南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

Lab 6, functions

廖琪梅，王大兴

Functions

- Function declaration, definition and call
- Function parameters and arguments
- Function return values
- Building shared libraries.

Functions

Functions implements the modularizing a program. It enables the software to reuse, avoiding code repetition and makes the program easier to test, debug and maintain.

Function definition:

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 the function name it is called.
- **parameters:** are variables to hold values of arguments passed while function is called. A function may or may not contain parameter list(**void**).

A **function prototype** is required unless the function is defined before it's used.The simplest way to get a prototype is to copy the **function header** and add a **semicolon**.

Function call:

function_name (arguments list);

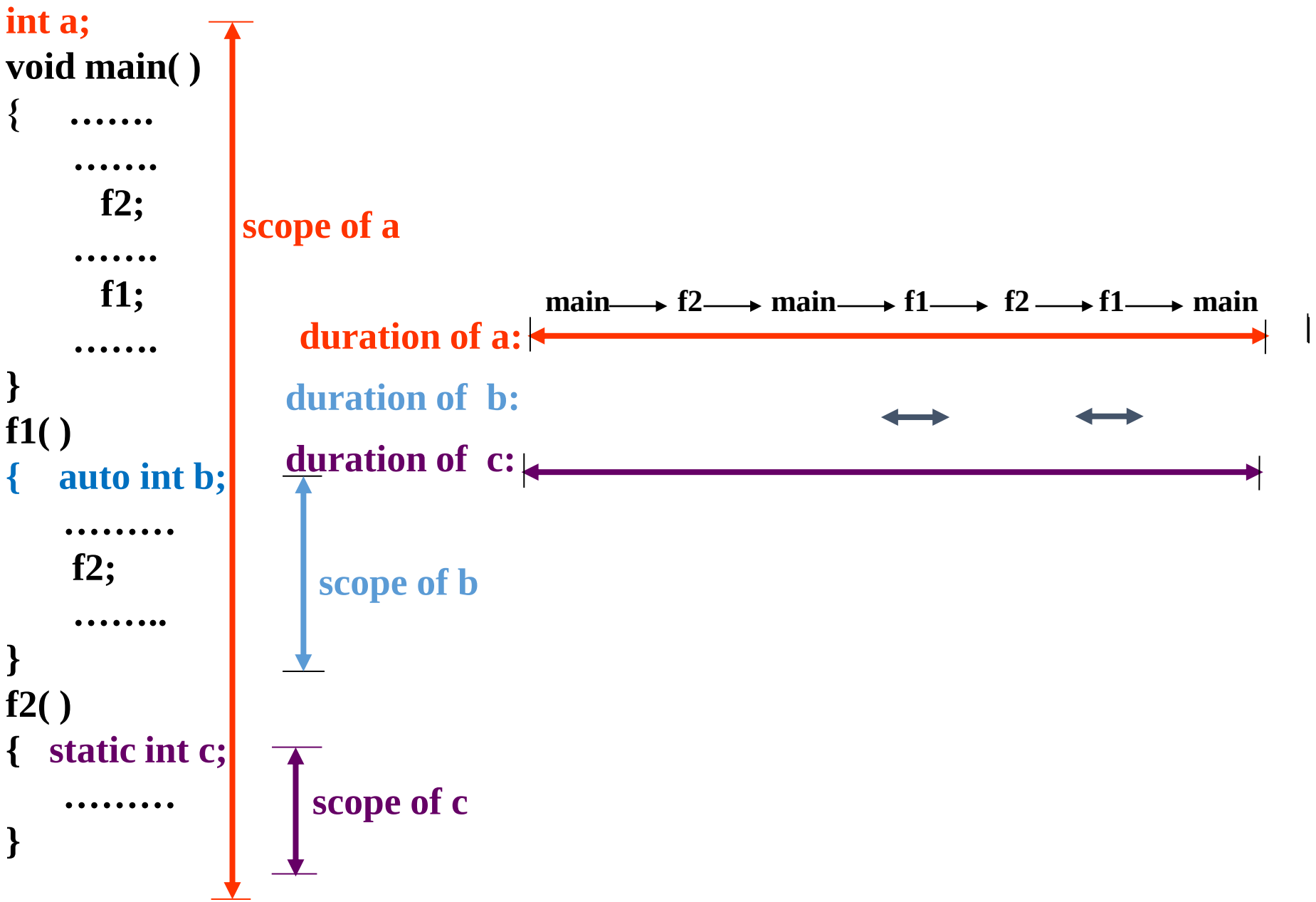
When a function is invoked, the values of arguments are passed to the parameters one by one according to the order in which the parameters are defined. The program flow proceeds to execute the function body until it encounters a return statement or closed brace at last.

Scope and duration of 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.

An variable's **storage duration** is the period during which that variable exits in memory.


int a;
void main( )
{ …….
…….
f2;
…….
f1;
…….
}
f1( )
{ auto int b;
……
f2;
……..
}
f2( )
{ static int c;
……
}
scope of a
scope of b
scope of c
main → f2 → main → f1 → f2 → f1 → main
duration of a:
duration of b:
duration of c:

Parameters and Arguments

A variable that's used to receive passed values is called a **formal parameter or formal argument**(or **parameter** for short). The value passed to the function is called the **actual parameter or actual argument**(or **argument** for short).

Actual parameter and Formal parameter

sumfunction.cpp > ...

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

//Declaring a function
int sum(int x, int y);
int main()
{
    int a = 10;
    int b = 20;
    int c;

    //Calling a function
    c = sum(a,b);

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

    return 0;
}

// Defining a function
int sum(int x, int y)
{
    return (x + y);
}
```

Actual parameters(arguments)

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

Formal parameters(parameters)

Passing Parameters

Parameters can be passed in two ways: *by value* or *by reference(or by address)*.

When you invoke a function with a parameter, the value of the argument is passed to the parameter. This is referred to as ***pass-by-value***, which means the **copy of argument** is assigned to the parameter. The value of argument is not affected, regardless of the changes made to the parameter inside the function.

For an argument of a pointer or a reference or an array type passing to the parameter is known as ***pass-by-reference(or by address)***, the reference or the address of argument is assigned to the parameter, which means both parameter and argument are pointed to the same memory, if you modify the content of parameter inside the function, you will see the same change of the argument outside the function.

Using **const pointer(or const reference)** in parameter can protect the value of the argument from modifying.

Return value

A function can return **a value**(primitive type, pointer type, reference type, structure type and so on) to the caller. Use **return statement** to get the return value in function definition.

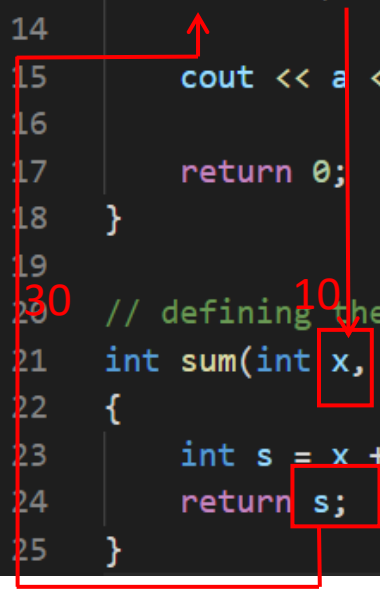
Note:

Do not return the address(or reference) of a local variable to the caller.

You can return dynamically allocated memory address or a static array or parameter pointer in the function definition.

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;
}
```



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.

Multiple files

student1.h > ...

just include once

```
1   #pragma once
2
3   struct student
4   {
5       int id;
6       char name[20];
7       float score;
8   };
9
10  void printstudent(student *record);
```

student_multifile.cpp > main()

```
1   #include <cstring>
2   #include "student1.h"
3
4   int main()
5   {
6       student record;
7
8       record.id = 1;
9       stpcpy(record.name, "Raju");
10      record.score = 86.5;
11
12      printstudent(&record);
13      return 0;
14  }
```

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.

```
1   #include <iostream>
2   #include "student1.h"
3
4   void printstudent(student *st)
5   {
6       std::cout << "Id is:" << st->id << std::endl;
7       std::cout << "Name is:" << st->name << std::endl;
8       std::cout << "Score is:" << st->score << std::endl;
9   }
```

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.

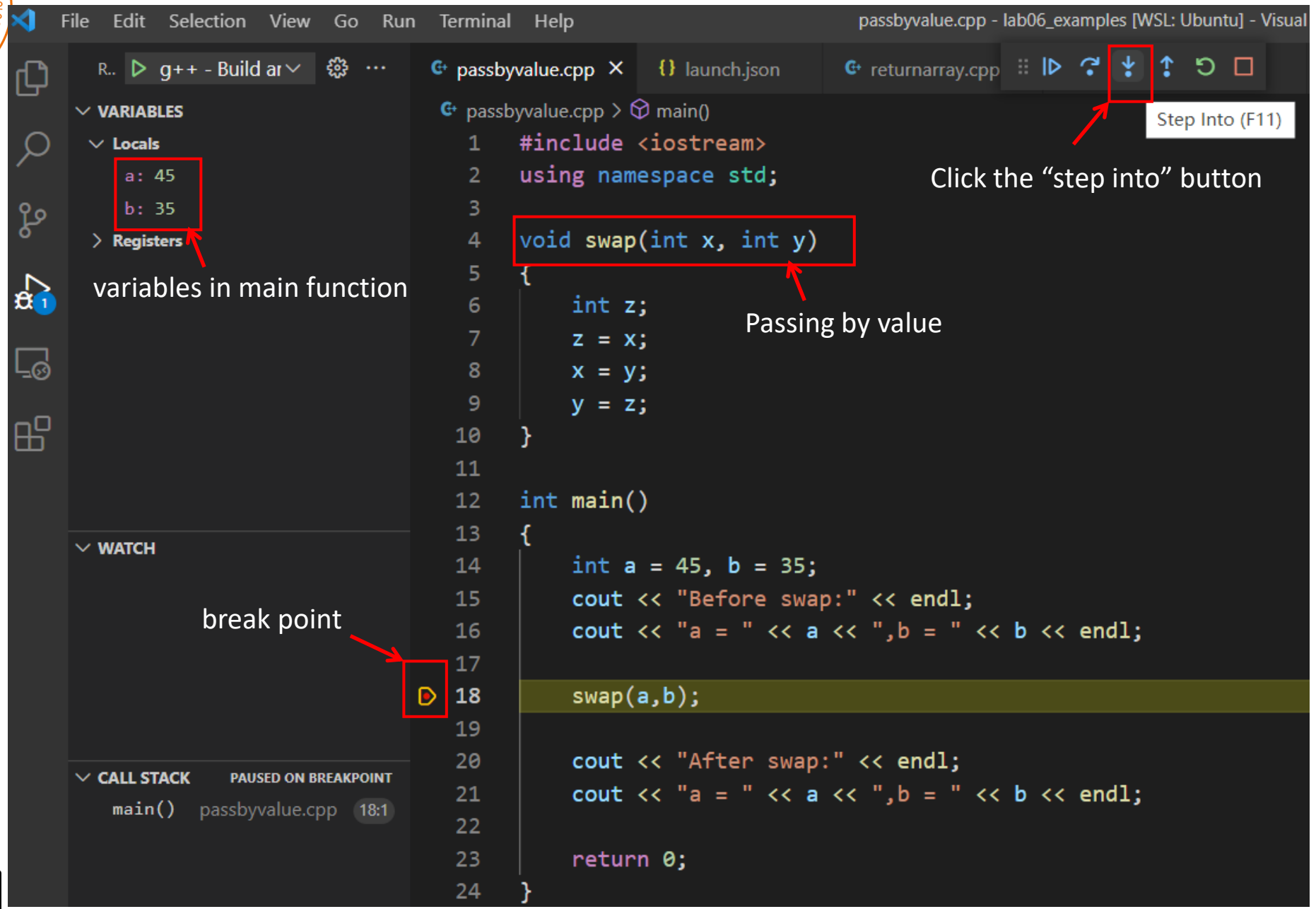
student.cpp
```
#include <iostream>
//#include "student1.h"
#include "student2.h"

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
```

