 3 * factorial(2) = 6

return 2 * factorial(1) = 2

return 1 * factorial(0) = 1

Calling itself until the function reaches to the **base condition!**



```
Enter a positive integer:0
The input number must be greater than 0!
Enter a positive integer:-3
The input number must be greater than 0!
Enter a positive integer:20
Factorial of 20 is:2432902008176640000
```

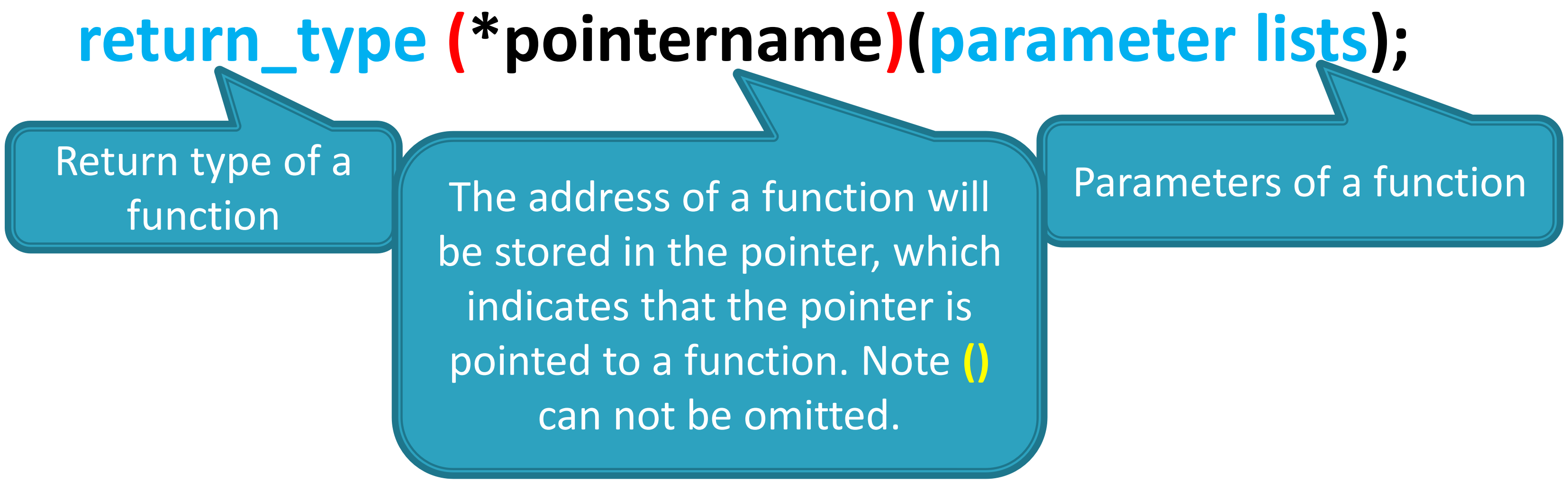
Disadvantages of recursion:

- **Recursive programs are generally slower than nonrecursive programs**. Because it needs to make a function call so the program must save all its current state and retrieve them again later. This consumes more time making recursive programs slower.

- **Recursive programs requires more memory to hold intermediate states in a stack**. Non recursive programs don’t have any intermediate states, hence they don’t require any extra memory.

2.4 Pointers to Functions(Function Pointer)

Declare a pointer to a function:



return_type (*pointername)(parameter lists);

Return type of a function

The address of a function will be stored in the pointer, which indicates that the pointer is pointed to a function. Note () can not be omitted.

Parameters of a function

Example:

`int findmax(int, int);` ← Declaring a function

`int   (*funptr)(int,int);` ← Declaring a pointer to a function

`funptr = findmax;` ← Assigning the address of a function to the pointer

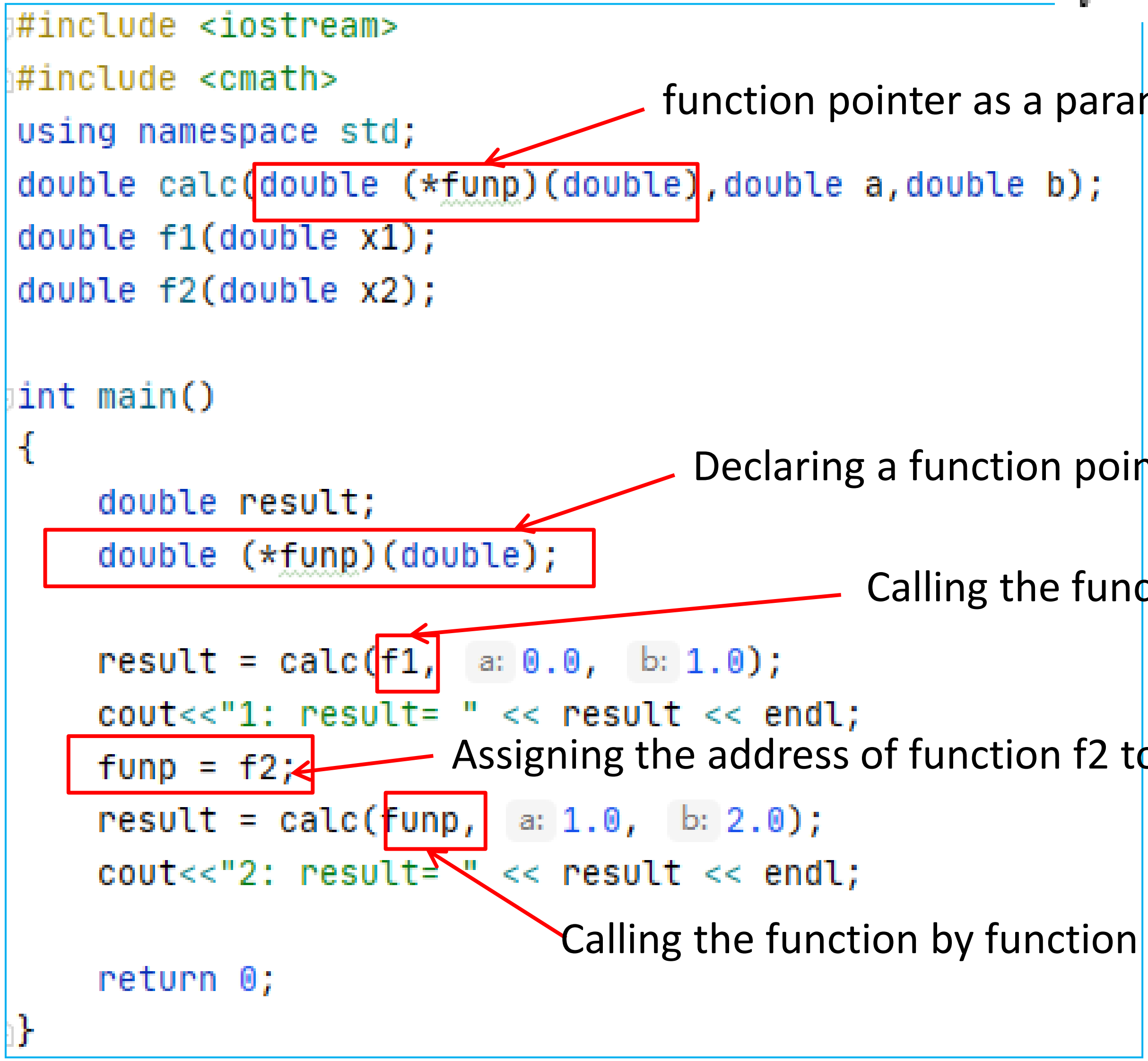
`int max = funptr(3,5);` ← Calling the function by the pointer

