FUNDAMENTALS OF STRUCTURED PROGRAMMING

Lecture 8

Functions III (User-Defined:Pass by Reference)

Course Coordinator: Prof. Zaki Taha Fayed

Presented by: Dr. Sally Saad

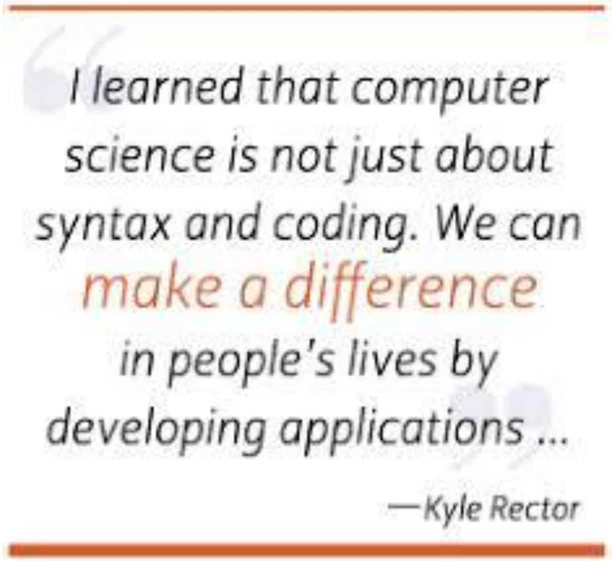
SallySaad@gmail.com

DropBox folder link

https://www.dropbox.com/sh/85vnrgkfqgrzhwn/AABdwKLJZqZs26a7u-y0AFwia?dl=0

Credits to Dr.Salma Hamdy for content preparation

Quotes of the Day!



I learned that computer science is not just about syntax and coding. We can make a difference in people's lives by developing applications ...

—*Kyle Rector*

Procedural Abstraction - Functions III

1. Programmer-defined Functions – (cont.)

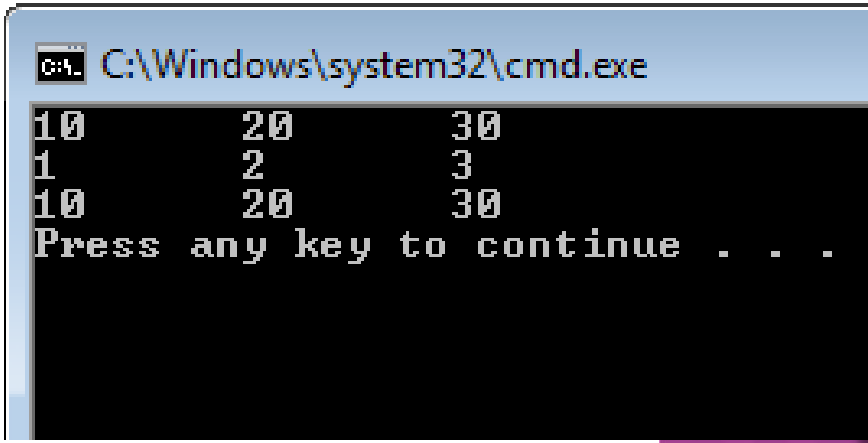
Remember Calling by Value

- When you pass an argument variable, the value of the variable is plugged/copied into the function's formal parameter, hence the name *call by value*.
- Arguments vs. formal parameters

1. Programmer-defined Functions – (cont.)

Remember Calling by Value

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

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

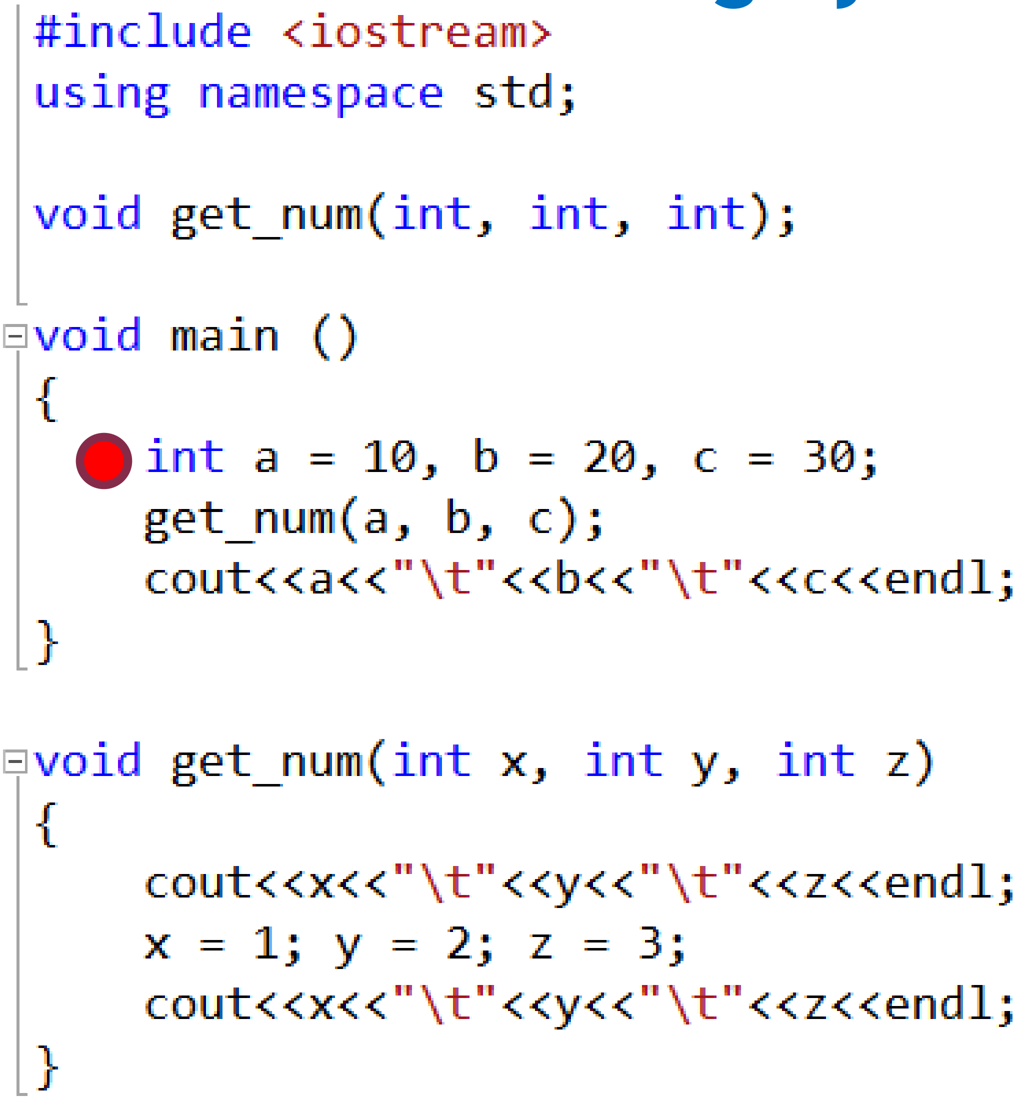


```
10      20      30
1       2       3
10      20      30
Press any key to continue . . .
```

Call by value:
Arguments and formal parameters are totally different places
Any change in the formal parameters does not affect the argument variables