Debugging program inside function by step into



R.. g++ - Build ar

VARIABLES

Locals

z: 32767

x: 21845

y: 1431655296

Registers

variables in swap function

WATCH

CALL STACK PAUSED ON STEP

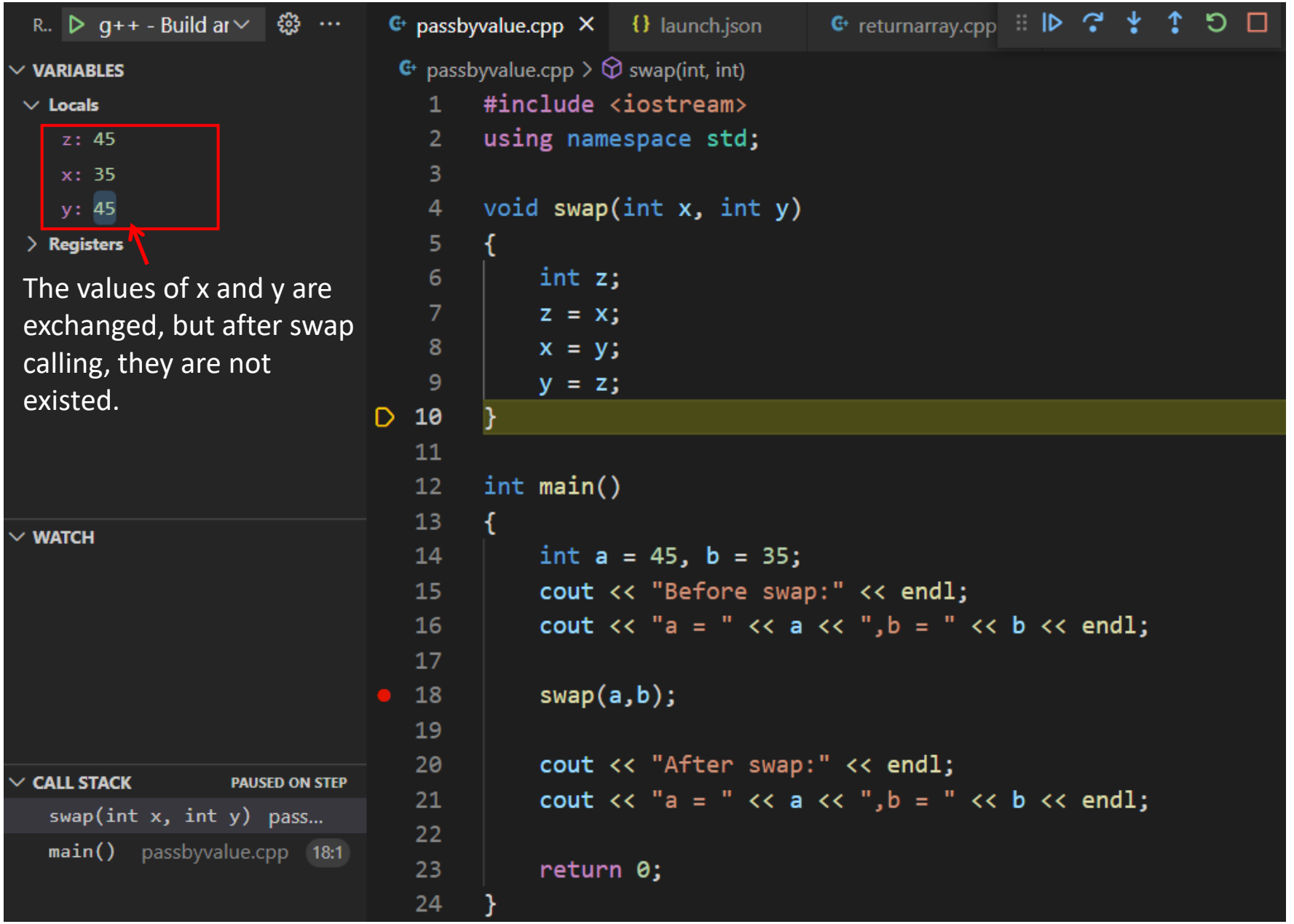
swap(int x, int y) pass...