Example:

Compute the definite integral, suppose calculate the following definite integrals

$$\int_b^a f(x)dx=(b-a)/2*(f(a)+f(b))$$

$$\int_0^1 x^2\,dx \qquad \int_1^2 \mathrm{sinx/x}\,dx$$



```
#include <iostream>
#include <cmath>
using namespace std;
double calc(double (*funp)(double),double a,double b);
double f1(double x1);
double f2(double x2);

int main()
{
    double result;
    double (*funp)(double);

    result = calc(f1, a: 0.0, b: 1.0);
    cout<<"1: result= " << result << endl;
    funp = f2;
    result = calc(funp, a: 1.0, b: 2.0);
    cout<<"2: result= " << result << endl;

    return 0;
}
```

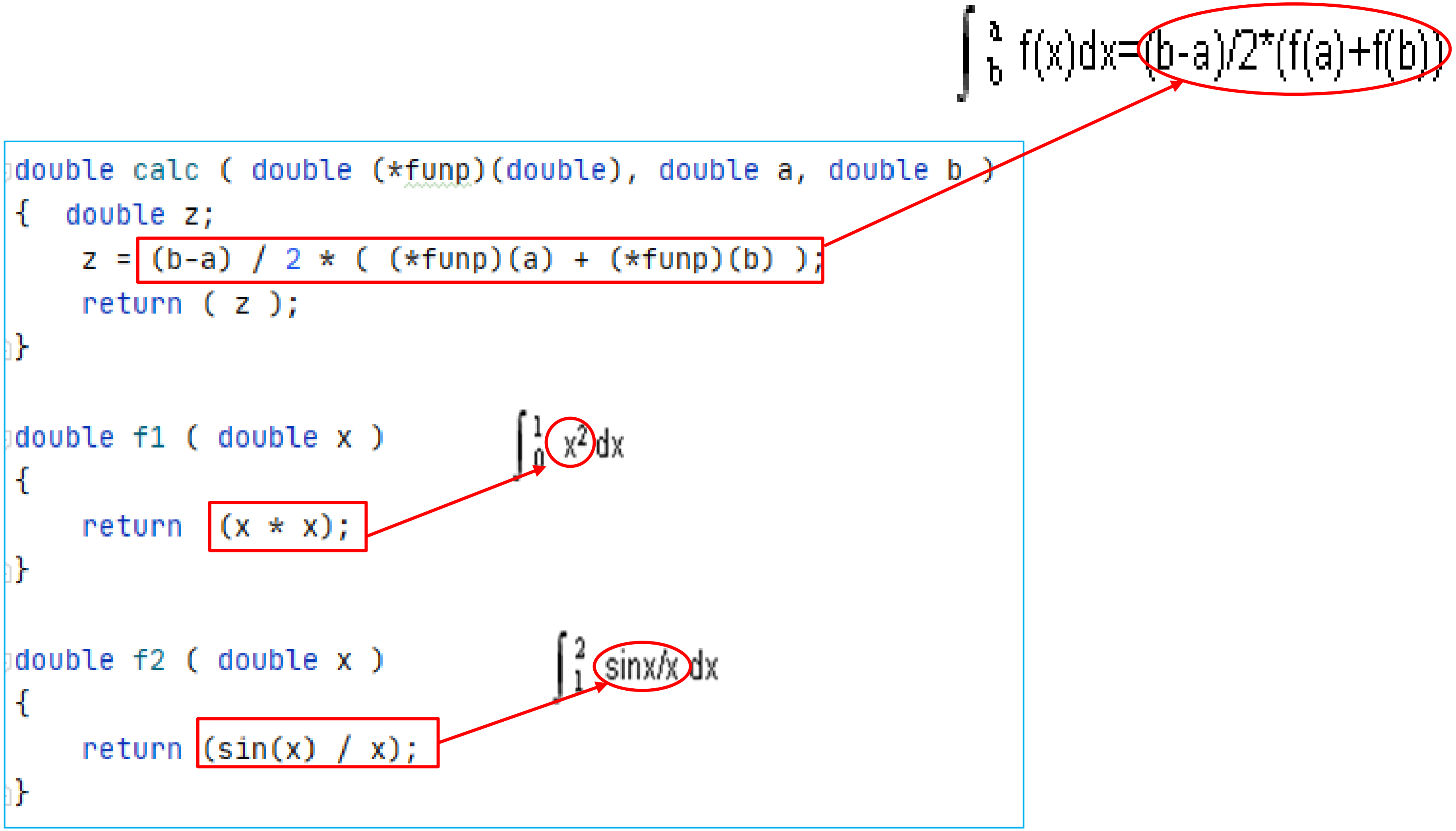
function pointer as a parameter

Declaring a function pointer

Calling the function by function name

Assigning the address of function f2 to the pointer

Calling the function by function pointer



$$\int_b^a f(x)dx=(b-a)/2*(f(a)+f(b))$$