1. Programmer-defined Functions – (cont.)

Remember Calling by Value

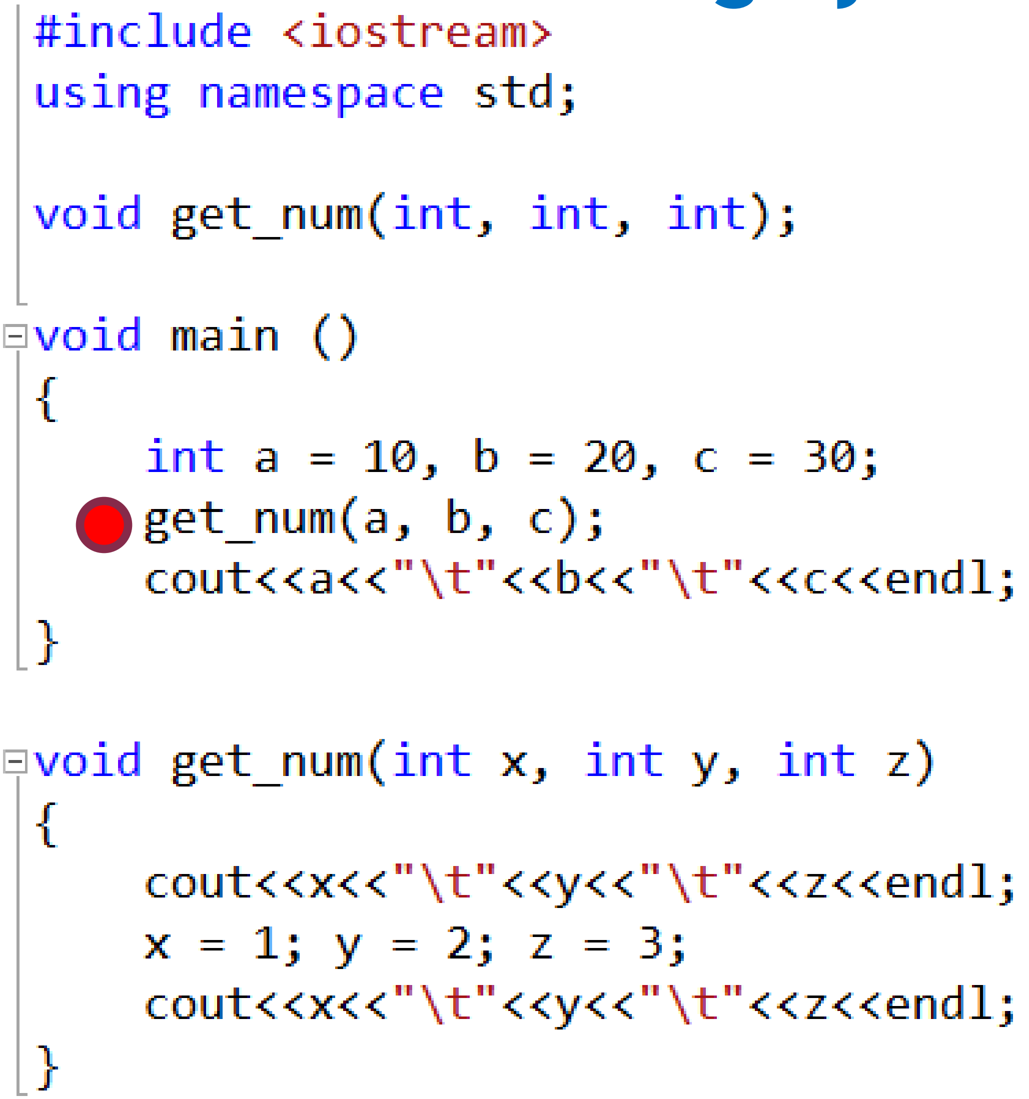


```
#include <iostream>
using namespace std;

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

Memory location	value	
…		
1006		
1008		
1010		
1012		
1014		
1016		
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

Remember Calling by Value

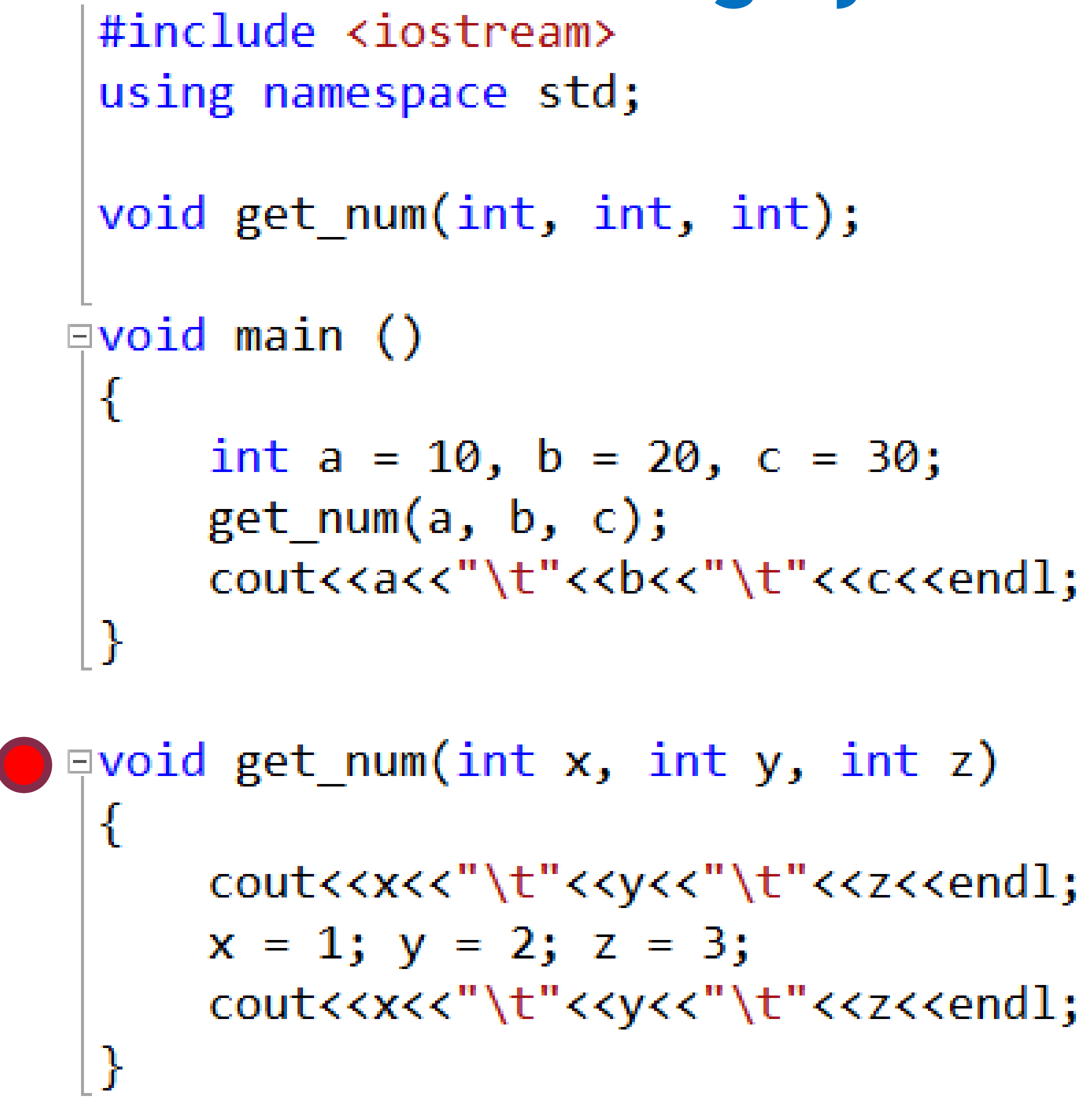


```
#include <iostream>
using namespace std;

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014		
1016	30	c
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

Remember Calling by Value

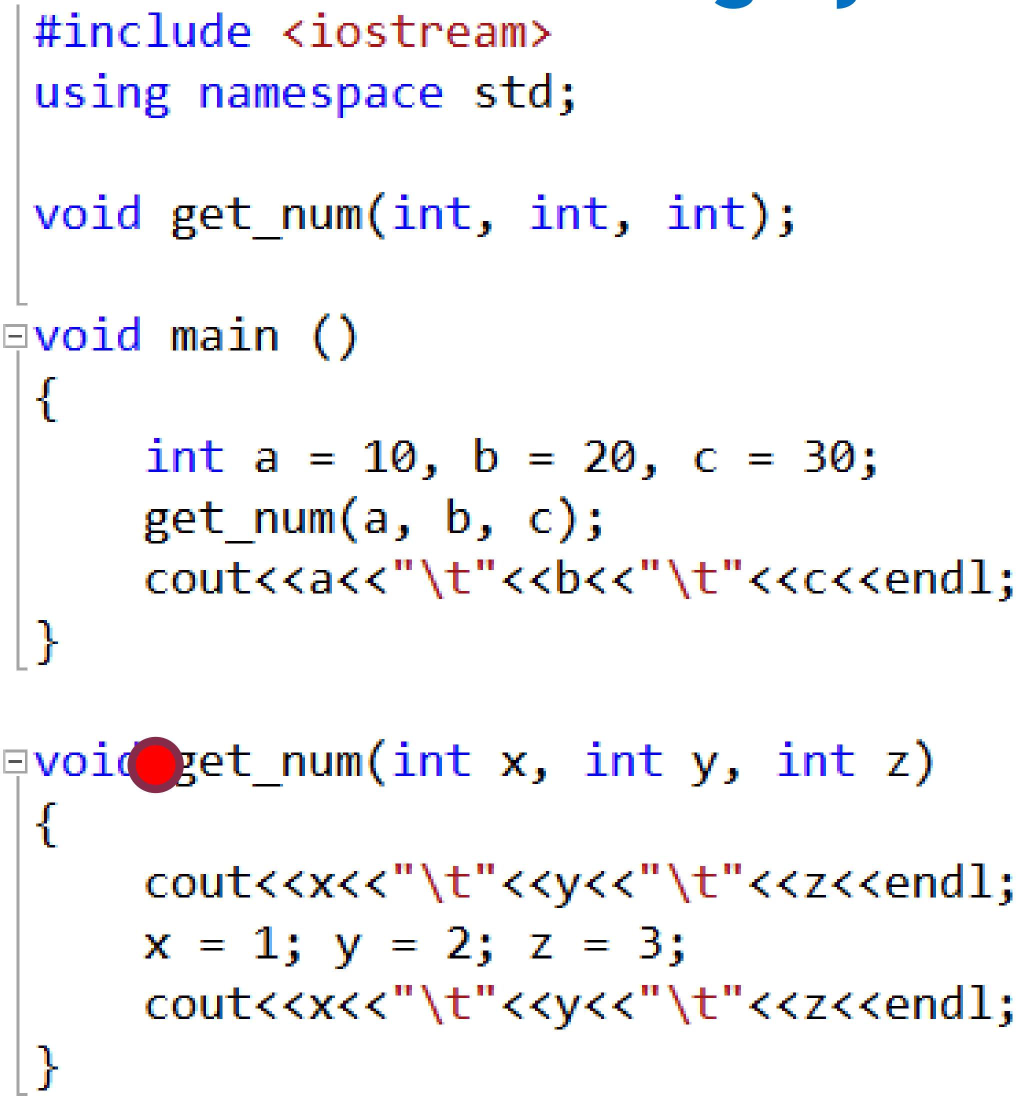


```
#include <iostream>
using namespace std;

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

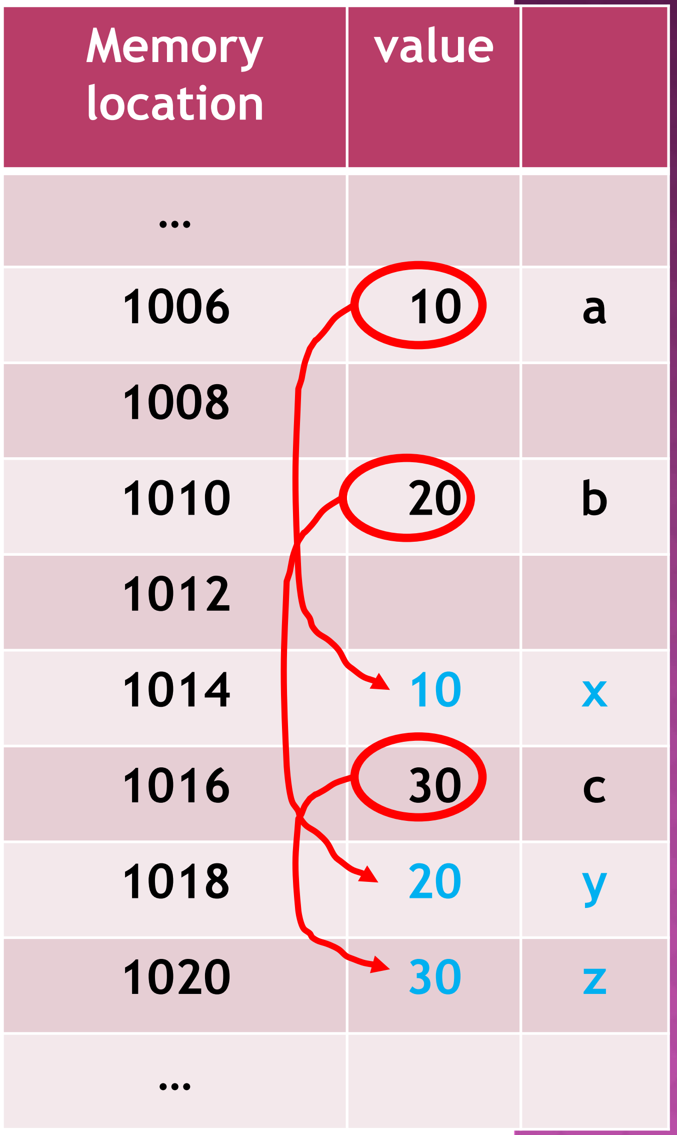
Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014		x
1016	30	c
1018		y
1020		z
…		

1. Programmer-defined Functions – (cont.)

Remember Calling by Value



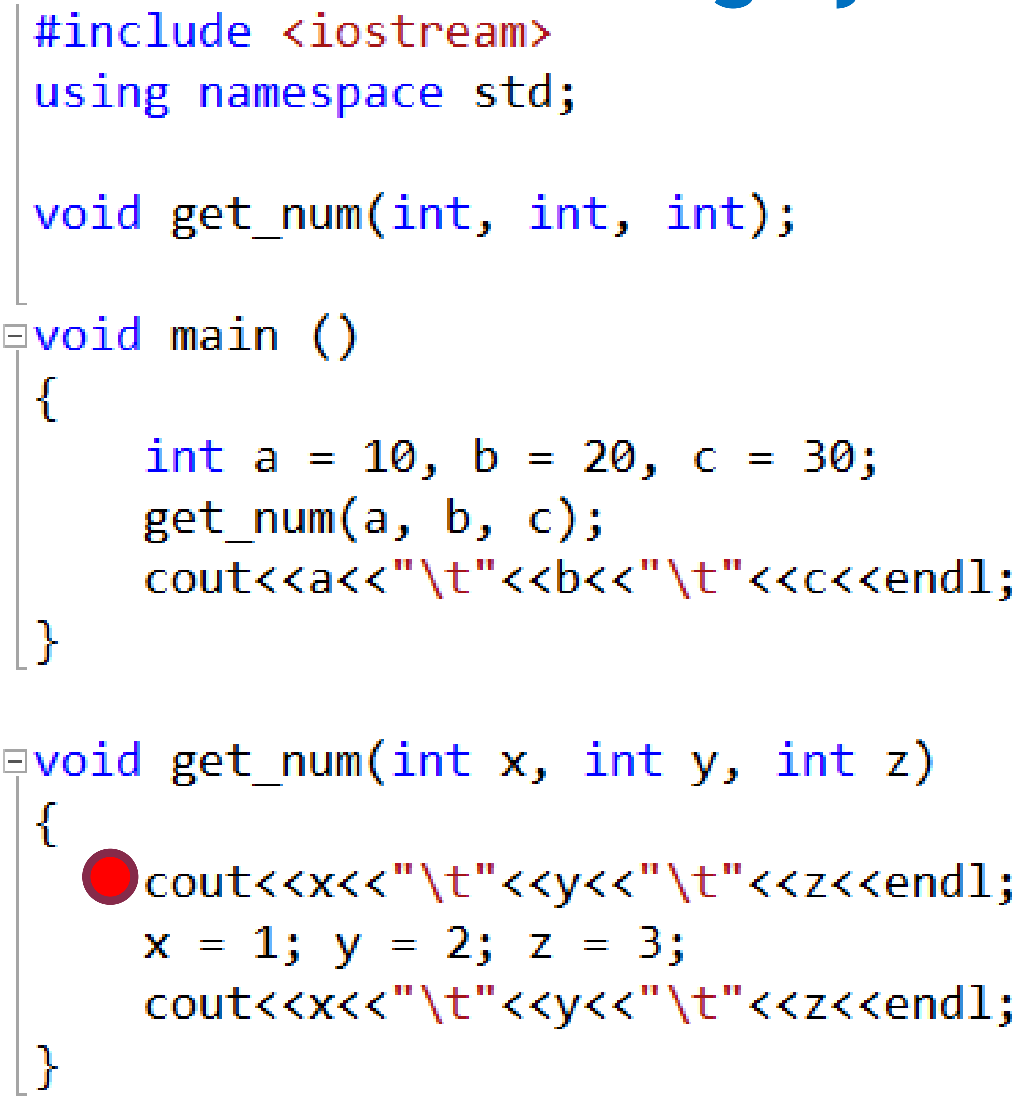
```
#include <iostream>
using namespace std;