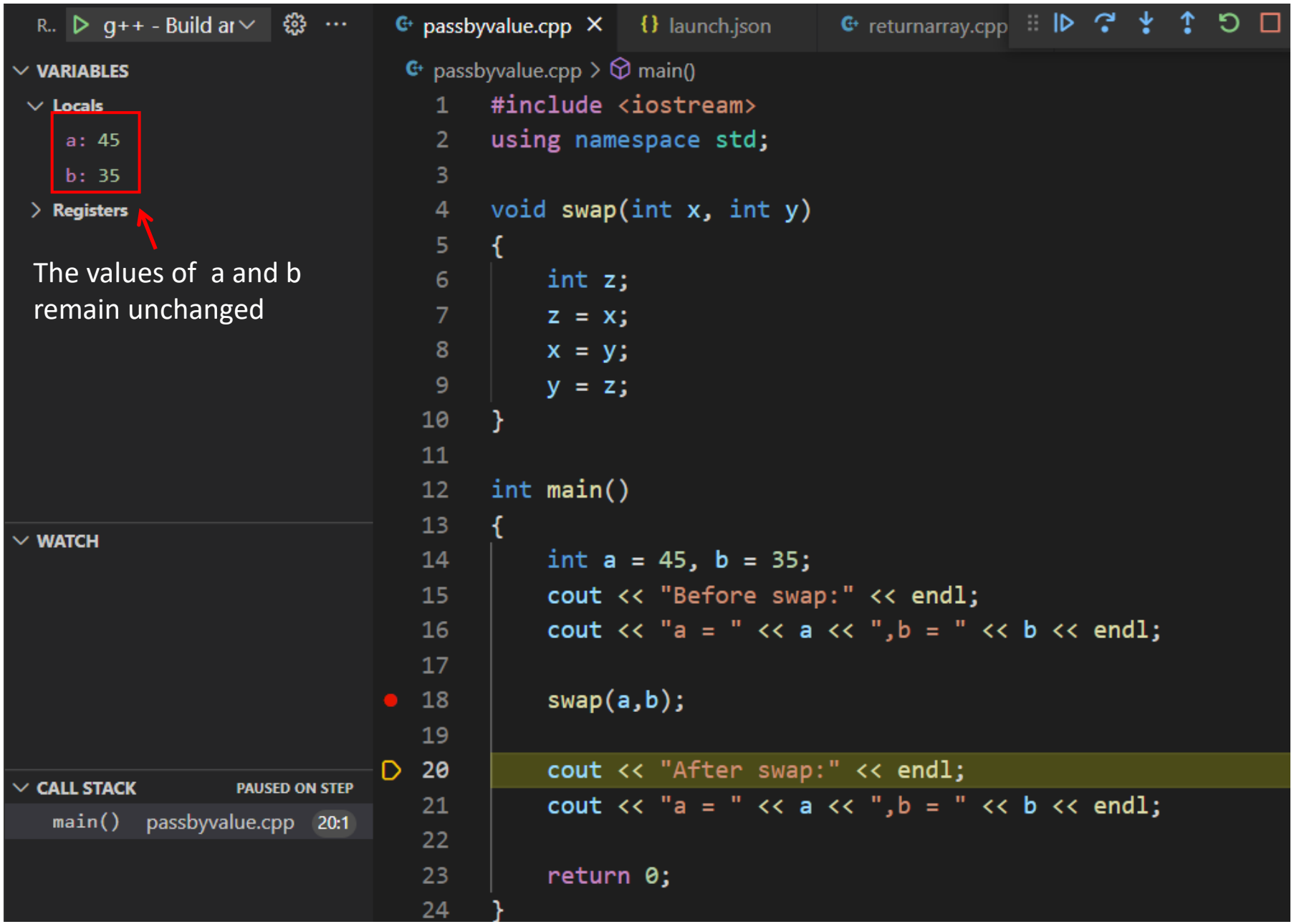
main() passbyvalue.cpp 18:1

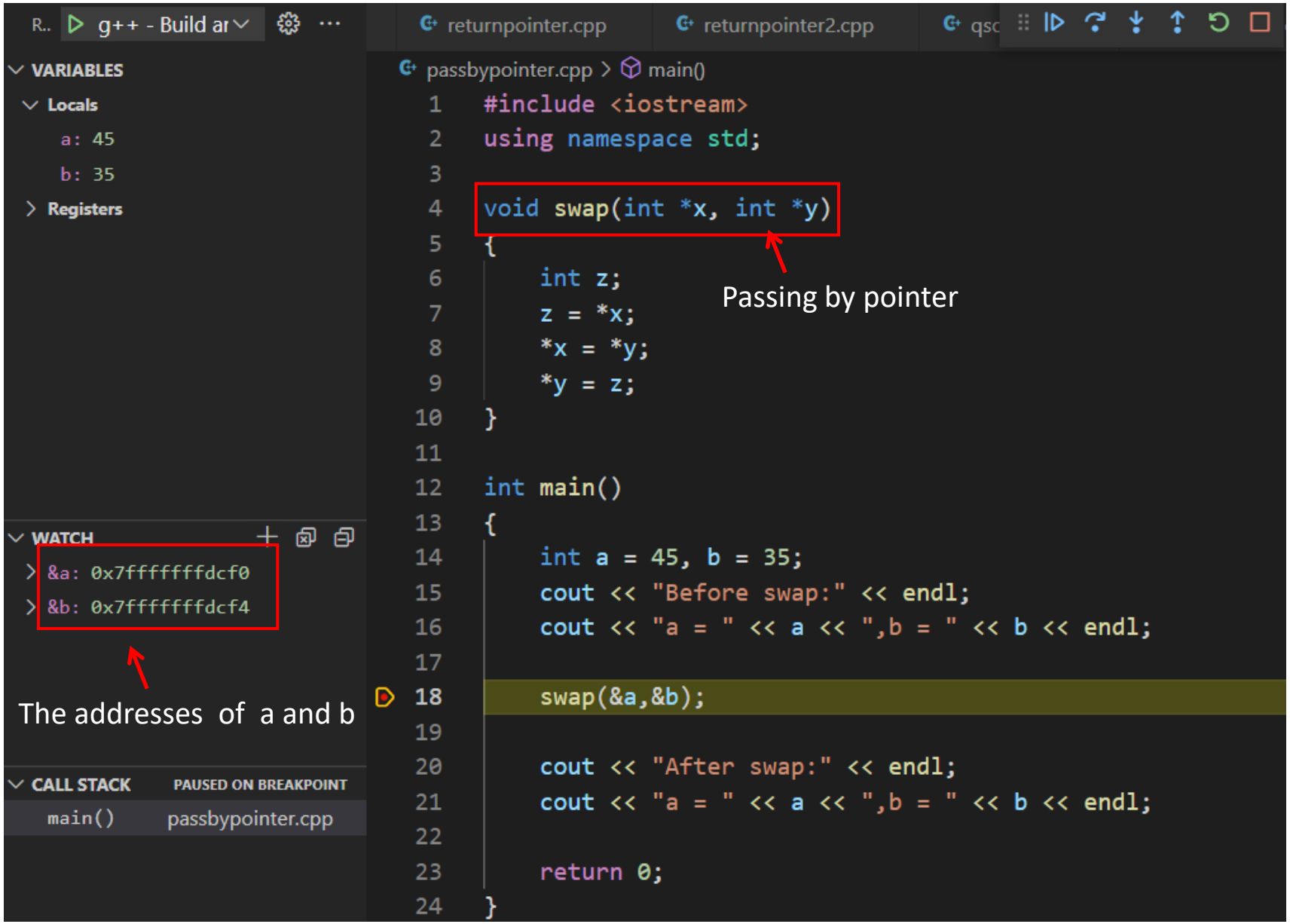
passbyvalue.cpp launch.json returnarray.cpp

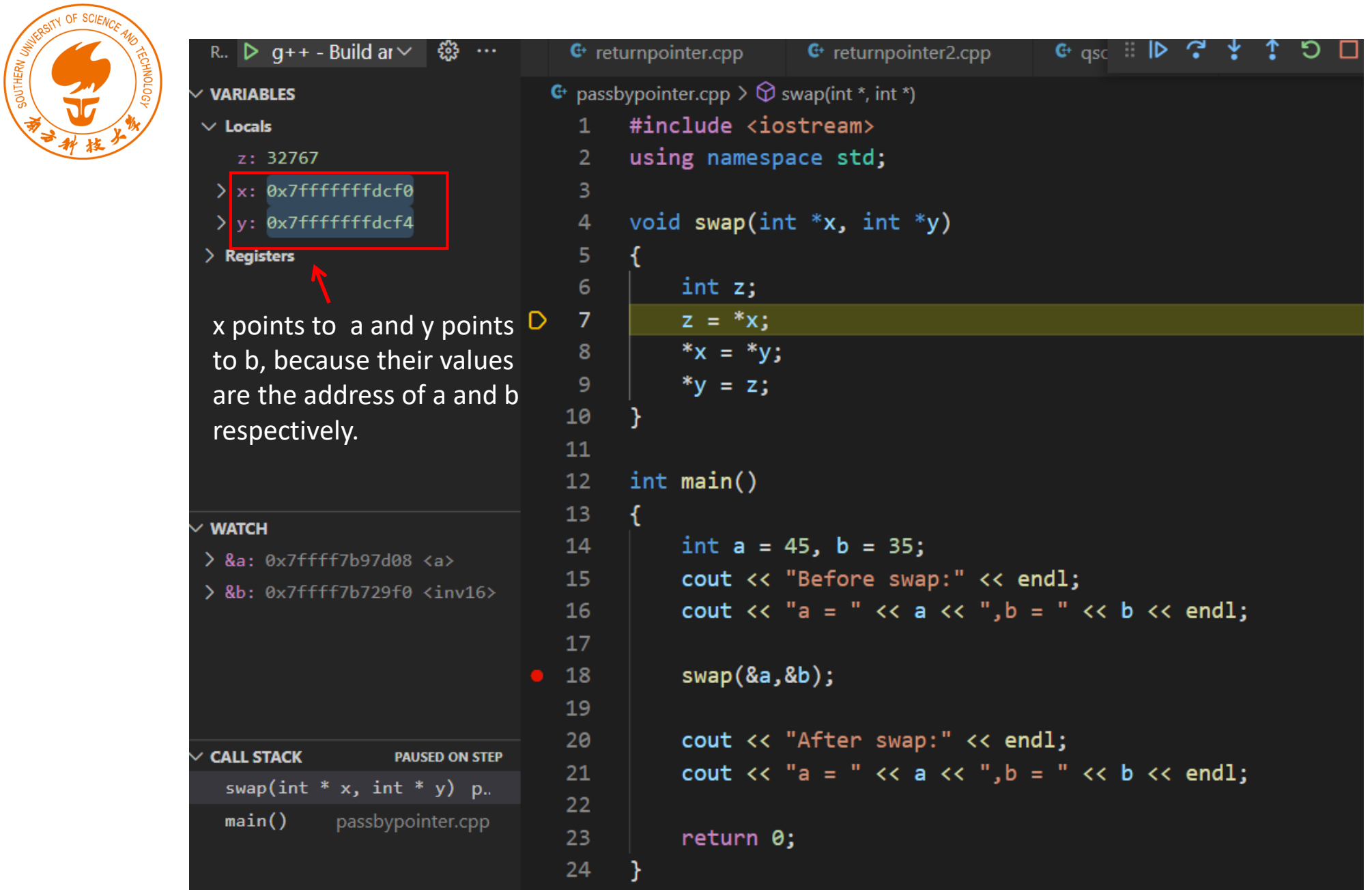
passbyvalue.cpp > swap(int, int)

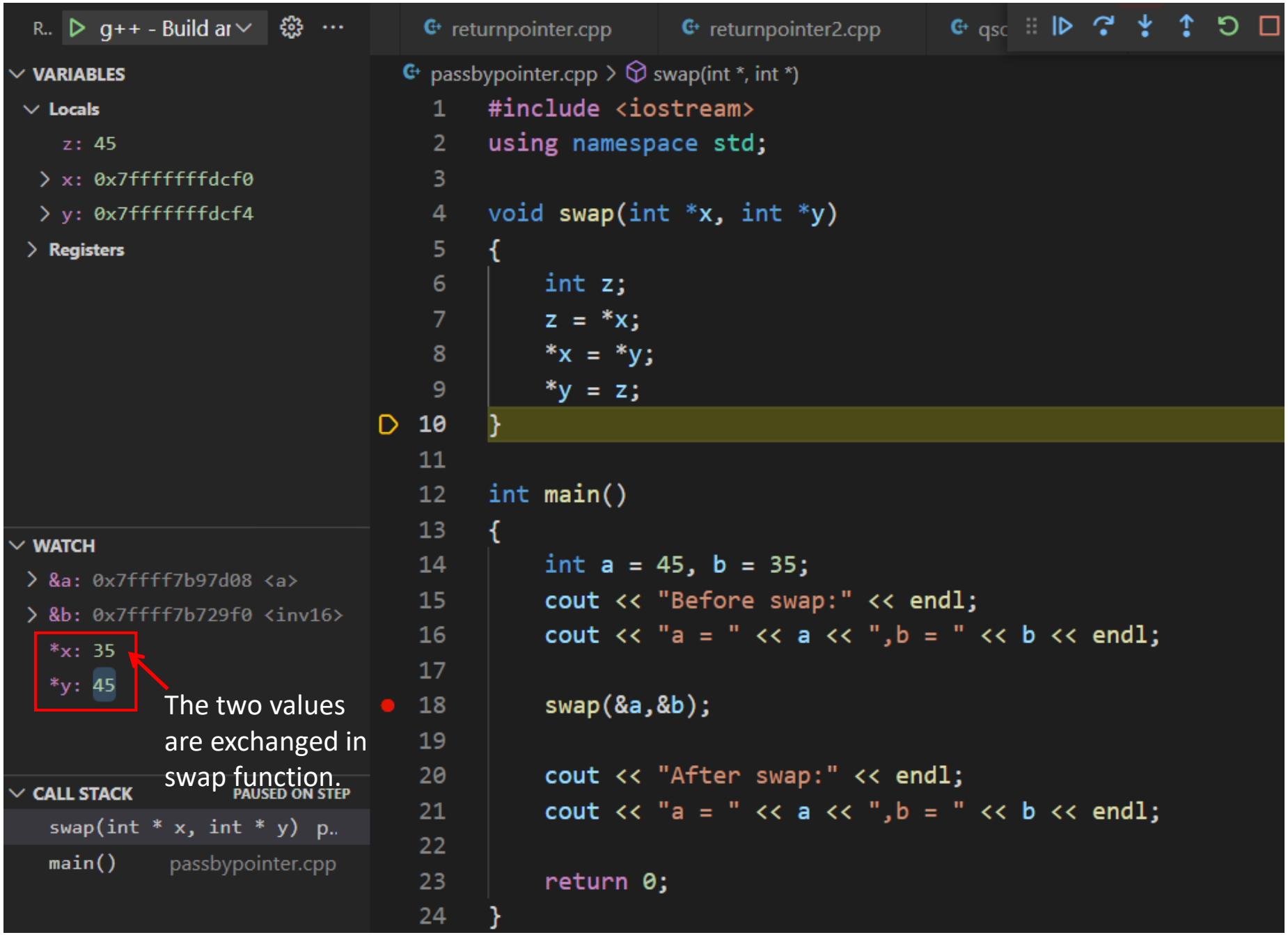
```
1   #include <iostream>
2   using namespace std;
3
4   void swap(int x, int y)
5   {
6       int z;
7       z = x;
8       x = y;
9       y = z;
10  }
11
12  int main()
13  {
14      int a = 45, b = 35;
15      cout << "Before swap:" << endl;
16      cout << "a = " << a << ",b = " << b << endl;
17
18      swap(a,b);
19
20      cout << "After swap:" << endl;
21      cout << "a = " << a << ",b = " << b << endl;
22
23      return 0;
24  }
```