```
double calc ( double (*funp)(double), double a, double b )
{  double z;
    z = (b-a) / 2 * ( (*funp)(a) + (*funp)(b) );
    return ( z );
}
```

```
double f1 ( double x )
{
    return  (x * x);
}
```

$$\int_0^1 x^2 dx$$

```
double f2 ( double x )
{
    return (sin(x) / x);
}
```

$$\int_1^2 \sin x/x\, dx$$

Output:

```
1: result= 0.5
2: result= 0.64806
```

qsort() in general utilities library stdlib.h

The quick sort method is one of the most effective sorting algorithms. **qsort()** function sorts an array of data object.

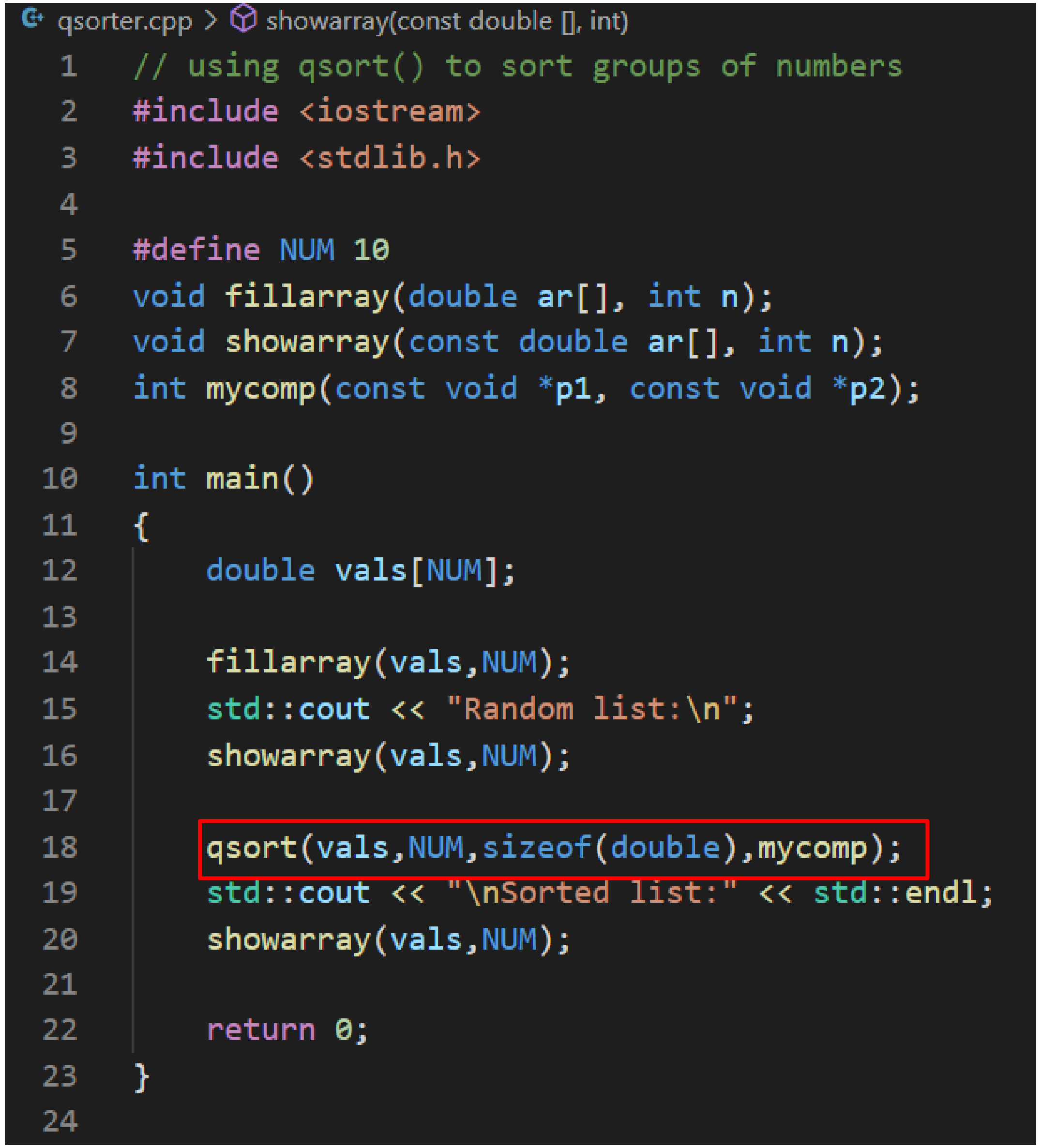
```
void qsort(void *base, size_t nmemb, size_t size, int(*compar)(const void *, const void *));
```

void *base: pointer to the beginning of the array to be sorted, it permits any data pointer type to be typecast to a pointer-to-void.

size_t nmemb: number of items to be sorted.