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```



Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014	10	x
1016	30	c
1018	20	y
1020	30	z
…		

1. Programmer-defined Functions – (cont.)

Remember Calling by Value

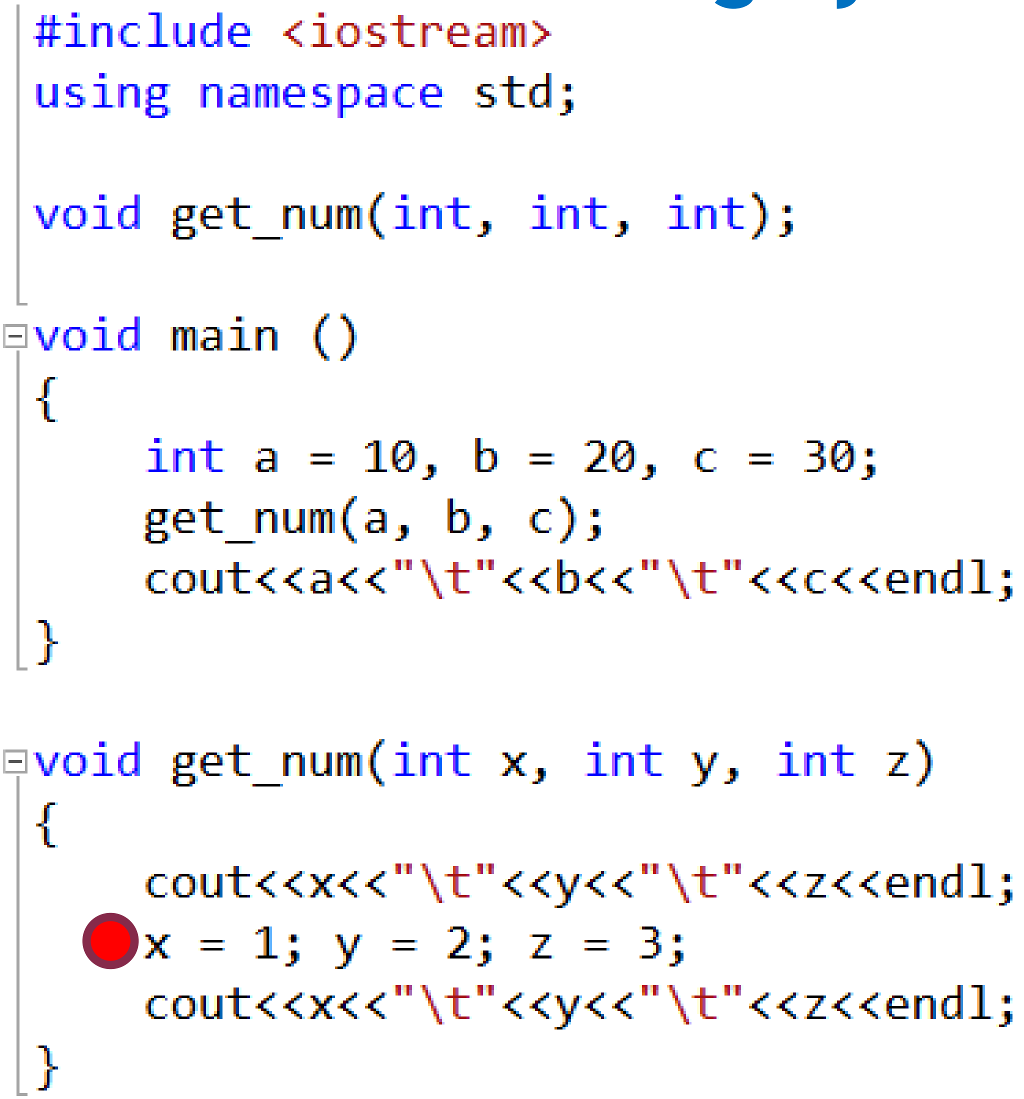


```
#include <iostream>
using namespace std;

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
●   cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014	10	x
1016	30	c
1018	20	y
1020	30	z
…		

1. Programmer-defined Functions – (cont.)

Remember Calling by Value

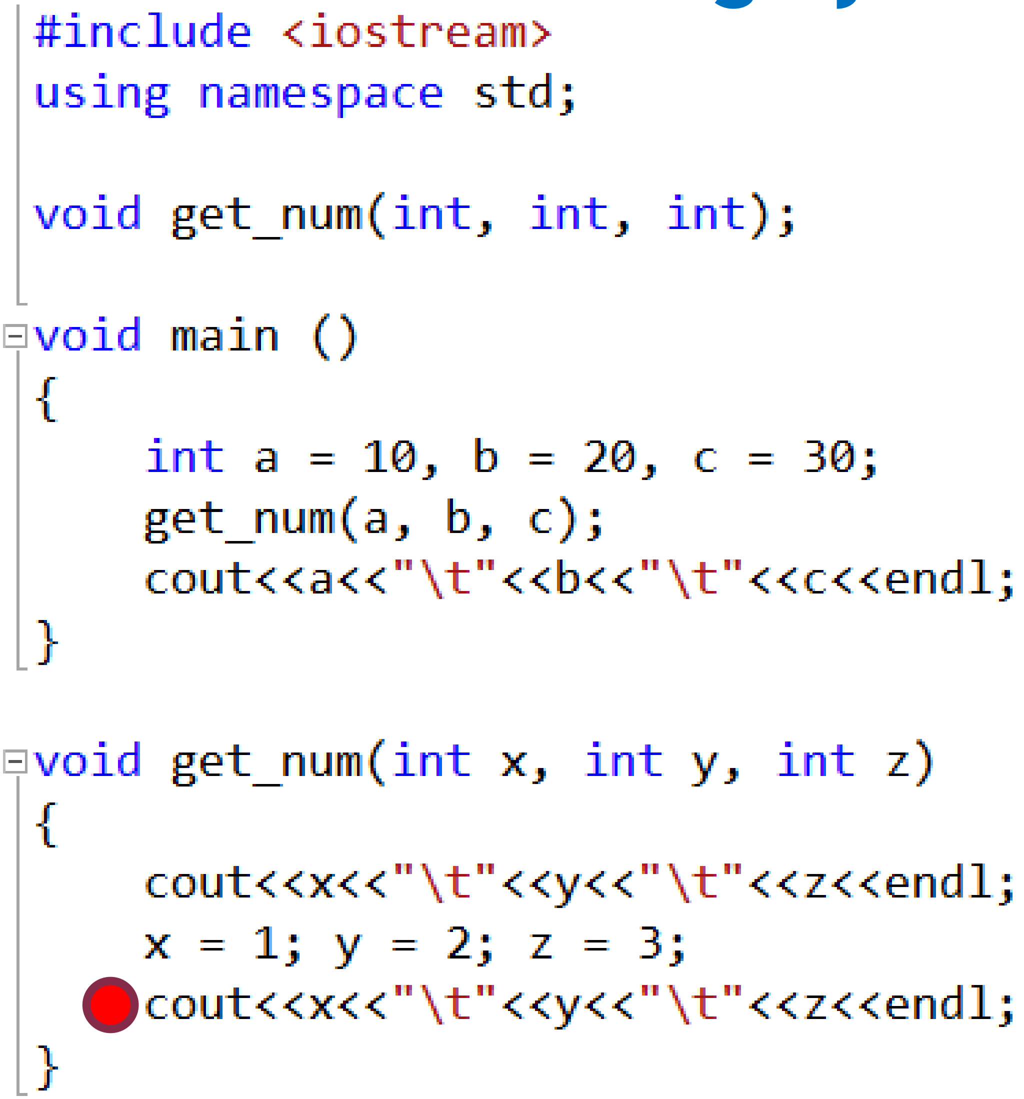