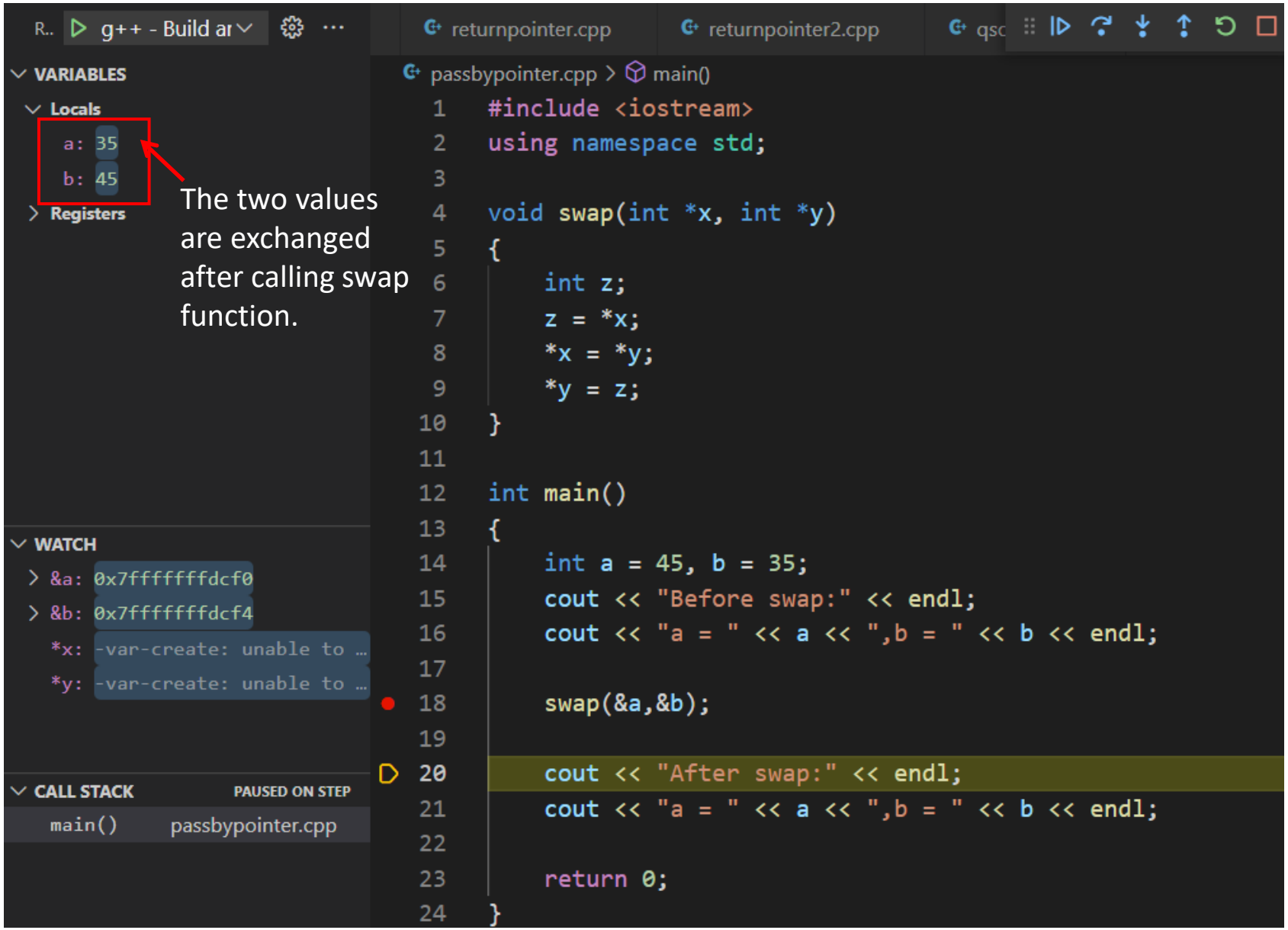

R.. g++ - Build ar
passbyvalue.cpp
launch.json
returnarray.cpp
passbyvalue.cpp > swap(int, int)
VARIABLES
Locals
z: 45
x: 35
y: 45
Registers
The values of x and y are exchanged, but after swap calling, they are not existed.
WATCH
CALL STACK
PAUSED ON STEP
swap(int x, int y) pass...
main() passbyvalue.cpp 18:1
1 #include <iostream>
2 using namespace std;
3
4 void swap(int x, int y)
5 {
6 int z;
7 z = x;
8 x = y;
9 y = z;
10 }
11
12 int main()
13 {
14 int a = 45, b = 35;
15 cout << "Before swap:" << endl;
16 cout << "a = " << a << ",b = " << b << endl;
17
18 swap(a,b);
19
20 cout << "After swap:" << endl;
21 cout << "a = " << a << ",b = " << b << endl;
22
23 return 0;
24 }


R.. g++ - Build ar
passbyvalue.cpp
launch.json
returnarray.cpp
passbyvalue.cpp > main()
VARIABLES
Locals
a: 45
b: 35
Registers
The values of a and b remain unchanged
WATCH
CALL STACK
PAUSED ON STEP
main() passbyvalue.cpp 20:1
1 #include <iostream>
2 using namespace std;
3
4 void swap(int x, int y)
5 {
6 int z;
7 z = x;
8 x = y;
9 y = z;
10 }
11
12 int main()
13 {
14 int a = 45, b = 35;
15 cout << "Before swap:" << endl;
16 cout << "a = " << a << ",b = " << b << endl;
17
18 swap(a,b);
19
20 cout << "After swap:" << endl;
21 cout << "a = " << a << ",b = " << b << endl;
22
23 return 0;
24 }


R.. g++ - Build ar
returnpointer.cpp
returnpointer2.cpp
qsc
VARIABLES
Locals
a: 45
b: 35
Registers
WATCH
&a: 0x7fffffffdcf0
&b: 0x7fffffffdcf4
The addresses of a and b
CALL STACK
PAUSED ON BREAKPOINT
main() passbypointer.cpp
passbypointer.cpp > main()
1 #include <iostream>
2 using namespace std;
3
4 void swap(int *x, int *y)
5 {
6 int z;
7 z = *x;
8 *x = *y;
9 *y = z;
10 }
11
12 int main()
13 {
14 int a = 45, b = 35;
15 cout << "Before swap:" << endl;
16 cout << "a = " << a << ",b = " << b << endl;
17
18 swap(&a,&b);
19
20 cout << "After swap:" << endl;
21 cout << "a = " << a << ",b = " << b << endl;
22
23 return 0;
24 }
Passing by pointer


R.. g++ - Build ar
returnpointer.cpp
returnpointer2.cpp
qsc
VARIABLES
Locals
z: 32767
x: 0x7fffffffdcf0
y: 0x7fffffffdcf4
Registers
x points to a and y points to b, because their values are the address of a and b respectively.
WATCH
&a: 0x7ffff7b97d08 <a>
&b: 0x7ffff7b729f0 <inv16>
CALL STACK
PAUSED ON STEP
swap(int * x, int * y) p..
main() passbypointer.cpp
passbypointer.cpp > swap(int *, int *)
1 #include <iostream>
2 using namespace std;
3
4 void swap(int *x, int *y)
5 {
6 int z;
7 z = *x;
8 *x = *y;
9 *y = z;
10 }
11
12 int main()
13 {
14 int a = 45, b = 35;
15 cout << "Before swap:" << endl;
16 cout << "a = " << a << ",b = " << b << endl;
17
18 swap(&a,&b);
19
20 cout << "After swap:" << endl;
21 cout << "a = " << a << ",b = " << b << endl;
22
23 return 0;
24 }


VARIABLES
Locals
z: 45
x: 0x7fffffffdcf0
y: 0x7fffffffdcf4
Registers
WATCH
&a: 0x7ffff7b97d08 <a>
&b: 0x7ffff7b729f0 <inv16>
*x: 35
*y: 45
The two values are exchanged in swap function.
CALL STACK
PAUSED ON STEP
swap(int * x, int * y) p..
main() passbypointer.cpp
returnpointer.cpp
returnpointer2.cpp
passbypointer.cpp > swap(int *, int *)
#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;
}


R..
g++ - Build ar
returnpointer.cpp
returnpointer2.cpp
qsc
VARIABLES
Locals
a: 35
b: 45
Registers
The two values are exchanged after calling swap function.
WATCH
&a: 0x7fffffffdcf0
&b: 0x7fffffffdcf4
*x: -var-create: unable to ...
*y: -var-create: unable to ...
CALL STACK
PAUSED ON STEP
main()
passbypointer.cpp
passbypointer.cpp > main()
#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;
}

Reference in functions

A **reference** defines an alternative name (or **alias**) for an object. A reference type "refer to" another variable. Using "**&**" to declare a reference.

```
int ival = 1024;
int &refVal = ival;  // reVal refers to (is another name for) ival
int &refVal2;        // error:a reference must be initialized
```

Once initialized, a reference remains bound to its initial object. There is no way to rebind a reference to refer to a different object.