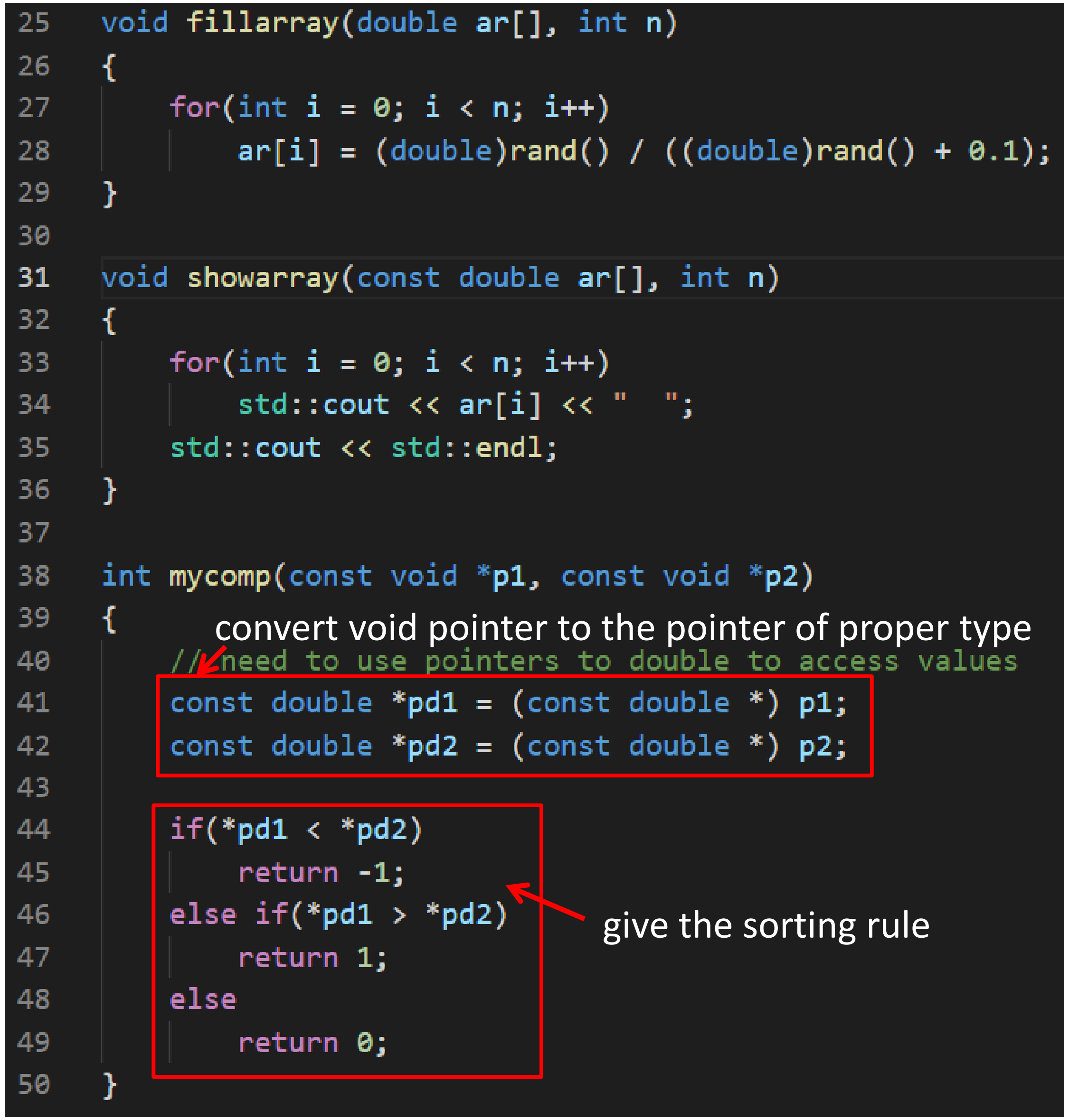
size_t size: the size of the data object, for example, if you want to sort an array of double, you would sizeof(double).

int (*compar)(const void *, const void *): a pointer to a function that returns an **int** and take two arguments, each of which is a pointer to type const void. These two pointers point to the items being compared.



qsorter.cpp > showarray(const double [], int)

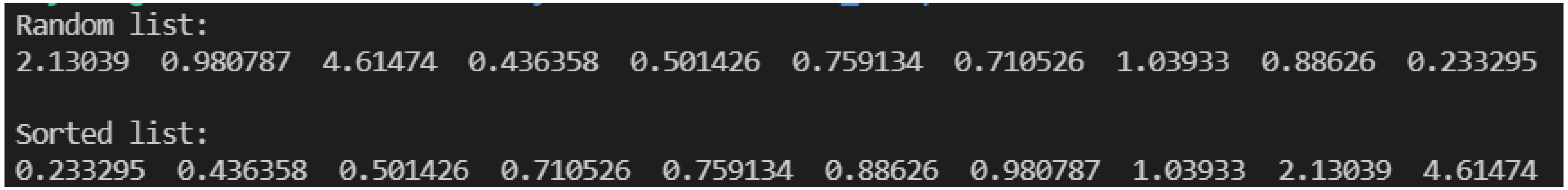
```cpp
// using qsort() to sort groups of numbers
#include <iostream>
#include <stdlib.h>

#define NUM 10
void fillarray(double ar[], int n);
void showarray(const double ar[], int n);
int mycomp(const void *p1, const void *p2);

int main()
{
    double vals[NUM];

    fillarray(vals,NUM);
    std::cout << "Random list:\n";
    showarray(vals,NUM);

    qsort(vals,NUM,sizeof(double),mycomp);
    std::cout << "\nSorted list:" << std::endl;
    showarray(vals,NUM);

    return 0;
}