```
#include <iostream>
using namespace std;

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
  ● x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014	1	x
1016	30	c
1018	2	y
1020	3	z
…		

1. Programmer-defined Functions – (cont.)

Remember Calling by Value

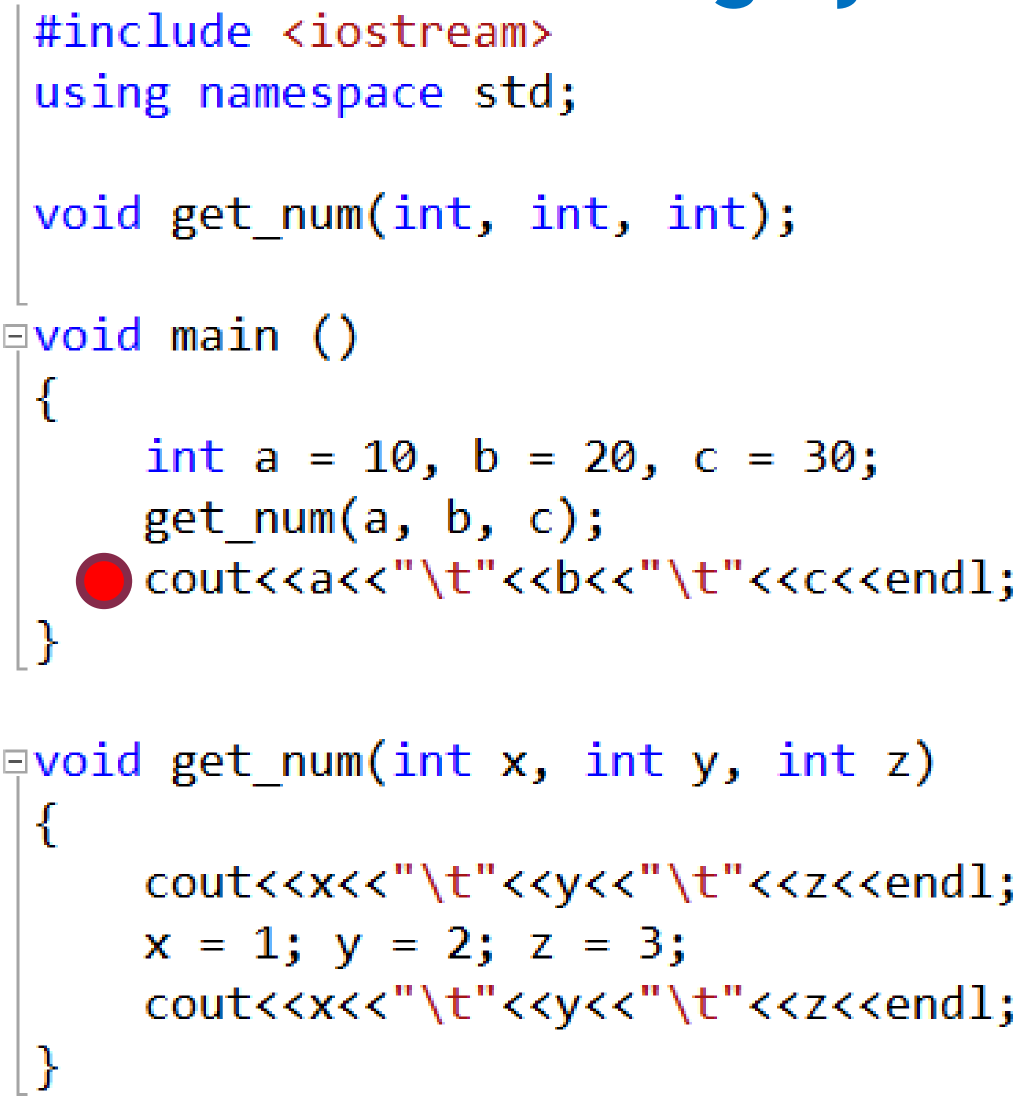


```
#include <iostream>
using namespace std;

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014	1	x
1016	30	c
1018	2	y
1020	3	z
…		

1. Programmer-defined Functions – (cont.)

Remember Calling by Value



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

void get_num(int, int, int);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int x, int y, int z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014		
1016	30	c
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

(9) Passing Different Types

- Formal parameters, and hence arguments, can be of simple types (int, float, char, …), or of aggregate types (struct,..) .

(10) Returning Different Types

- A function's return value, can be of simple type (int, float, char, …), or of aggregate type(struct,..) .

1. Programmer-defined Functions – (cont.)

Example 1 - Remember Lec8Ex1.cpp

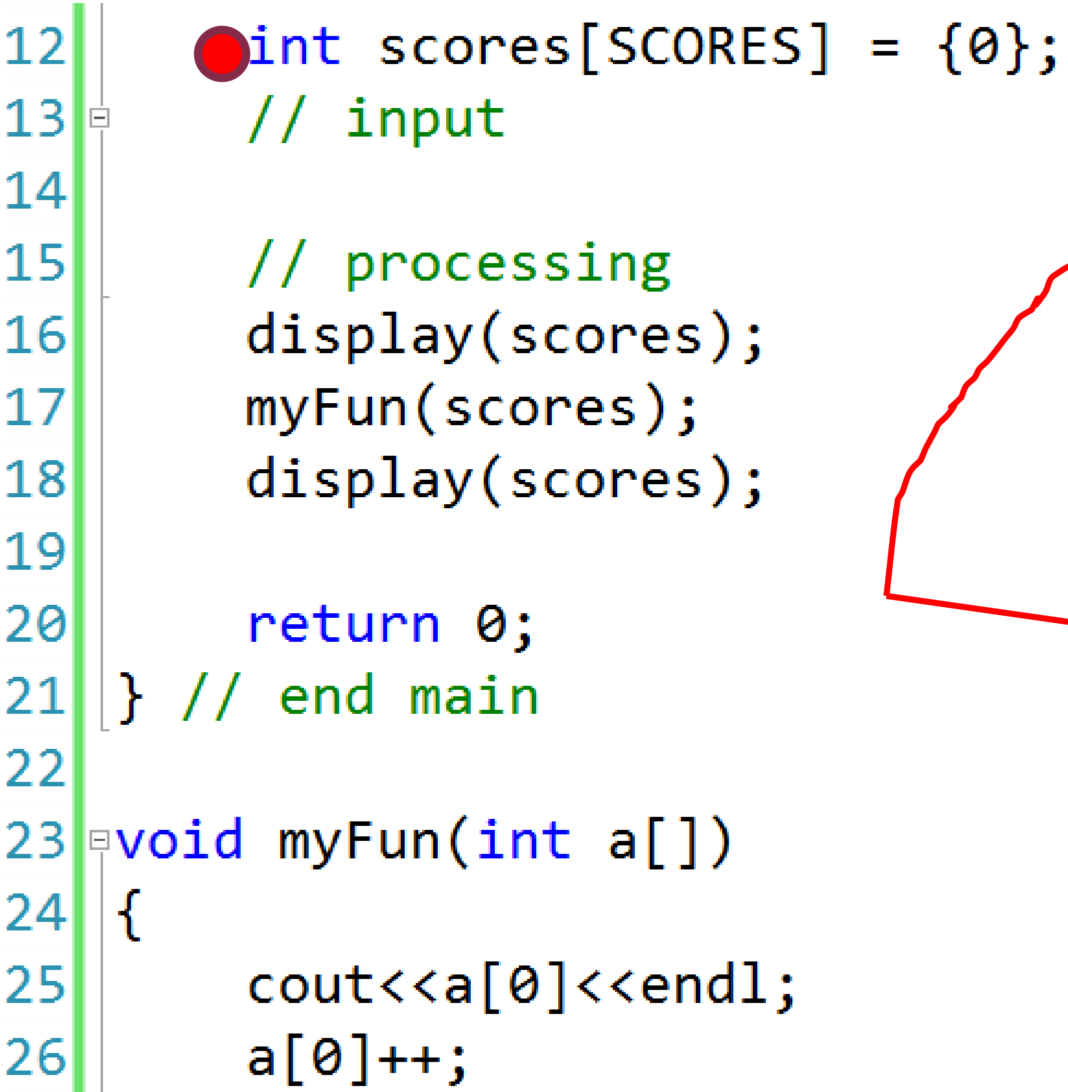
- Function to output structure fields.
- Function to input (fill) structure fields.

Example 2 - Lec8Ex2.cpp

- Function to output array elements.
- Function to input (fill) array elements.
- Overloading the display functions. (HOW?)

1. Programmer-defined Functions – (cont.)

Example 2 - Lec8Ex2.cpp

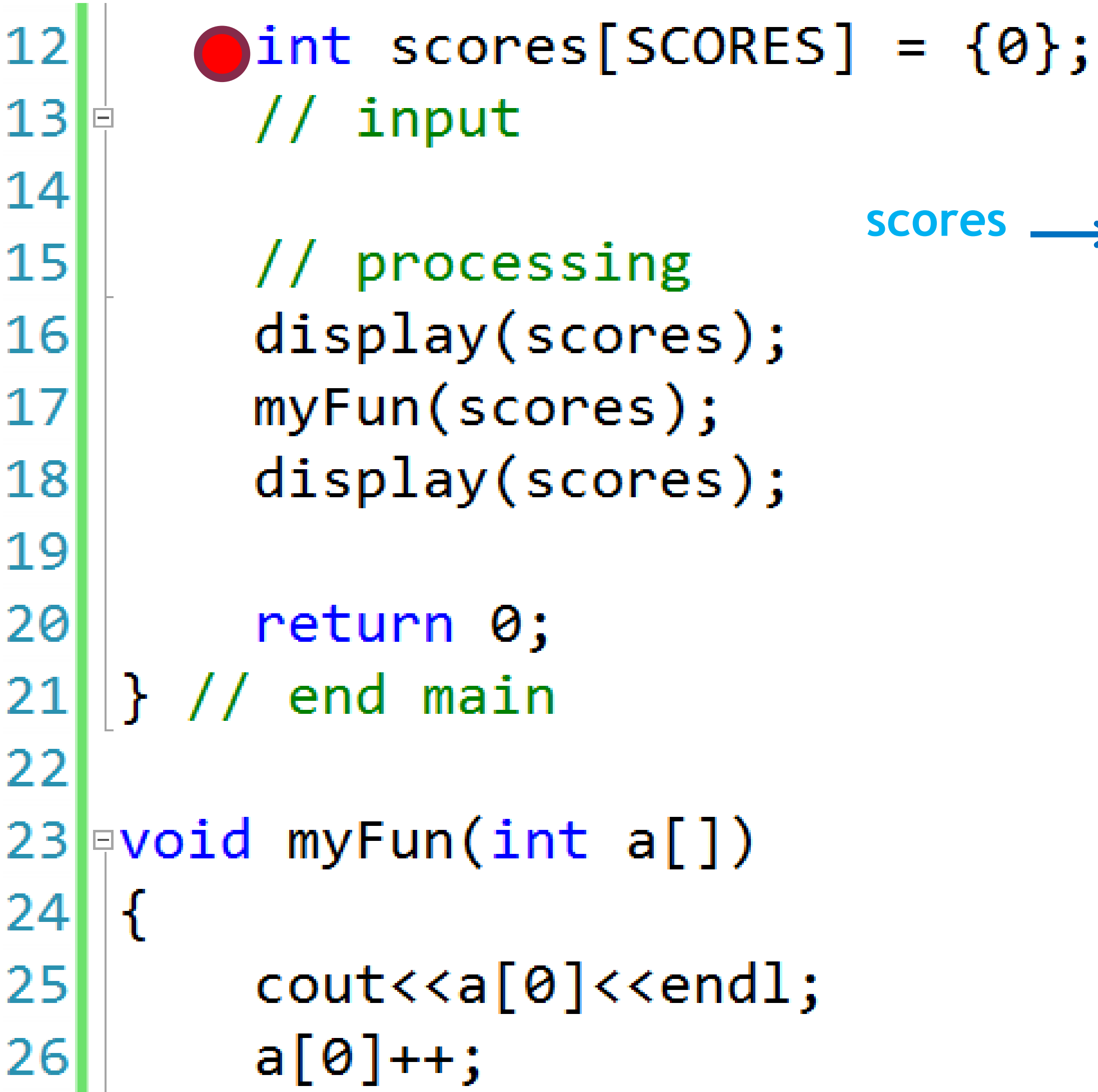