After a reference has been defined, all operations on that reference are actually operations on the object to which the reference is bound.

```
refVal = 2;   // assign 2 to the object to which refVal refers, i.e., to ival
int ii = refVal;  // same as ii = ival

int &refVal2 = 10;   // error:initializer must be an object
double dval = 3.14;
int &refVal3 = dval; // error:initializer must be an int object
```

Reference as function parameters –passing by reference

passreference.cpp > ...

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

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

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;
}
```

Only by checking the function prototype or function definition can you tell whether the function passing by value or by reference.
In the called function's body, the reference parameter actually refers to ***the original variable*** in the calling function, and the original variable can be ***modified*** directly by the called function.

The style of the arguments are like common variables

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

const reference: reference that refers to a const type. A reference to const cannot be used to change the object to which the reference is bound.

```
const int ci = 100;
const int &r1 = ci;  // OK: both reference and underlying object are const
r1 = 42;             // error: r1 is a const reference
int &r2 = ci;       // error: non const reference to a const object
```

The type of a reference must match the type of the object to which it refers. But there are two exceptions to the rule. **The one** is that a const reference can refer to an non const object. **The other** is that we can initialize a const reference from any expression that can be converted to the type of the reference.

```
int i = 42;
int &r1 = i;        // OK: r1 bound to i
const int &r2 = i; // OK: r2 is const reference, bound to non const object
const int &r3 = 99; //OK: r3 is const reference
const int &r4 = r1 * 2;  //OK: r4 is const reference
int &r5 = r1 * 3;  // error: r5 is non const reference
r1 = 0;   //OK: r1 is non const reference, it might be changed
r2 = 5;   // error: r2 is const reference, it can not be used to change i
```

passparameter.cpp > ...

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

void passbyval(int n)
{
    cout << "Pass by value---the operation address of the function is:" << &n << endl;
    n++;
}

void passbypoi(int *n)
{
    cout << "Pass by pointer---the operation address of the function is:" << n << endl;
    ++(*n);
}

void passbyref(int &n)
{
    cout << "Pass by reference---the operation address of the function is:" << &n << endl;
    ++n;
}

int main()
{
    int n = 10;

    cout << "The address of the argument is:" << &n << endl << endl;

    passbyval(n);
    cout << "After calling passbyval(), n = " << n << endl << endl;

    passbypoi(&n);
    cout << "After calling passbypoi(), n = " << n << endl << endl;

    passbyref(n);
    cout << "After calling passbyref(), n = "<< n << endl << endl;

    return 0;

}
```

Pass by value

Pass by pointer

Pass by reference

```
The address of the argument is:0x7ffd980a7e24

Pass by value---the operation address of the function is:0x7ffd980a7e0c
After calling passbyval(), n = 10

Pass by pointer---the operation address of the function is:0x7ffd980a7e24
After calling passbypoi(), n = 11

Pass by reference---the operation address of the function is:0x7ffd980a7e24
After calling passbyref(), n = 12
```

different

the same

the same

Passing by value, the address that the function operates is not that of the argument; but **passing by reference(or pointer)**, the function operates the address of argument.

Return a Reference

pointstructure.cpp > main()

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

struct point
{
    double x;
    double y;
};
point mid1(const point &, const point &);
point* mid2(const point &, const point &);
void mid3(const point &, const point &, point &);
point& mid4(const point &, const point &);

```

```cpp
point& mid4(const point &p1, const point &p2)
{
    point p;
    p.x = (p1.x + p2.x)/2;
    p.y = (p1.y + p2.y)/2;

    return p;
}
```

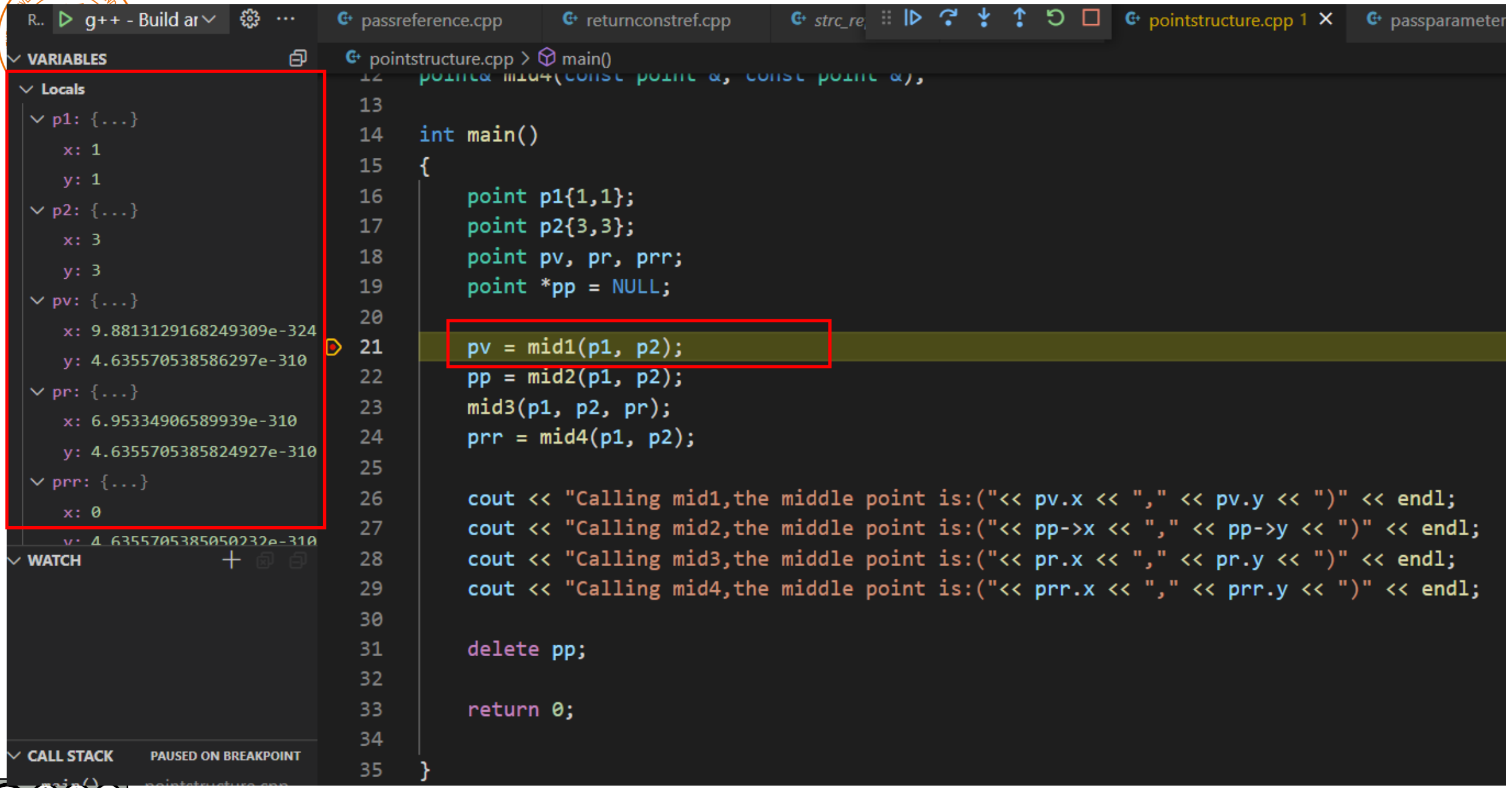
```cpp
int main()
{
    point p1{1,1};
    point p2{3,3};
    point pv, pr, prr;
    point *pp = NULL;

    pv = mid1(p1, p2);
    pp = mid2(p1, p2);
    mid3(p1, p2, pr);
    prr = mid4(p1, p2);

    cout << "Calling mid1,the middle point is:(" << pv.x << "," << pv.y << ")" << endl;
    cout << "Calling mid2,the middle point is:(" << pp->x << "," << pp->y << ")" << endl;
    cout << "Calling mid3,the middle point is:(" << pr.x << "," << pr.y << ")" << endl;
    cout << "Calling mid4,the middle point is:(" << prr.x << "," << prr.y << ")" << endl;

    delete pp;

    return 0;

}
```

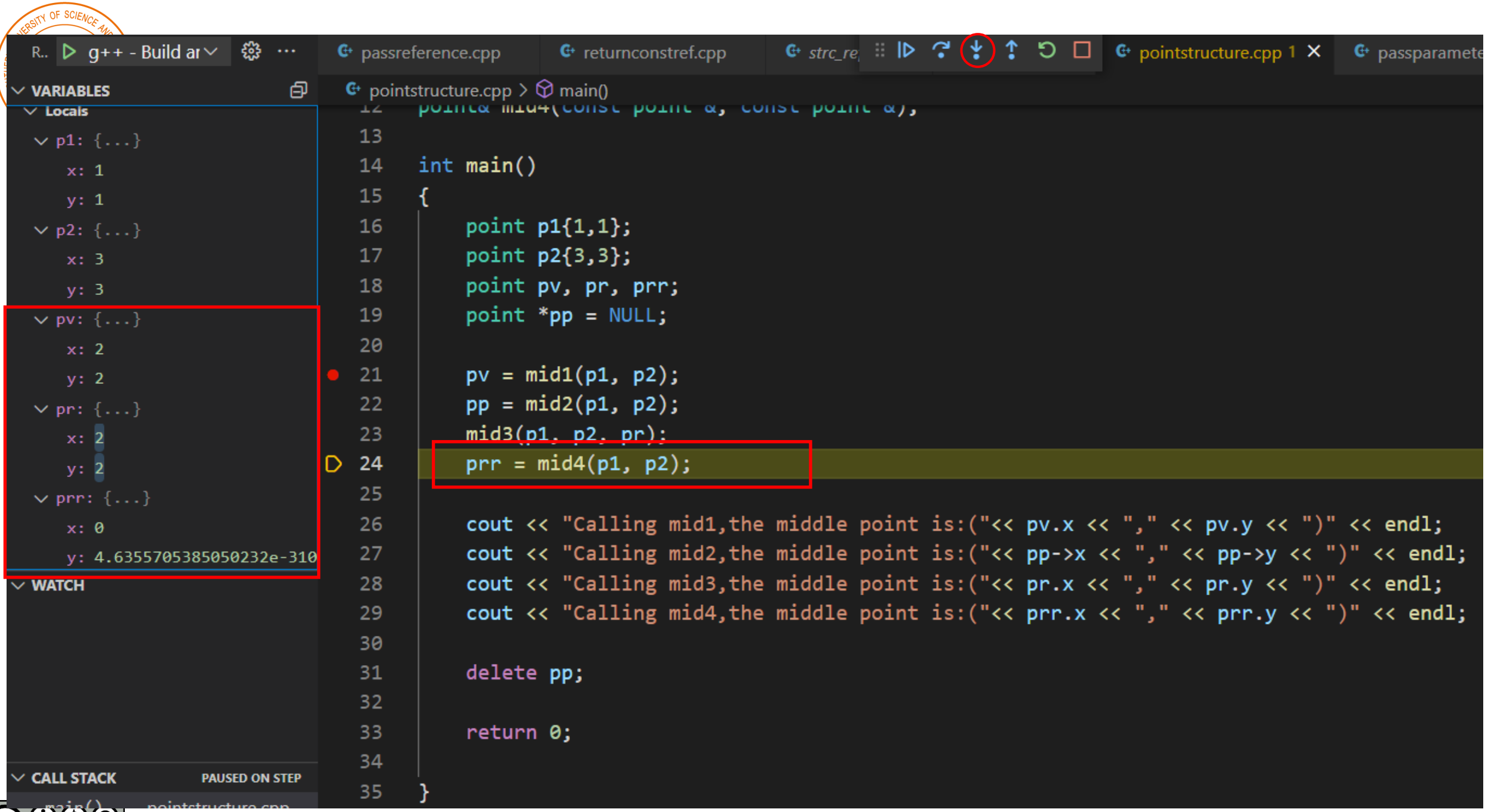
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab07_examples$ g++ pointstructure.cpp
pointstructure.cpp: In function 'point& mid4(const point&, const point&)':
pointstructure.cpp:67:12: warning: reference to local variable 'p' returned [-Wreturn-local-addr]
   67 |     return p;
      |            ^
pointstructure.cpp:63:11: note: declared here
   63 |     point p;
      |           ^
```