```



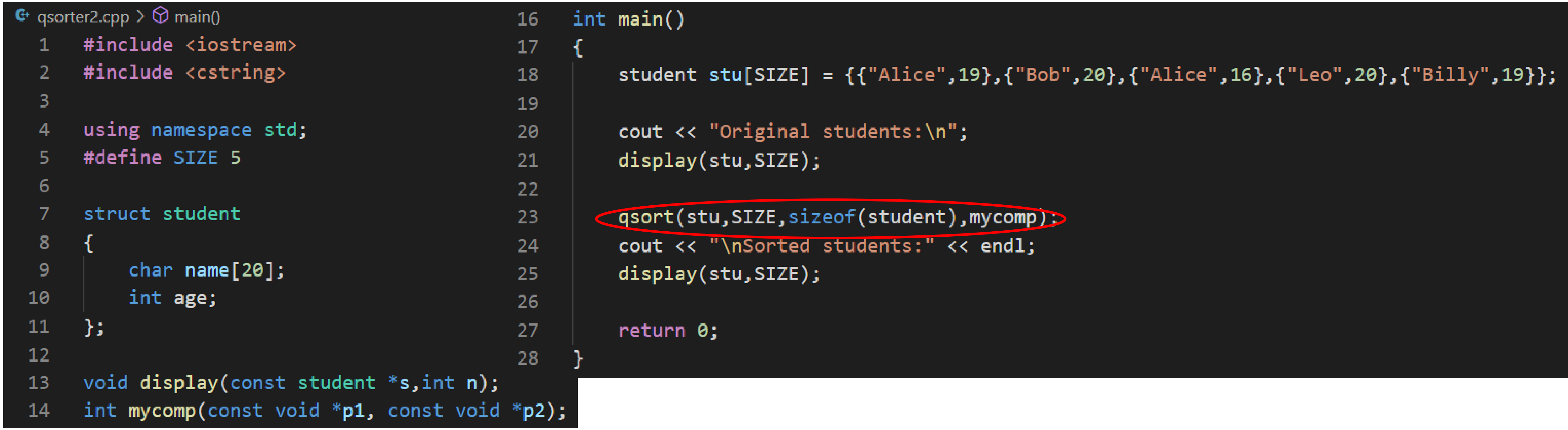
```cpp
void fillarray(double ar[], int n)
{
    for(int i = 0; i < n; i++)
        ar[i] = (double)rand() / ((double)rand() + 0.1);
}

void showarray(const double ar[], int n)
{
    for(int i = 0; i < n; i++)
        std::cout << ar[i] << "  ";
    std::cout << std::endl;
}

int mycomp(const void *p1, const void *p2)
{
    //need to use pointers to double to access values
    const double *pd1 = (const double *) p1;
    const double *pd2 = (const double *) p2;

    if(*pd1 < *pd2)
        return -1;
    else if(*pd1 > *pd2)
        return 1;
    else
        return 0;
}
```

convert void pointer to the pointer of proper type

give the sorting rule



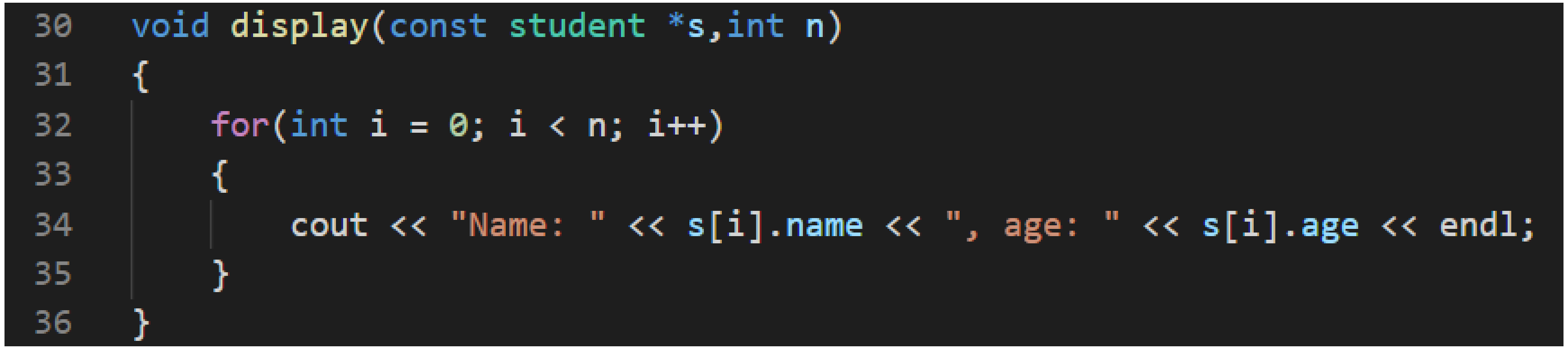
```
Random list:
2.13039  0.980787  4.61474  0.436358  0.501426  0.759134  0.710526  1.03933  0.88626  0.233295

Sorted list:
0.233295  0.436358  0.501426  0.710526  0.759134  0.88626  0.980787  1.03933  2.13039  4.61474
```



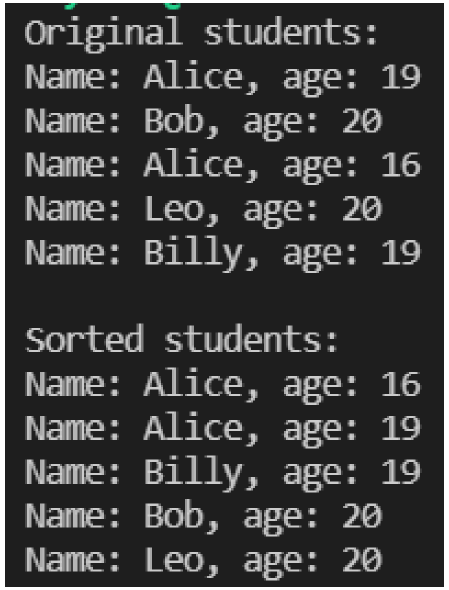
qsorter2.cpp > main()

```cpp
#include <iostream>
#include <cstring>

using namespace std;
#define SIZE 5

struct student
{
    char name[20];
    int age;
};

void display(const student *s,int n);
int mycomp(const void *p1, const void *p2);
```