```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

Memory location	value	
…		
1006		0
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

Example 2 - Lec8Ex2.cpp



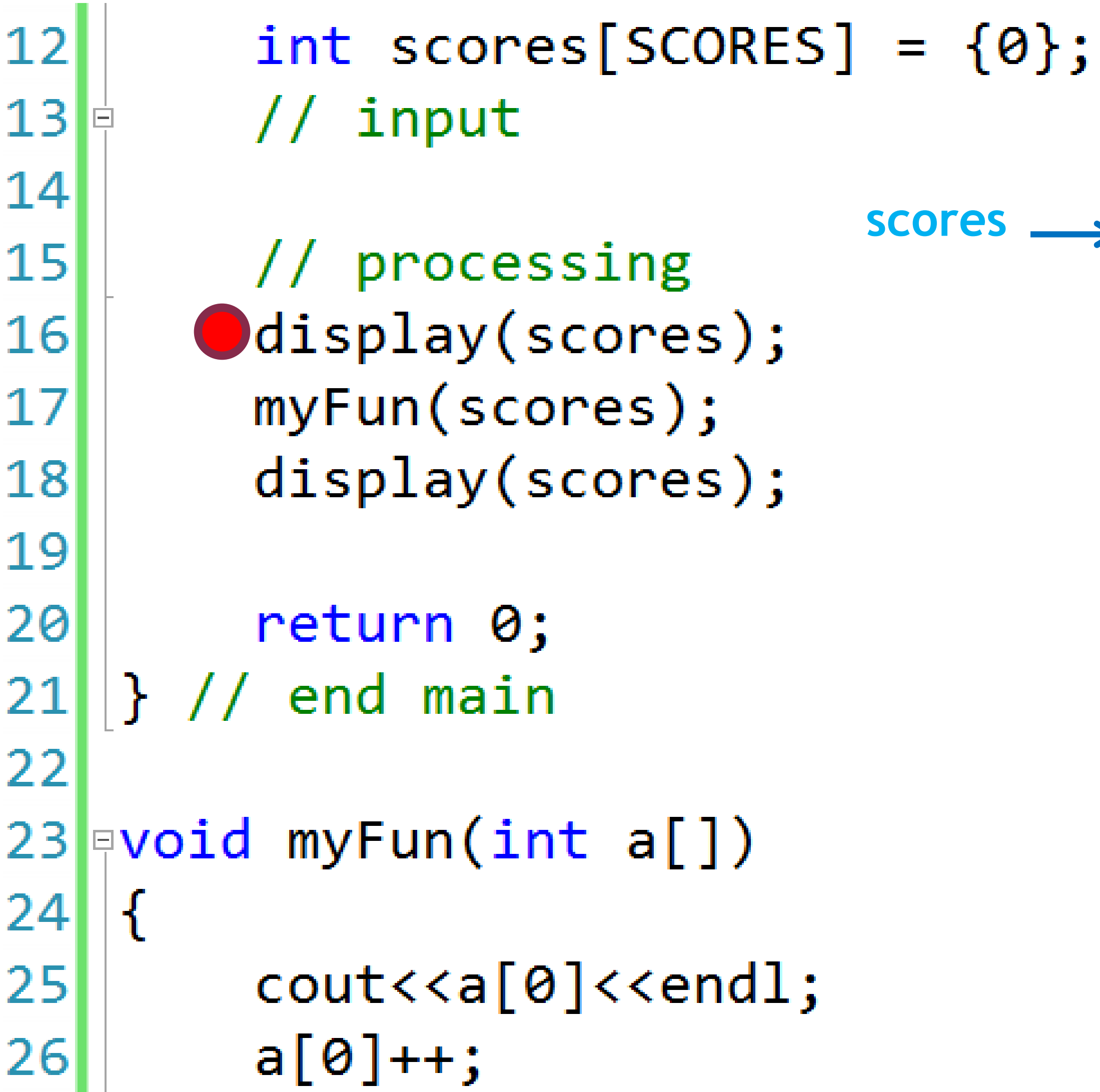
```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

scores →

Memory location	value	
…		
1006		0
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

Example 2 - Lec8Ex2.cpp



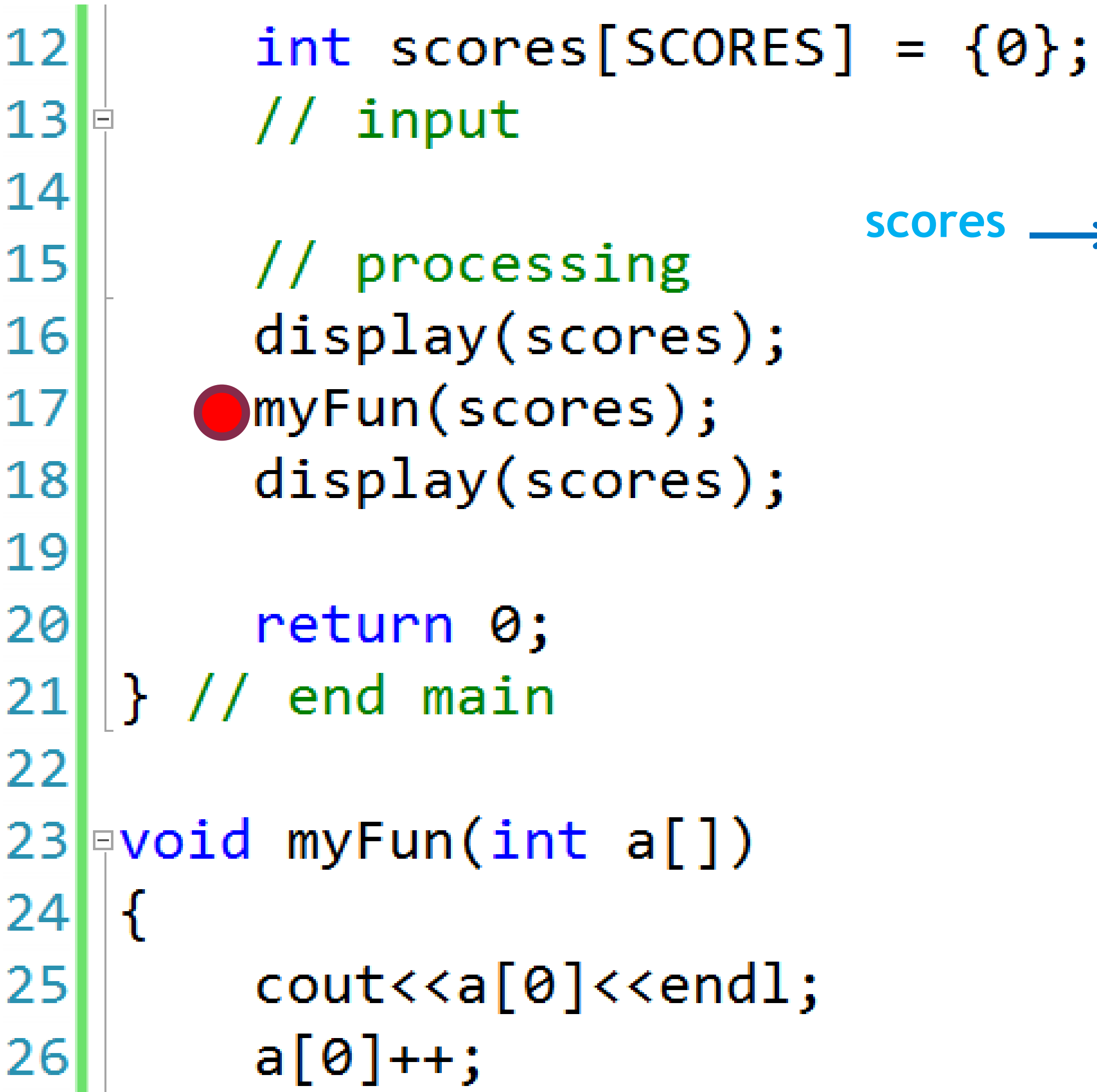
```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

scores →

Memory location	value	
…		
1006		0
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