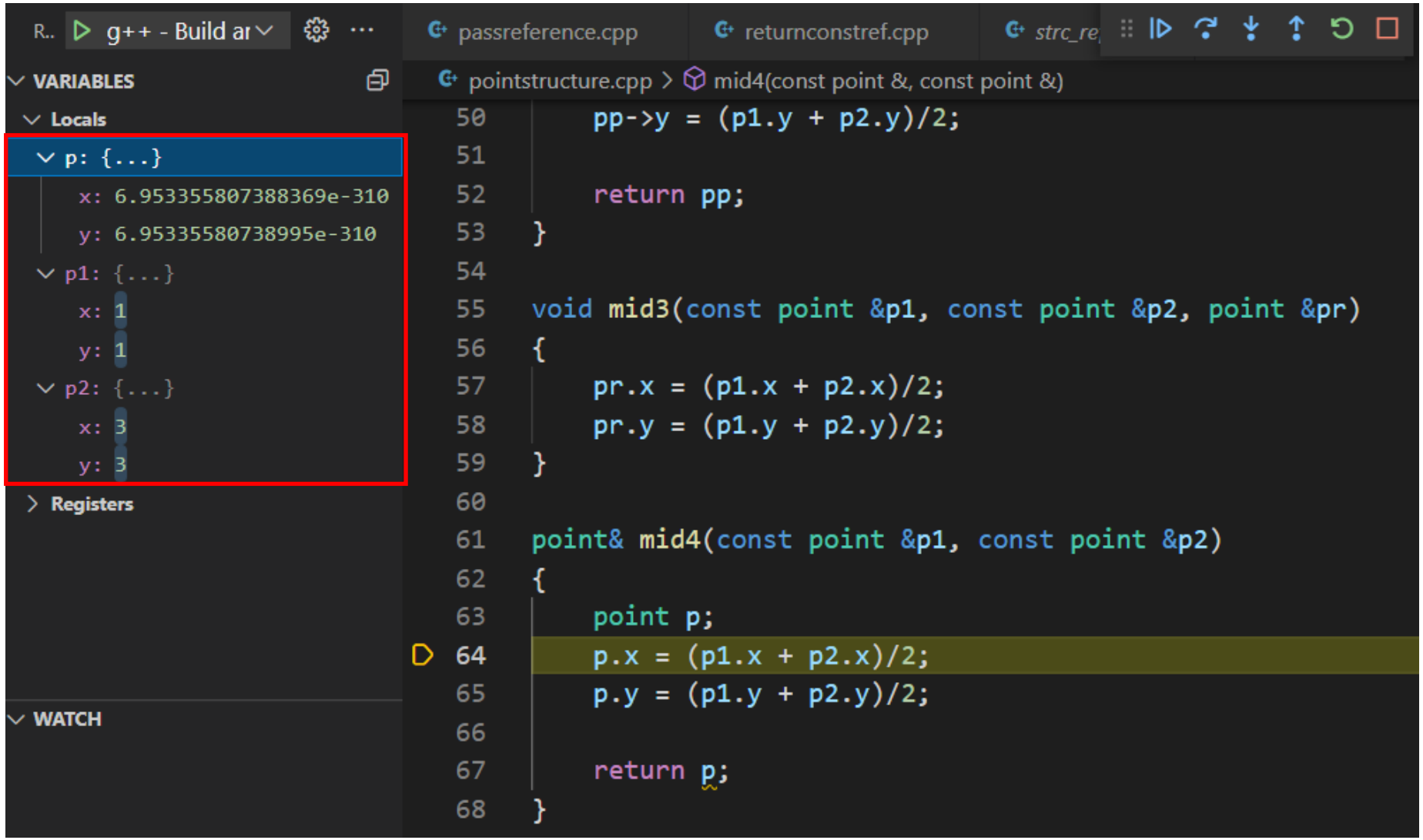
```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab07_examples$ ./a.out
Segmentation fault
```

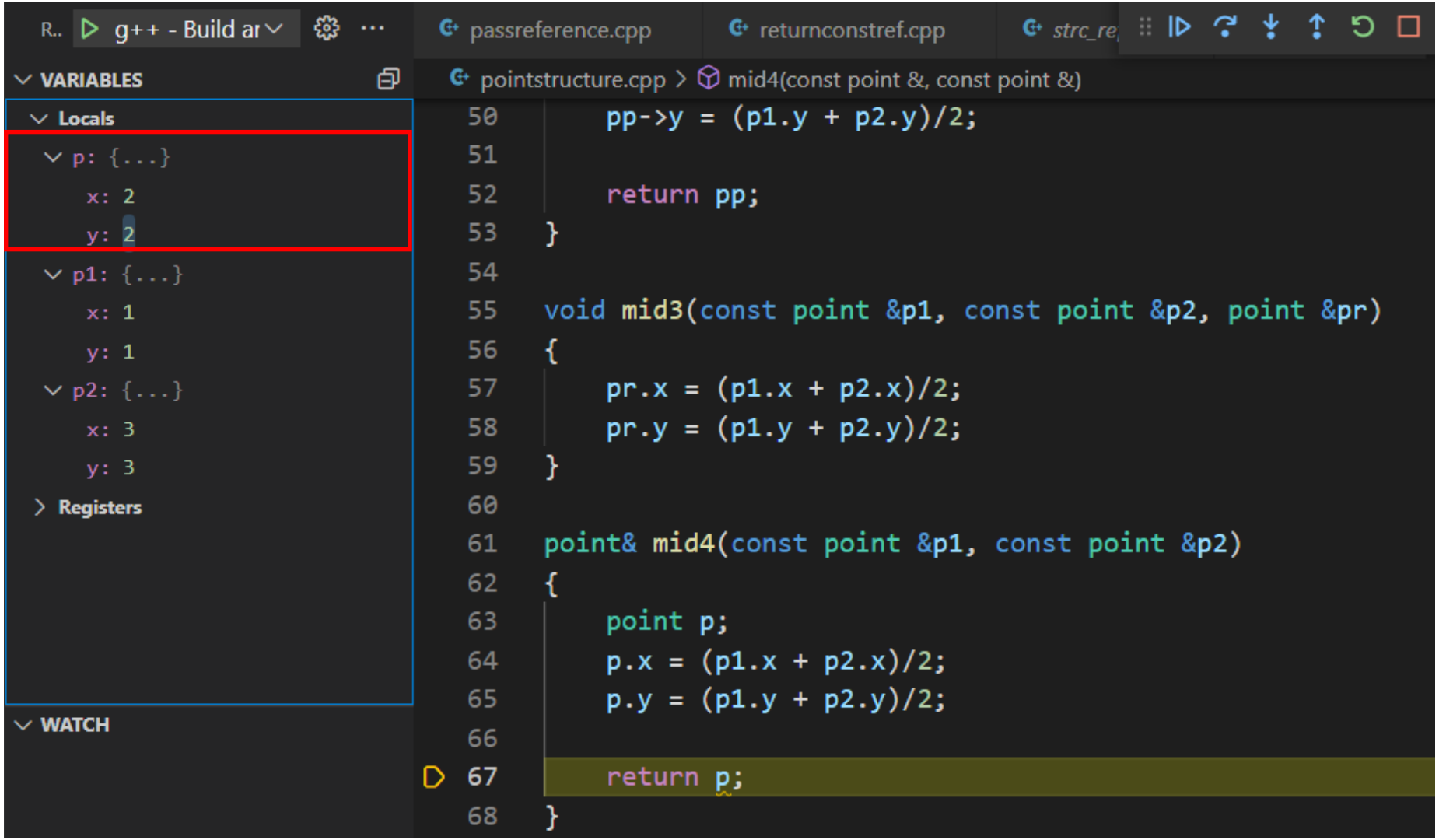
The program can not be executed.