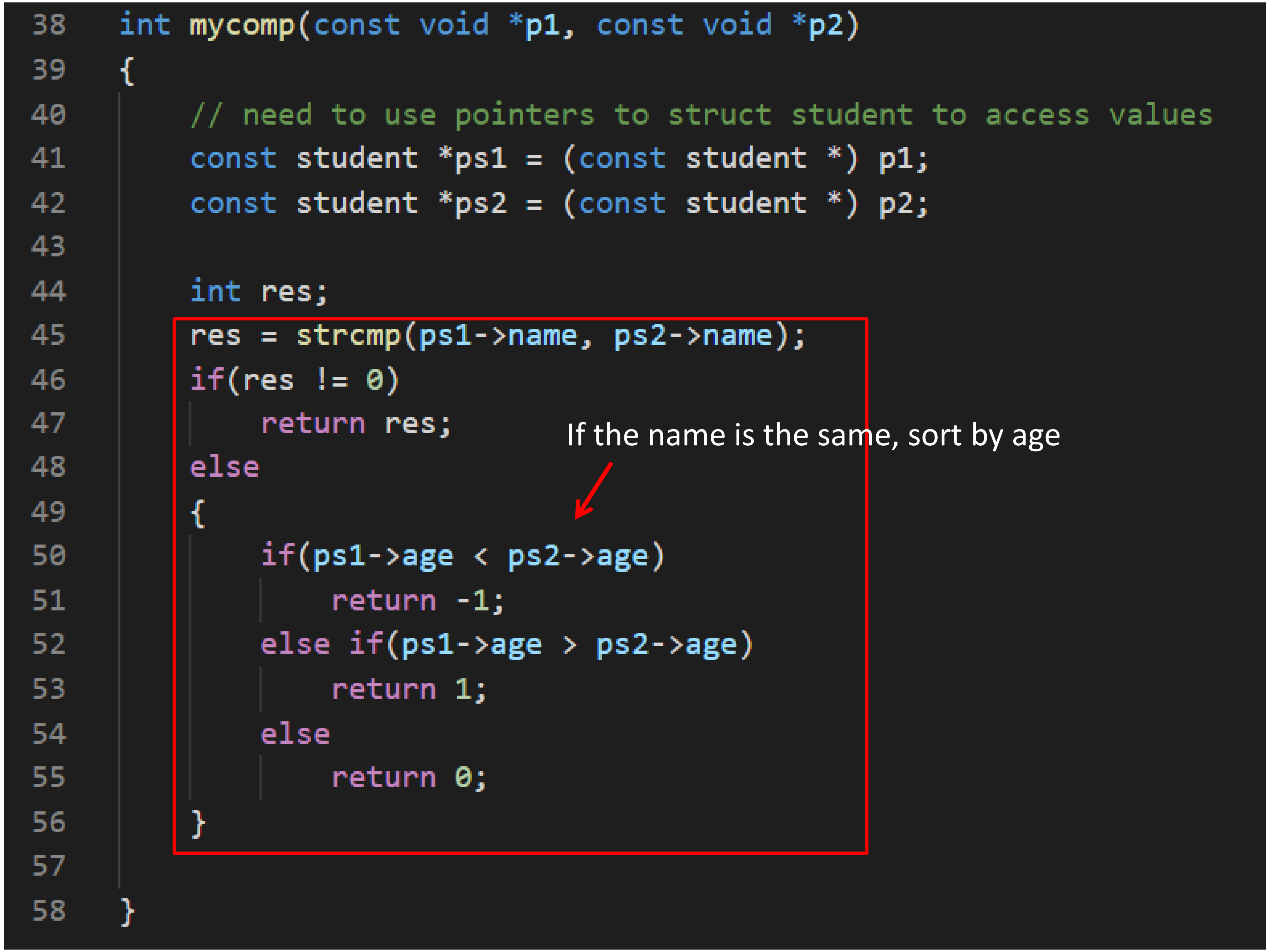
```cpp
int main()
{
    student stu[SIZE] = {{"Alice",19},{"Bob",20},{"Alice",16},{"Leo",20},{"Billy",19}};

    cout << "Original students:\n";
    display(stu,SIZE);

    qsort(stu,SIZE,sizeof(student),mycomp);
    cout << "\nSorted students:" << endl;
    display(stu,SIZE);

    return 0;
}
```



```cpp
void display(const student *s,int n)
{
    for(int i = 0; i < n; i++)
    {
        cout << "Name: " << s[i].name << ", age: " << s[i].age << endl;
    }
}
```



```
Original students:
Name: Alice, age: 19
Name: Bob, age: 20
Name: Alice, age: 16
Name: Leo, age: 20
Name: Billy, age: 19

Sorted students:
Name: Alice, age: 16
Name: Alice, age: 19
Name: Billy, age: 19
Name: Bob, age: 20
Name: Leo, age: 20
```