Example 2 - Lec8Ex2.cpp



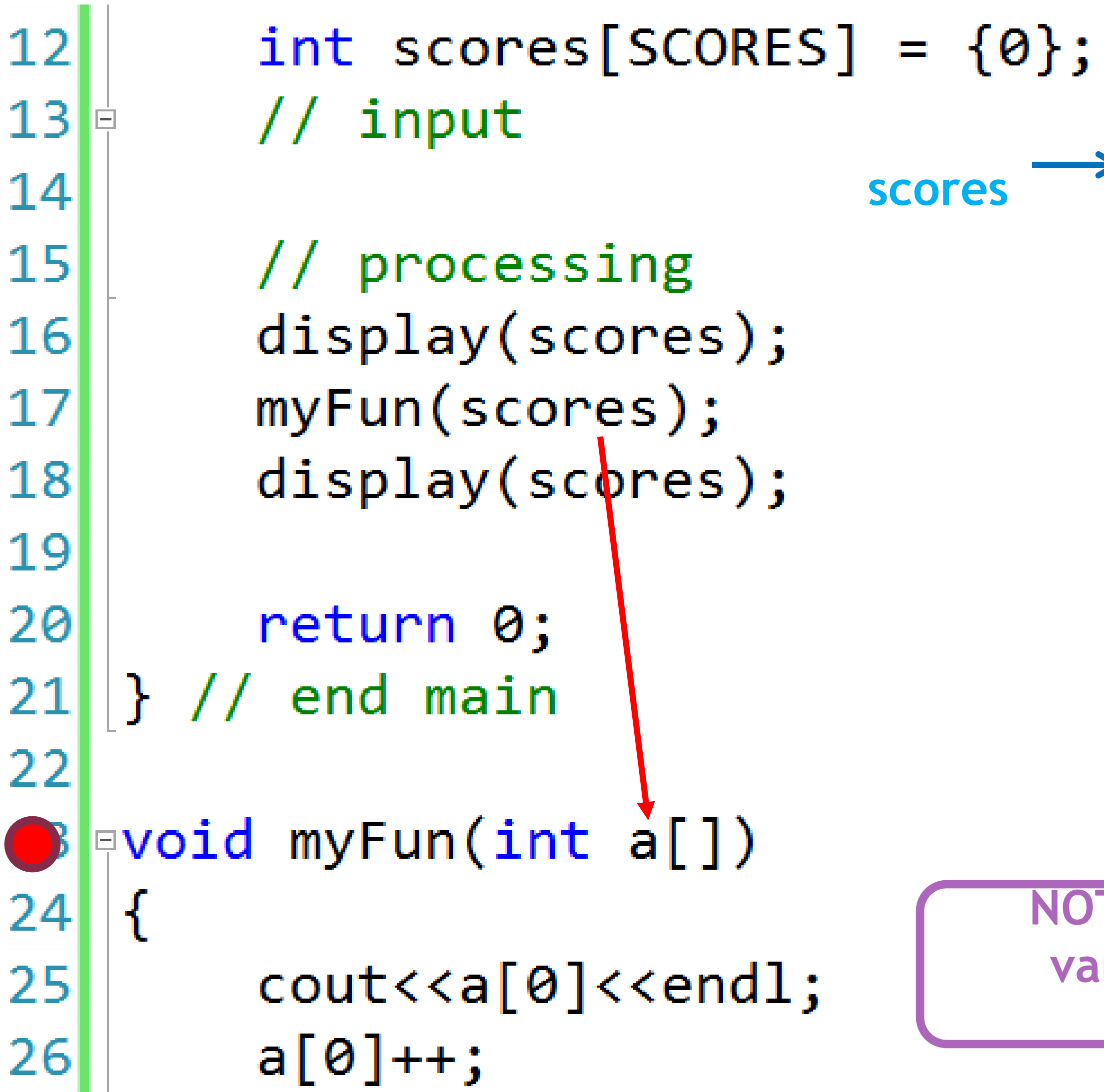
```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

scores →

Memory location	value	
…		
1006		0
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

Example 2 - Lec8Ex2.cpp



```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

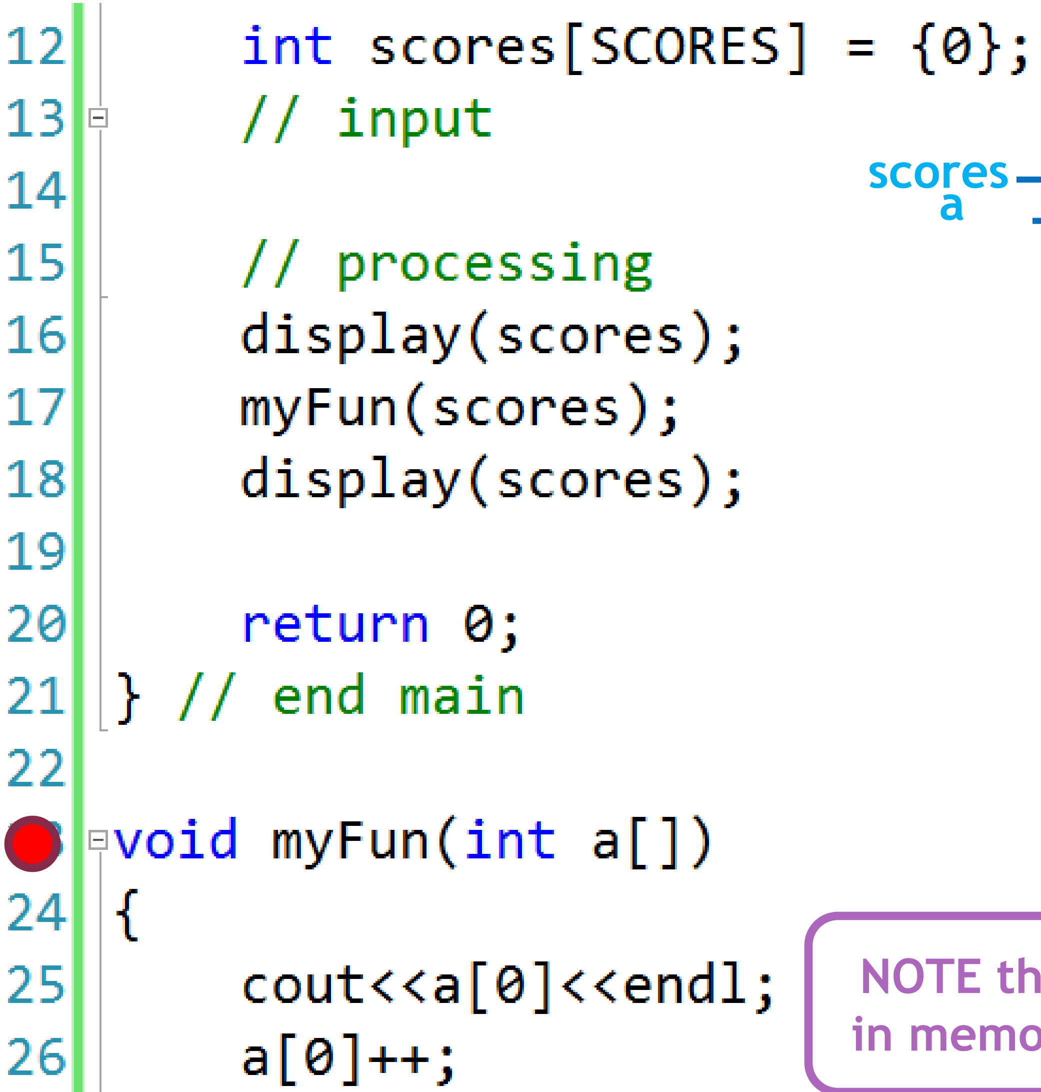
scores →

Memory location	value	
…		
1006		0
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018	1006	a
1020		
…		

NOTE that **a** is not an actual variable in memory. It is a reference variable.

1. Programmer-defined Functions – (cont.)

Example 2 - Lec8Ex2.cpp



```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

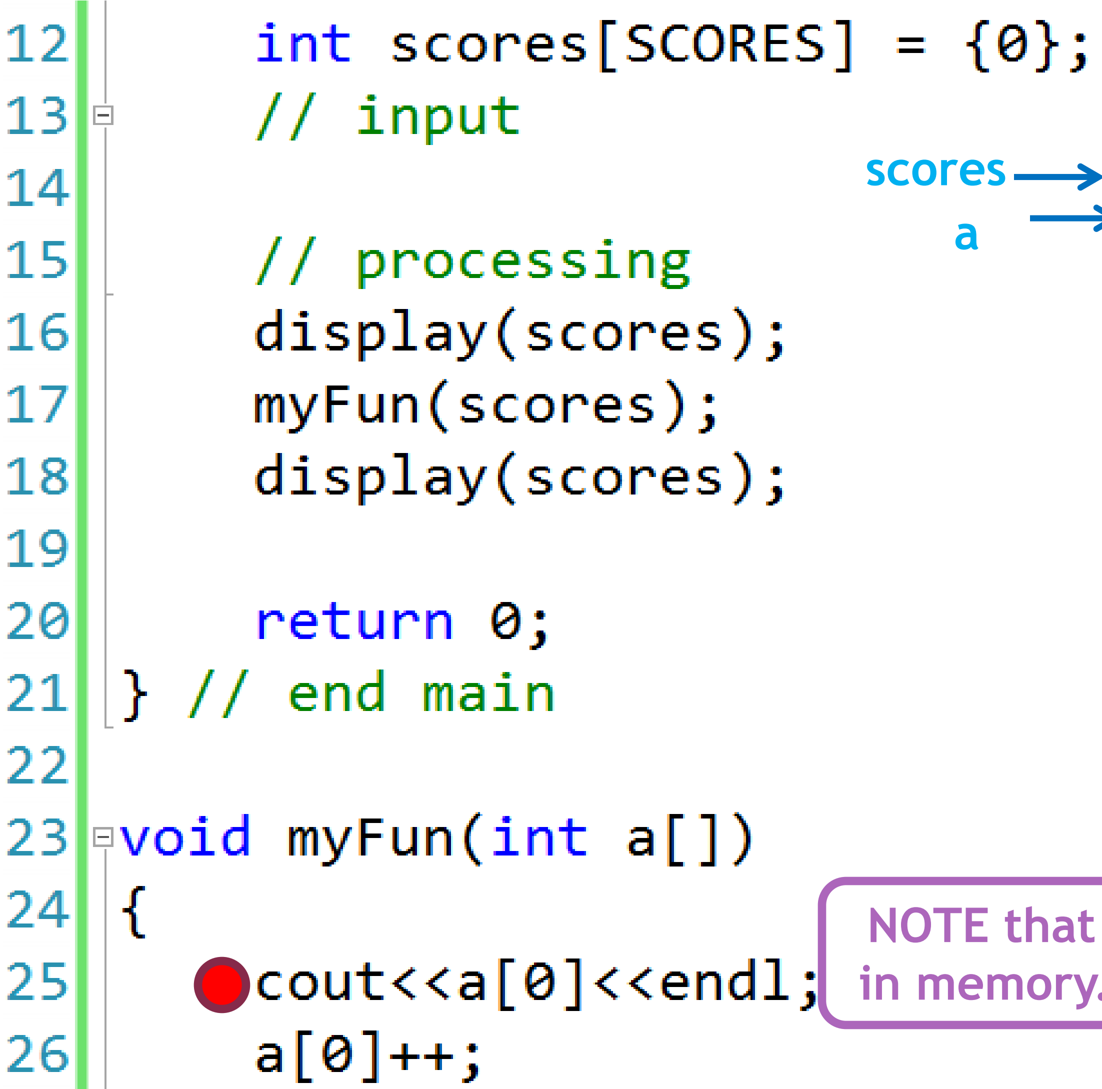
scores →
a →

Memory location	value	
…		
1006		0
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018	~~1006~~	~~a~~
1020		
…		

NOTE that **a** is not an actual variable in memory. It is a reference variable.

1. Programmer-defined Functions – (cont.)

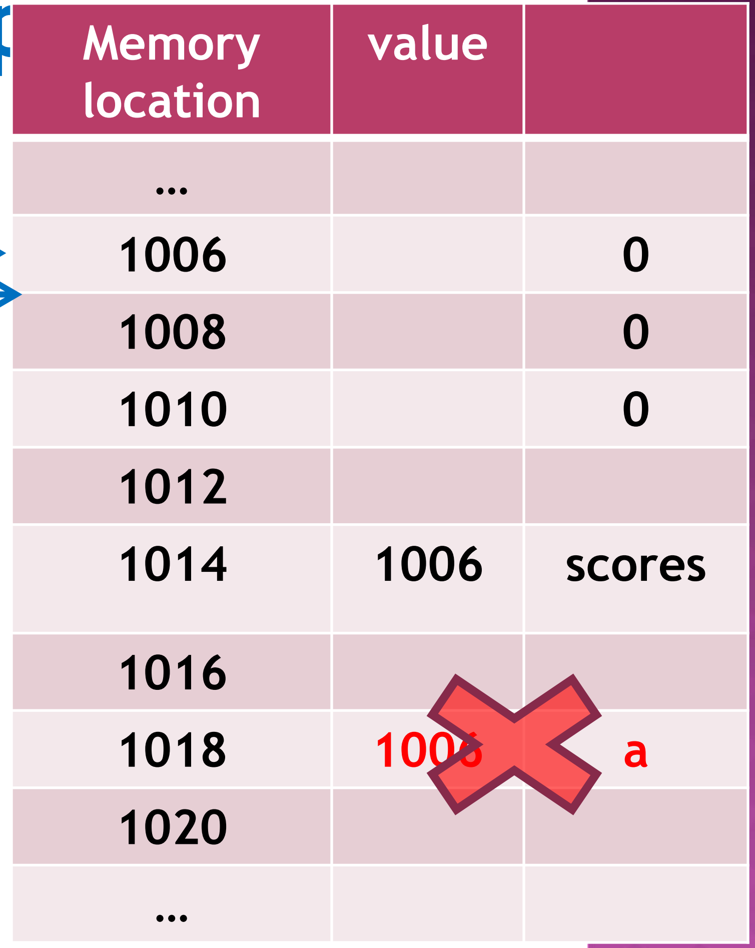
Example 2 - Lec8Ex2.cpp