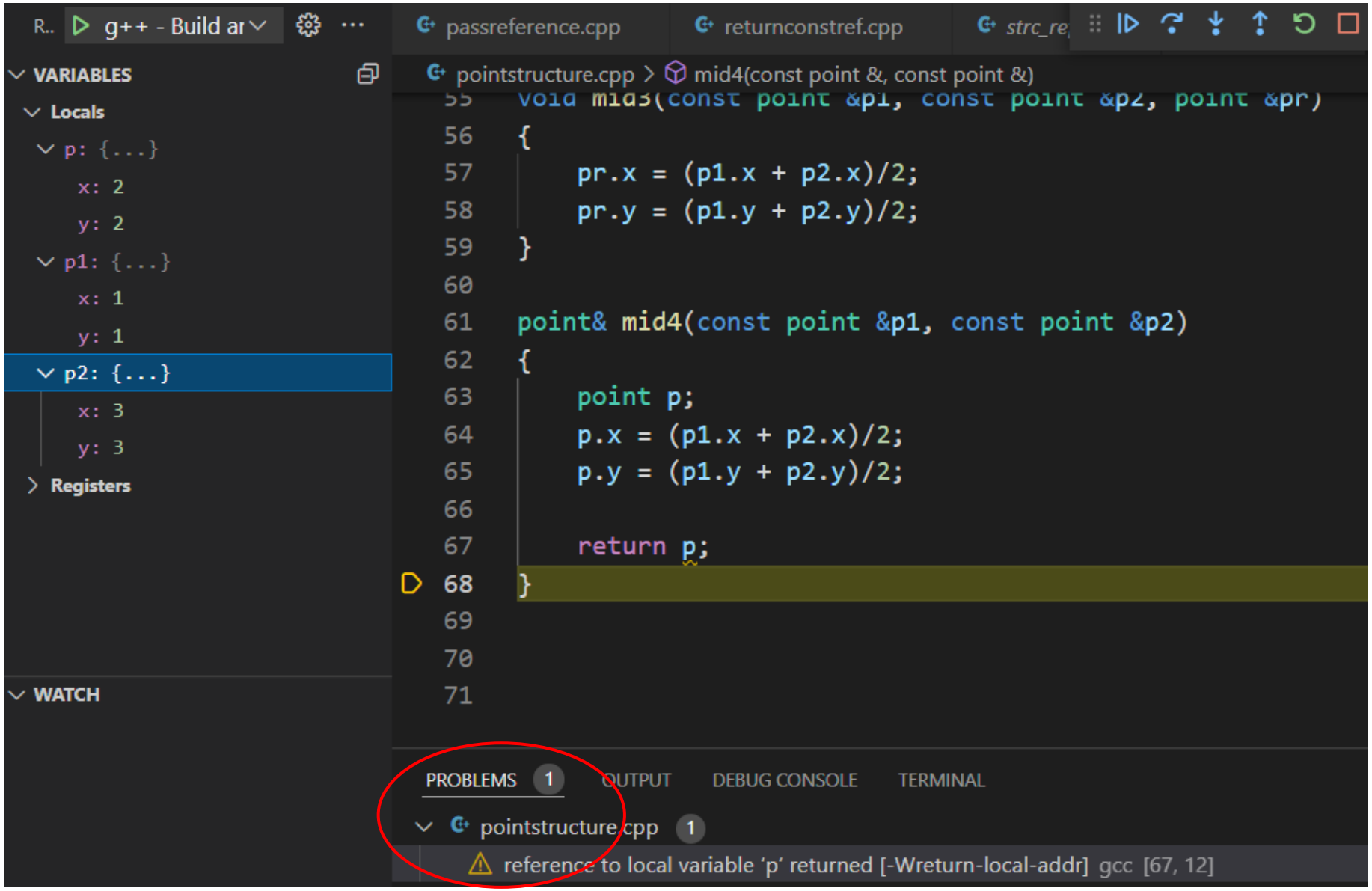
Do not return a reference of a local variable.
You can return a reference parameter.

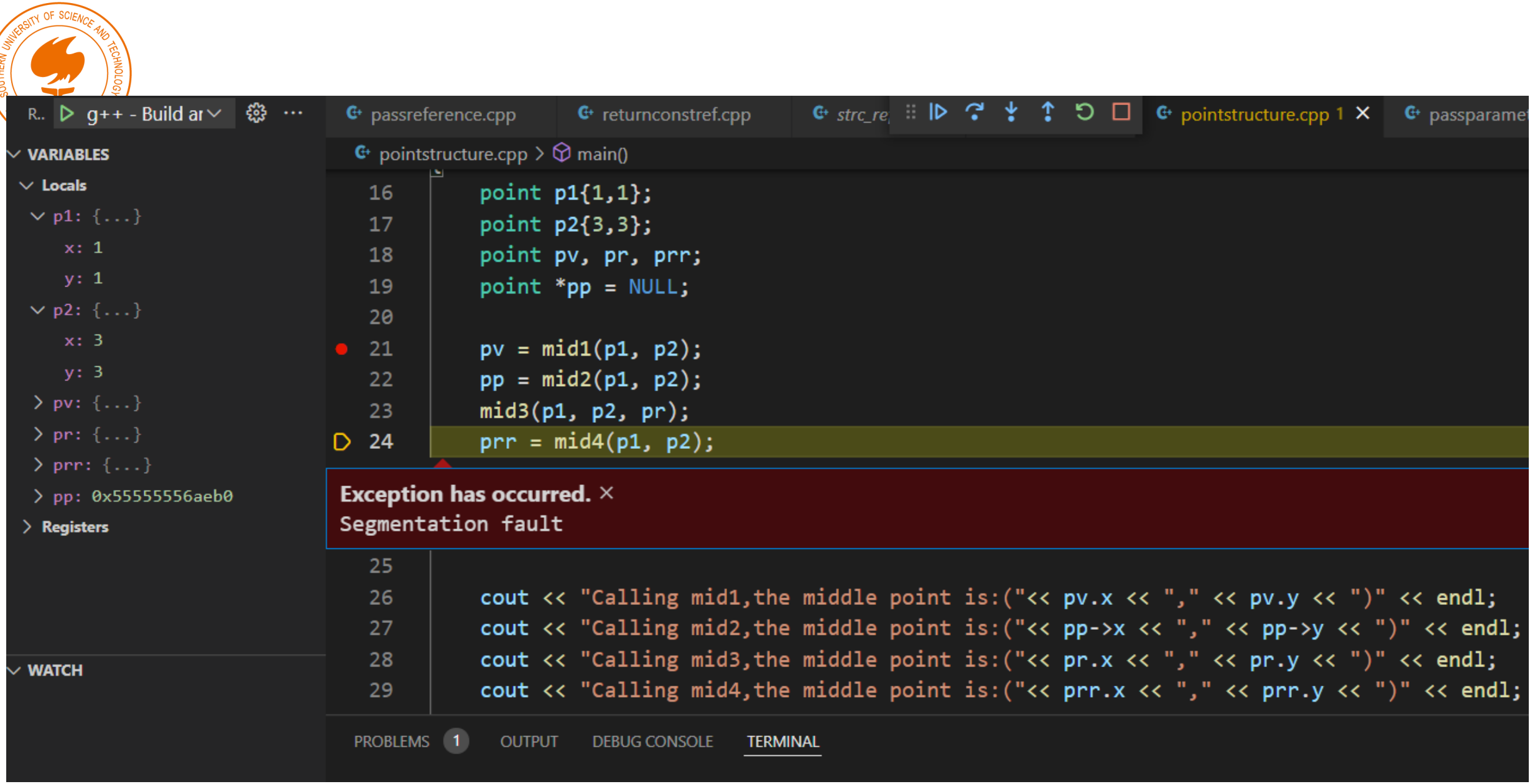

R.. g++ - Build ar
passreference.cpp
returnconstref.cpp
strc_re
pointstructure.cpp 1
passparameter
VARIABLES
Locals
p1: {...}
x: 1
y: 1
p2: {...}
x: 3
y: 3
pv: {...}
x: 9.8813129168249309e-324
y: 4.635570538586297e-310
pr: {...}
x: 6.95334906589939e-310
y: 4.6355705385824927e-310
prr: {...}
x: 0
WATCH
CALL STACK
PAUSED ON BREAKPOINT
pointstructure.cpp > main()
int main()
{
point p1{1,1};
point p2{3,3};
point pv, pr, prr;
point *pp = NULL;
pv = mid1(p1, p2);
pp = mid2(p1, p2);
mid3(p1, p2, pr);
prr = mid4(p1, p2);
cout << "Calling mid1,the middle point is:("<< pv.x << "," << pv.y << ")" << endl;
cout << "Calling mid2,the middle point is:("<< pp->x << "," << pp->y << ")" << endl;
cout << "Calling mid3,the middle point is:("<< pr.x << "," << pr.y << ")" << endl;
cout << "Calling mid4,the middle point is:("<< prr.x << "," << prr.y << ")" << endl;
delete pp;
return 0;
}


g++ - Build ar
passreference.cpp
returnconstref.cpp
pointstructure.cpp 1
passparamete
VARIABLES
Locals
p1: {...}
x: 1
y: 1
p2: {...}
x: 3
y: 3
pv: {...}
x: 2
y: 2
pr: {...}
x: 2
y: 2
prr: {...}
x: 0
y: 4.6355705385050232e-310
WATCH
CALL STACK
PAUSED ON STEP
pointstructure.cpp > main()
13
14 int main()
15 {
16 point p1{1,1};
17 point p2{3,3};
18 point pv, pr, prr;
19 point *pp = NULL;
20
21 pv = mid1(p1, p2);
22 pp = mid2(p1, p2);
23 mid3(p1, p2, pr);
24 prr = mid4(p1, p2);
25
26 cout << "Calling mid1,the middle point is:(" << pv.x << "," << pv.y << ")" << endl;
27 cout << "Calling mid2,the middle point is:(" << pp->x << "," << pp->y << ")" << endl;
28 cout << "Calling mid3,the middle point is:(" << pr.x << "," << pr.y << ")" << endl;
29 cout << "Calling mid4,the middle point is:(" << prr.x << "," << prr.y << ")" << endl;
30
31 delete pp;
32
33 return 0;
34
35 }


R..
g++ - Build ar
passreference.cpp
returnconstref.cpp
strc_re
VARIABLES
Locals
p: {...}
x: 6.953355807388369e-310
y: 6.95335580738995e-310
p1: {...}
x: 1
y: 1
p2: {...}
x: 3
y: 3
Registers
WATCH
pointstructure.cpp > mid4(const point &, const point &)
50 pp->y = (p1.y + p2.y)/2;
51
52 return pp;
53 }
54
55 void mid3(const point &p1, const point &p2, point &pr)
56 {
57 pr.x = (p1.x + p2.x)/2;
58 pr.y = (p1.y + p2.y)/2;
59 }
60
61 point& mid4(const point &p1, const point &p2)
62 {
63 point p;
64 p.x = (p1.x + p2.x)/2;
65 p.y = (p1.y + p2.y)/2;
66
67 return p;
68 }