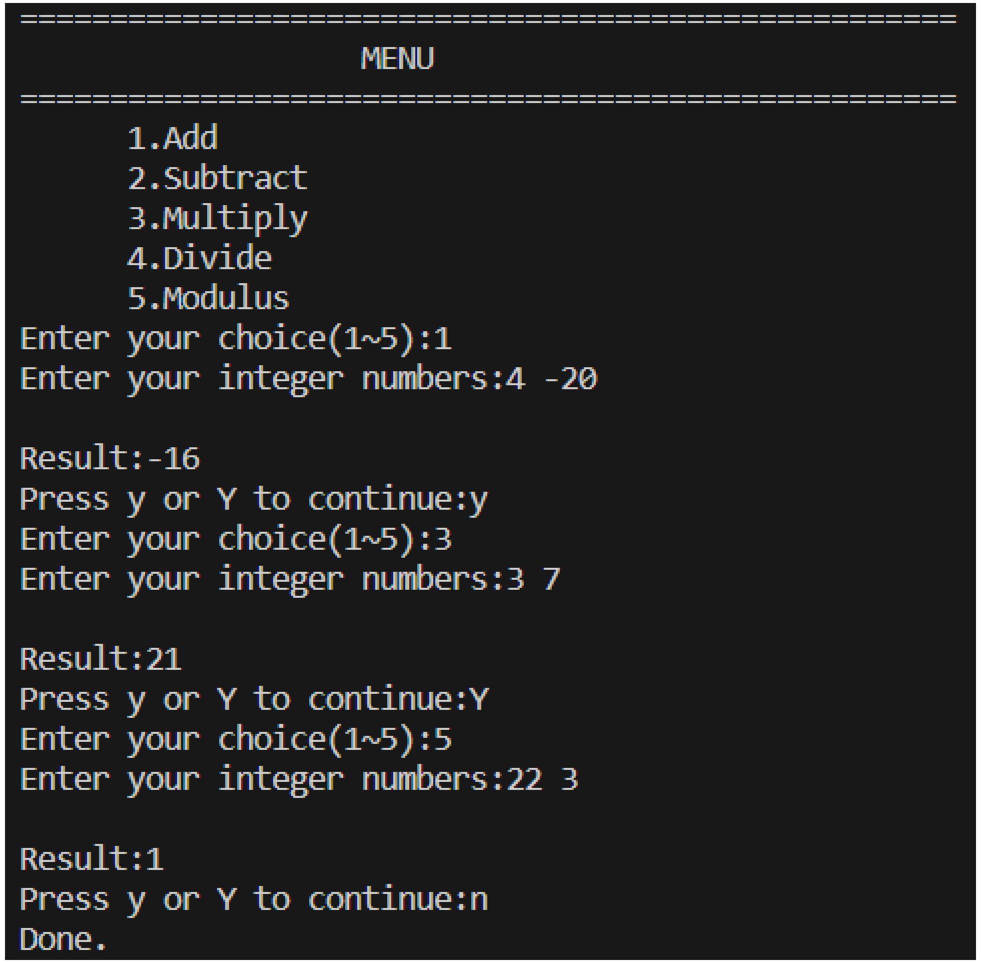


```c
int mycomp(const void *p1, const void *p2)
{
    // need to use pointers to struct student to access values
    const student *ps1 = (const student *) p1;
    const student *ps2 = (const student *) p2;

    int res;
    res = strcmp(ps1->name, ps2->name);
    if(res != 0)
        return res;
    else
    {
        if(ps1->age < ps2->age)
            return -1;
        else if(ps1->age > ps2->age)
            return 1;
        else
            return 0;
    }

}
```

If the name is the same, sort by age

3 Exercises

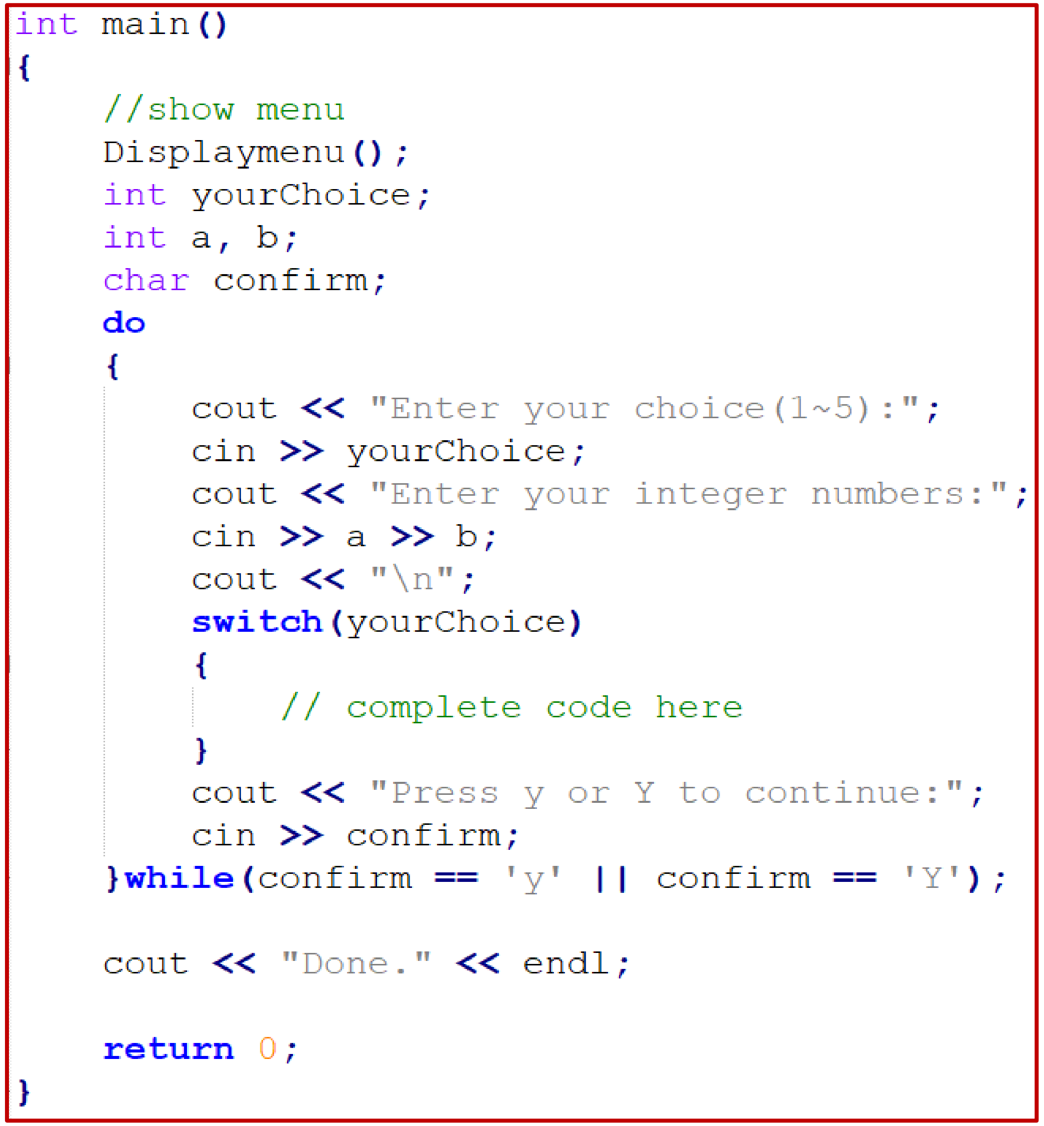
1. Write a program that will display the calculator menu. The program will prompt the user to choose the operation choice(from 1 to 5). Then it asks the user to input two integer values for the calculation. See the sample below.



```
====================================================
                   MENU
====================================================
      1.Add
      2.Subtract
      3.Multiply
      4.Divide
      5.Modulus
Enter your choice(1~5):1
Enter your integer numbers:4 -20

Result:-16
Press y or Y to continue:y
Enter your choice(1~5):3
Enter your integer numbers:3 7

Result:21
Press y or Y to continue:Y
Enter your choice(1~5):5
Enter your integer numbers:22 3

Result:1
Press y or Y to continue:n
Done.
```