```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

scores →

a →

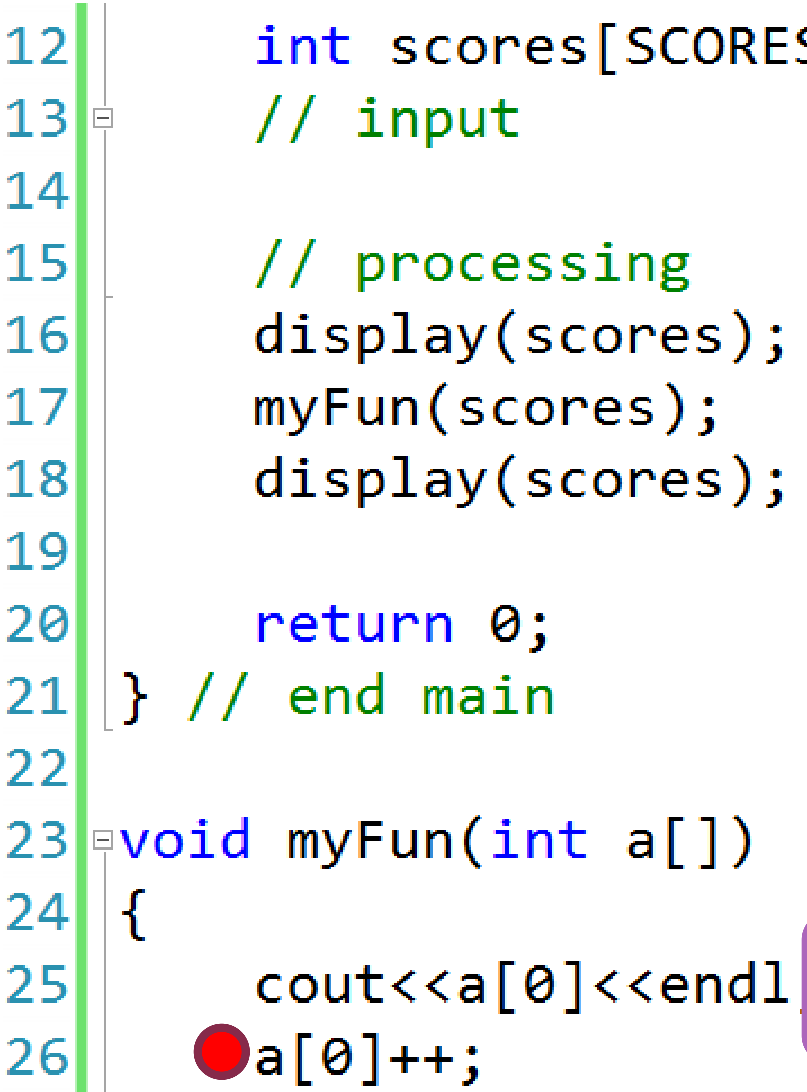


Memory location	value	
…		
1006		0
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018	1006	a
1020		
…		

NOTE that a is not an actual variable in memory. It is a reference variable.

1. Programmer-defined Functions – (cont.)

Example 2 - Lec8Ex2.cpp



```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

scores →

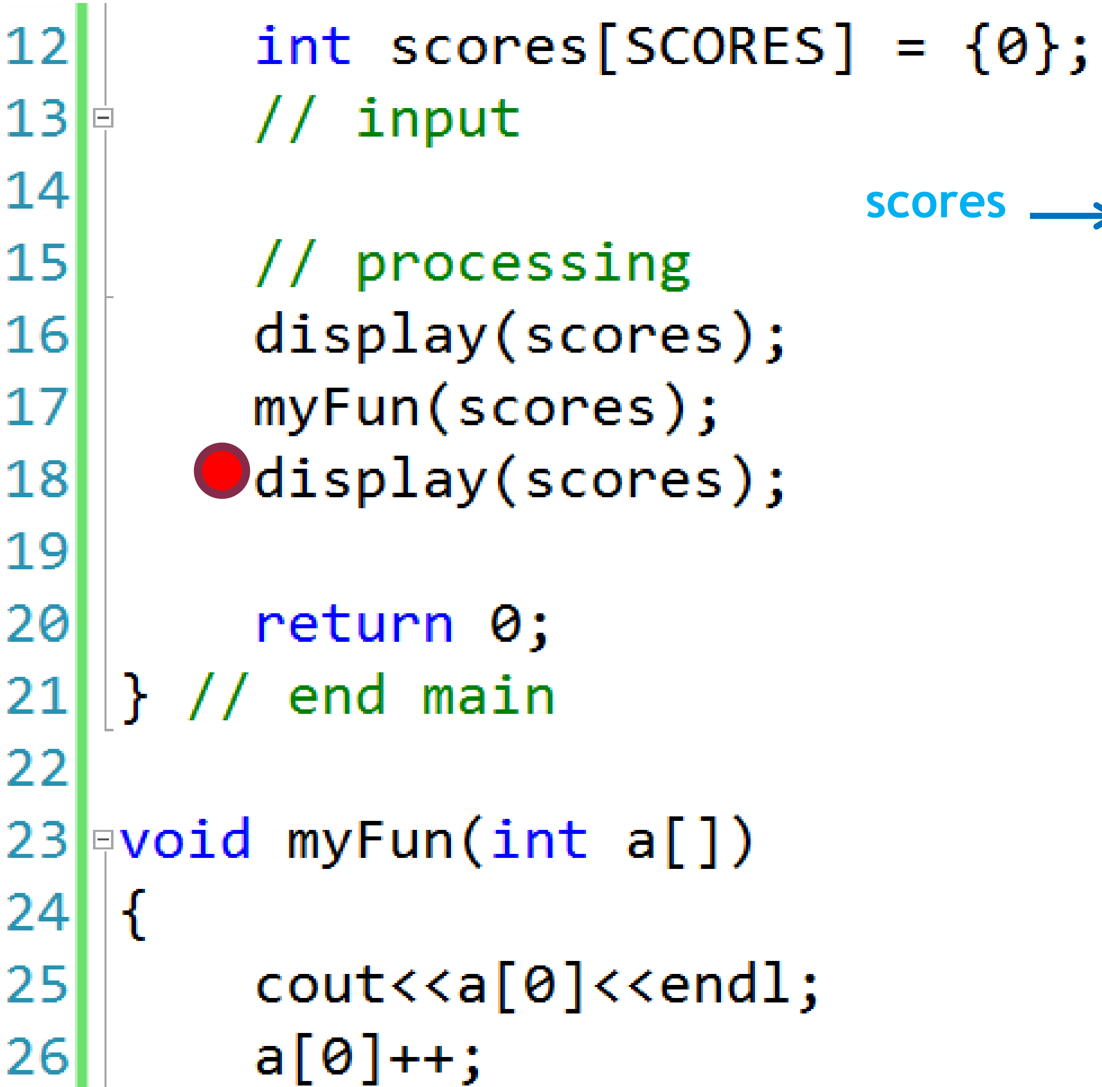
a →

Memory location	value	
…		
1006		1
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018	~~1006~~	a
1020		
…		

NOTE that a is not an actual variable in memory. It is a reference variable.

1. Programmer-defined Functions – (cont.)

Example 2 - Lec8Ex2.cpp