R..
g++ - Build ar
passreference.cpp
returnconstref.cpp
strc_re
VARIABLES
Locals
p: {...}
x: 2
y: 2
p1: {...}
x: 1
y: 1
p2: {...}
x: 3
y: 3
Registers
WATCH
pointstructure.cpp > mid4(const point &, const point &)
50 pp->y = (p1.y + p2.y)/2;
51
52 return pp;
53 }
54
55 void mid3(const point &p1, const point &p2, point &pr)
56 {
57 pr.x = (p1.x + p2.x)/2;
58 pr.y = (p1.y + p2.y)/2;
59 }
60
61 point& mid4(const point &p1, const point &p2)
62 {
63 point p;
64 p.x = (p1.x + p2.x)/2;
65 p.y = (p1.y + p2.y)/2;
66
67 return p;
68 }


R.. g++ - Build ar
passreference.cpp
returnconstref.cpp
pointstructure.cpp > mid4(const point &, const point &)
VARIABLES
Locals
p: {...}
x: 2
y: 2
p1: {...}
x: 1
y: 1
p2: {...}
x: 3
y: 3
Registers
WATCH
56 {
57 pr.x = (p1.x + p2.x)/2;
58 pr.y = (p1.y + p2.y)/2;
59 }
61 point& mid4(const point &p1, const point &p2)
62 {
63 point p;
64 p.x = (p1.x + p2.x)/2;
65 p.y = (p1.y + p2.y)/2;
67 return p;
68 }
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
pointstructure.cpp 1
reference to local variable 'p' returned [-Wreturn-local-addr] gcc [67, 12]


R..
g++ - Build ar
passreference.cpp
returnconstref.cpp
strc_re
pointstructure.cpp 1
passparame
pointstructure.cpp > main()
VARIABLES
Locals
p1: {...}
x: 1
y: 1
p2: {...}
x: 3
y: 3
pv: {...}
pr: {...}
prr: {...}
pp: 0x55555556aeb0
Registers
WATCH
16 point p1{1,1};
17 point p2{3,3};
18 point pv, pr, prr;
19 point *pp = NULL;
20
21 pv = mid1(p1, p2);
22 pp = mid2(p1, p2);
23 mid3(p1, p2, pr);
24 prr = mid4(p1, p2);
Exception has occurred.
Segmentation fault
25
26 cout << "Calling mid1,the middle point is:("<< pv.x << "," << pv.y << ")" << endl;
27 cout << "Calling mid2,the middle point is:("<< pp->x << "," << pp->y << ")" << endl;
28 cout << "Calling mid3,the middle point is:("<< pr.x << "," << pr.y << ")" << endl;
29 cout << "Calling mid4,the middle point is:("<< prr.x << "," << prr.y << ")" << endl;
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL

```
36  point mid1(const point &p1, const point &p2)
37  {
38      point pv;
39      pv.x = (p1.x + p2.x)/2;
40      pv.y = (p1.y + p2.y)/2;
41
42      return pv;
43  }
44
45  point* mid2(const point &p1, const point &p2)
46  {
47      point* pp = new point;
48      pp->x = (p1.x + p2.x)/2;
49      pp->y = (p1.y + p2.y)/2;
50
51      return pp;
52  }
53
54  void mid3(const point &p1, const point &p2, point &pr)
55  {
56      pr.x = (p1.x + p2.x)/2;
57      pr.y = (p1.y + p2.y)/2;
58  }
59
```

return a local structure variable is ok, but less efficient

return a local structure pointer which is allocated memory by new, is ok.

The function does not return anything.
The third parameter is a reference parameter, modifying the value of the parameter is exactly changing that of the argument.

Difference between reference and pointer

- The reference must be initialized when it is created; the pointer can be assigned later.
- The reference can not be initialized by NULL; the pointer can.
- Once the reference is initialized, it can not be reassigned to other variable; a pointer can be changed to point to other object.
- **sizeof(reference)** operation returns the size of the variable; **sizeof(pointer)** operation returns the size of pointer itself.

Building shared libraries

- Many compilers allows you to build your functions into shared libraries so that you can use those functions later.
- Shared library in linux are **.so** files.

Building shared libraries

- Suppose we have written the following code:

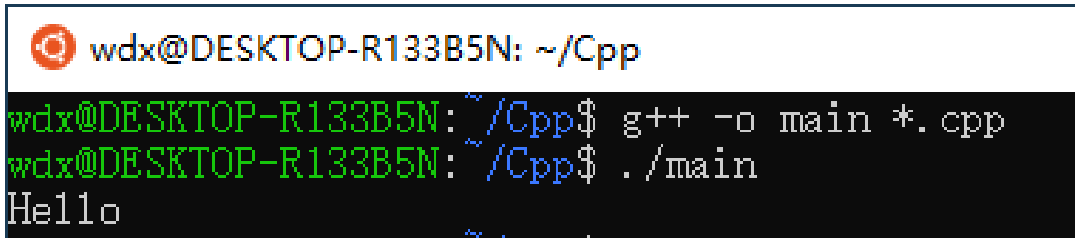
```
// function.cpp
#include <iostream>
#include "function.h"
using std::endl;
using std::cout;

void printHello() {
    cout<<"Hello"<<endl;
}
```

```
// function.h
#pragma once

void printHello();
```

```
// main.cpp
#include <iostream>
#include "function.h"

int main() {
    printHello();

    return 0;
}
```

Building shared libraries

- In previous class we do the following:
- This will compile the “main.cpp” and “function.cpp” into “main”
- And then run “main”



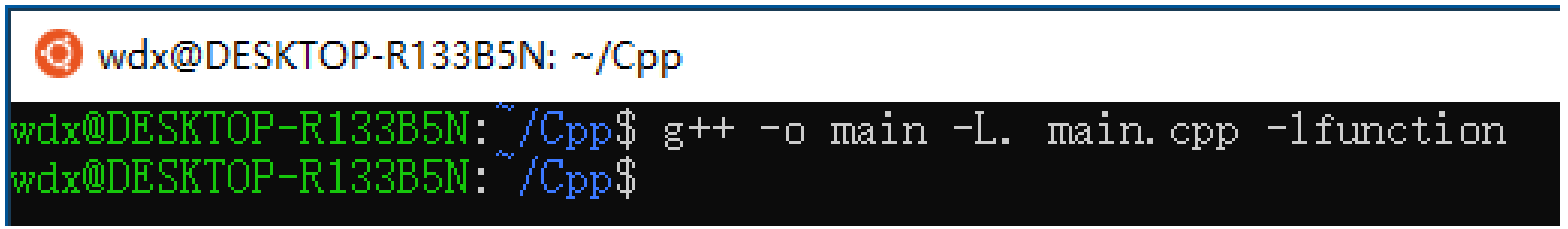
Building shared libraries

- Let’s build a shared library:
- Remember to use arguments “**-shared**” and “**-fPIC**” when building it.
- Now we should see “libfunction.so” in the directory

```
wdx@DESKTOP-R133B5N: ~/Cpp
wdx@DESKTOP-R133B5N:~/Cpp$ g++ -shared -fPIC -o libfunction.so function.cpp
wdx@DESKTOP-R133B5N:~/Cpp$
```

Using shared library

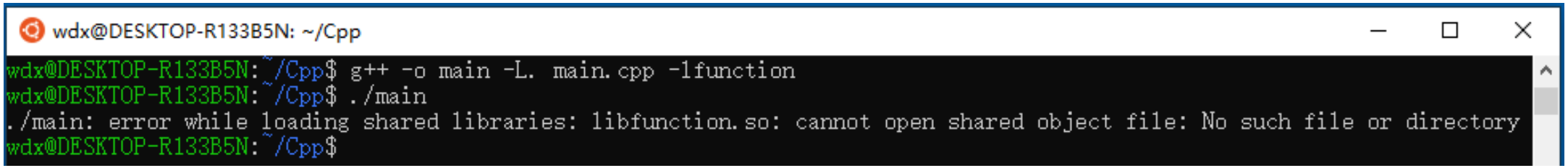
- Now we can use “printHello” function with the “.h” header file and the “.so” shared library.
- Let’s compile “main” again:



- Use “-L.” to tell it to find libraries in current directory.
- Use “-lfunction” to tell it to use “libfunction.so”.

Using shared library

- After the “main” has been compiled, try to run it:



- It failed because “main” now relys on “libfunction.so”. You must tell the terminal where to find “libfunction.so”.

Using shared library

- Using export command to set environment variable “LD_LIBRARY_PATH”
- And then run “main” again

```
wdx@DESKTOP-R133B5N: ~/Cpp
wdx@DESKTOP-R133B5N:~/Cpp$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
wdx@DESKTOP-R133B5N:~/Cpp$ echo $LD_LIBRARY_PATH
.:
wdx@DESKTOP-R133B5N:~/Cpp$ ./main
Hello
wdx@DESKTOP-R133B5N:~/Cpp$
```

export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH

There is no space on either side of the equal sign
. indicates the current directory

Another choice is to move(or copy) your .so file to /usr/lib folder by mv or cp command

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ sudo cp libfunction.so /usr/lib
[sudo] password for maydlee:
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ ./main
Hello
```

Exercise 1

Define three functions that swap two values of integer, please use integer arguments, pointer arguments and reference arguments respectively. Write a test program to call these functions and display the result.

You are required to compile these functions into a shared library “libswap.so”, and then compile and run your program with this shared library.

Exercise 2

Define a function whose prototype is **char* match(char* s, char ch);**
s is a C-style string, **ch** is a character. If the ch is in the s, return the position of s at ch; if the ch is not in the s, return NULL.
Write a test program to call the function and show the result. The output sample as follows:

```
Please input a string:
Enjoy the holiday.
Please input a character:
h
he holiday.
```

```
Please input a string:
Class is over.
Please input a character:
m
Not Found
```

Exercise 3

```
struct point{
float x;
float y;
};
```

Here is a structure declaration:

(1) Define a function that passes a point structure by value to set values for two points.

(2) Define another function that passes the address of a point structure to calculate the middle point of two points.

(3) Write a simple program to call these two function and display the results.