The program also asks the user to decide whether he/she wants to continue the operation. If he/she inputs ‘y’(‘Y’), the program will prompt the user to choose the operation gain. Otherwise, the program will show “Done” and terminate.

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

void Displaymenu()
{
    // complete code here
}

int Add(int a, int b)
{
   // complete code here
}

int Substract(int a, int b)
{
   // complete code here
}

int Multiply(int a, int b)
{
    // complete code here
}

int Divide(int a, int b)
{
       //complete code here
}

int Modulus(int a, int b)
{
   // complete code here
}
```

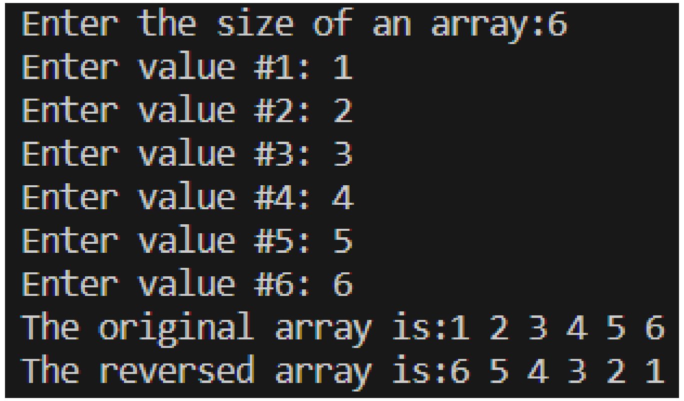


```
int main()
{
    //show menu
    Displaymenu();
    int yourChoice;
    int a, b;
    char confirm;
    do
    {
        cout << "Enter your choice(1~5):";
        cin >> yourChoice;
        cout << "Enter your integer numbers:";
        cin >> a >> b;
        cout << "\n";
        switch(yourChoice)
        {
            // complete code here
        }
        cout << "Press y or Y to continue:";
        cin >> confirm;
    }while(confirm == 'y' || confirm == 'Y');

    cout << "Done." << endl;

    return 0;
}
```

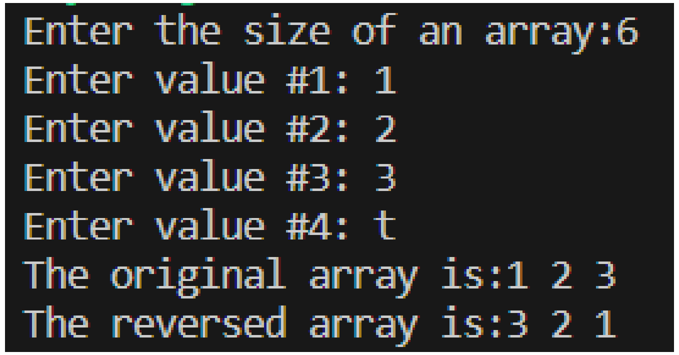
2. Write a program that uses the following functions:

- **int fill_array(double arr[], int size)** prompts the user to enter double values to the array. It ceases taking input when the array is full or when the user enters non-numeric input, and it returns the actual number of entries.
- **void show_array(double *arr, int size)** displays the contents of the array.
- **void reverse_array(double *arr, int size)** is a **recursive function**, it reverses the values stored in the array.

The program should use these functions to fill an array, show the array, reverse the array. Hint: use the dynamic array to store the data.



```
Enter the size of an array:6
Enter value #1: 1
Enter value #2: 2
Enter value #3: 3
Enter value #4: 4
Enter value #5: 5
Enter value #6: 6
The original array is:1 2 3 4 5 6
The reversed array is:6 5 4 3 2 1
```



```
Enter the size of an array:6
Enter value #1: 1
Enter value #2: 2
Enter value #3: 3
Enter value #4: t
The original array is:1 2 3
The reversed array is:3 2 1
```