```
    int scores[SCORES] = {0};
    // input

    // processing
    display(scores);
    myFun(scores);
    display(scores);

    return 0;
} // end main

void myFun(int a[])
{
    cout<<a[0]<<endl;
    a[0]++;
```

scores →

Memory location	value	
…		
1006		1
1008		0
1010		0
1012		
1014	1006	scores
1016		
1018		
1020		
…		

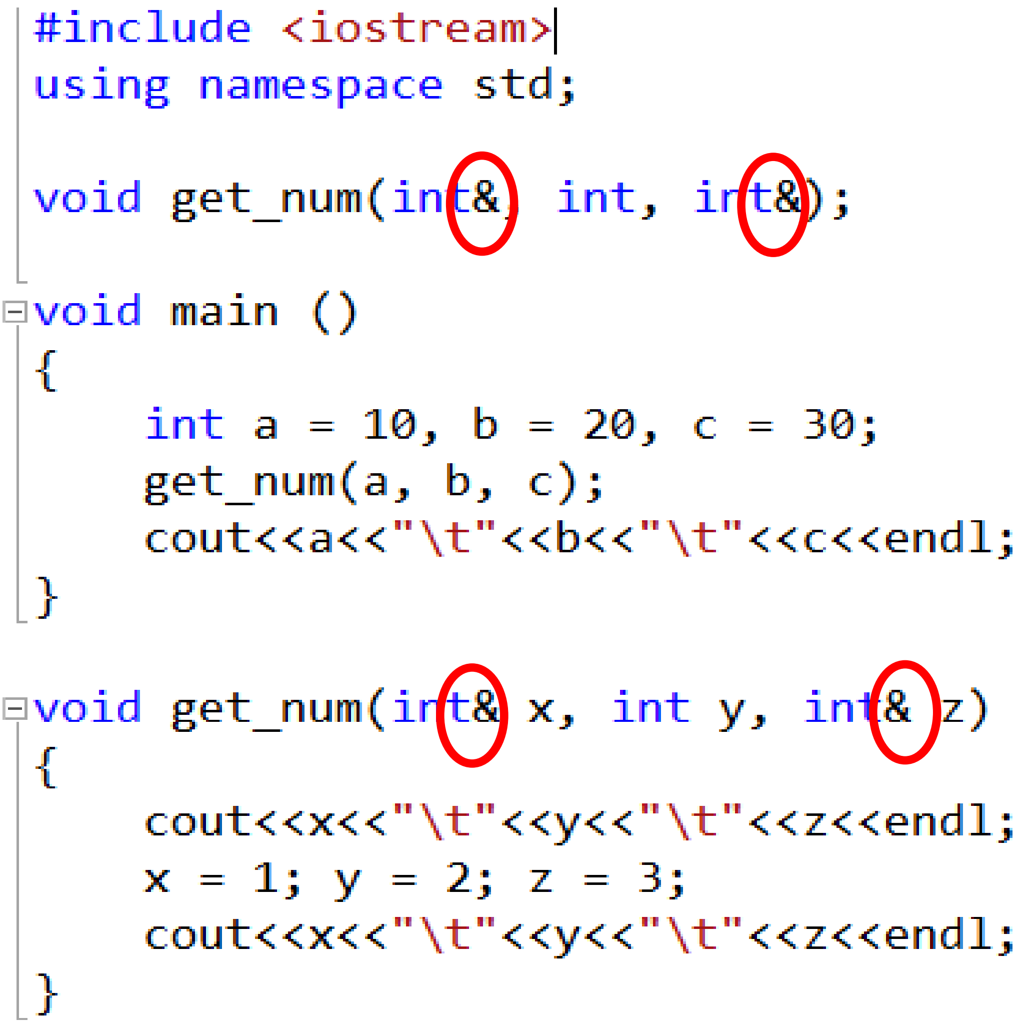
1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value

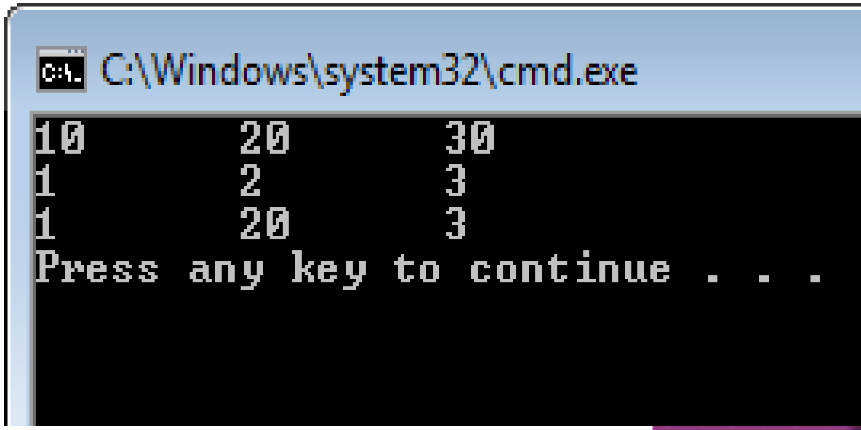
- The return statement can only return one value.
- You can “return” more than one value by passing the address of a variable instead of its value (*calling by reference*). The parameter in this case is called a reference variable.

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value



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

void get_num(int&, int, int&);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int& x, int y, int& z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```



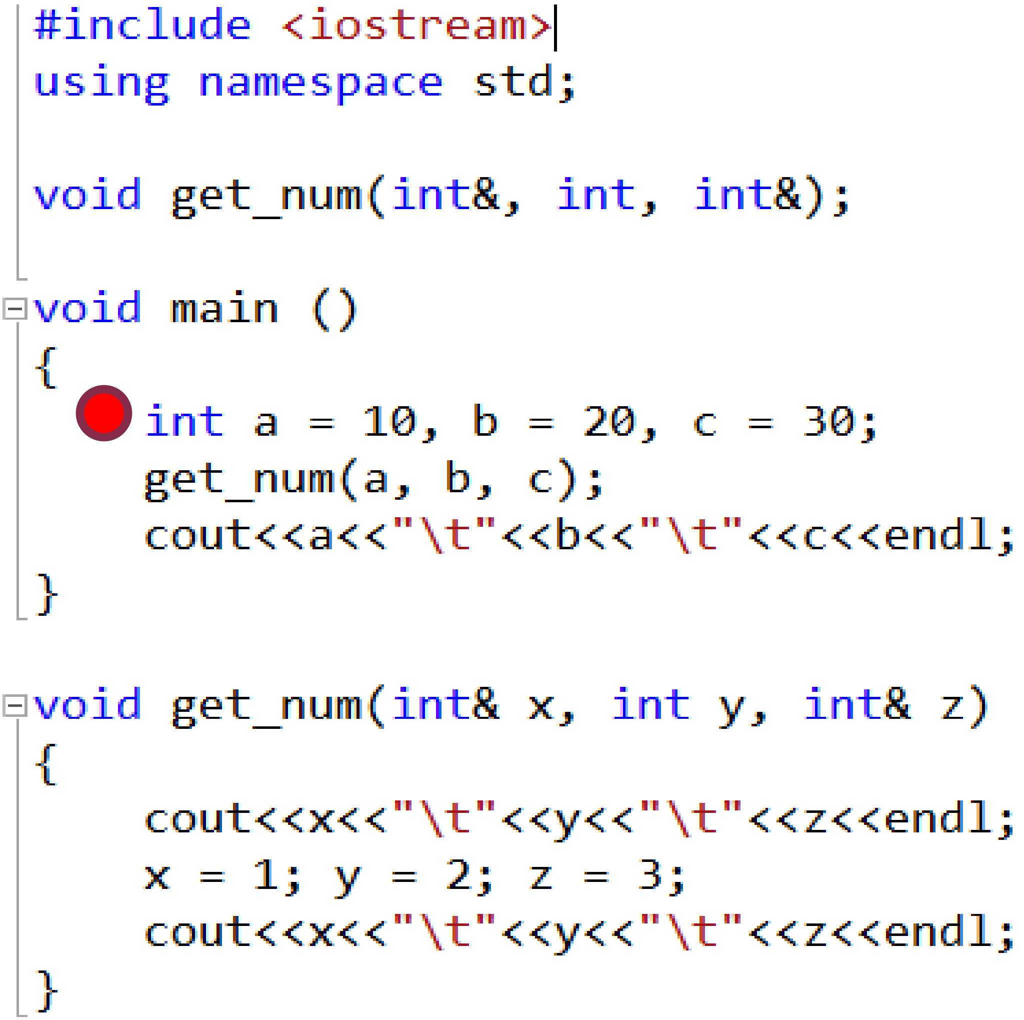
```
10      20      30
1       2       3
1       20      3
Press any key to continue . . .
```

Call by reference:

- Arguments and formal parameters are the same places in memory.
- Any change in the formal parameters WILL AFFECT the argument variables.

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value

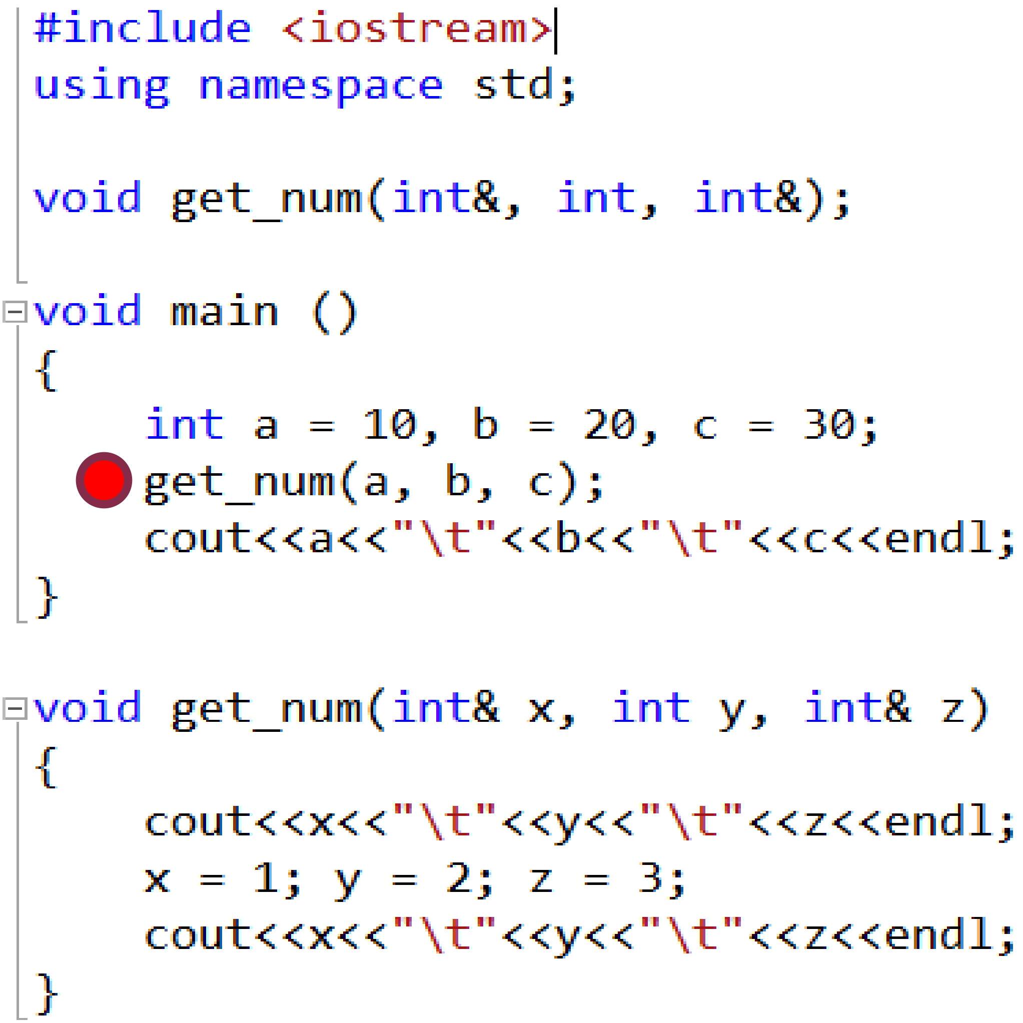


```
#include <iostream>
using namespace std;

void get_num(int&, int, int&);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int& x, int y, int& z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

Memory location	value	
…		
1006		
1008		
1010		
1012		
1014		
1016		
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value

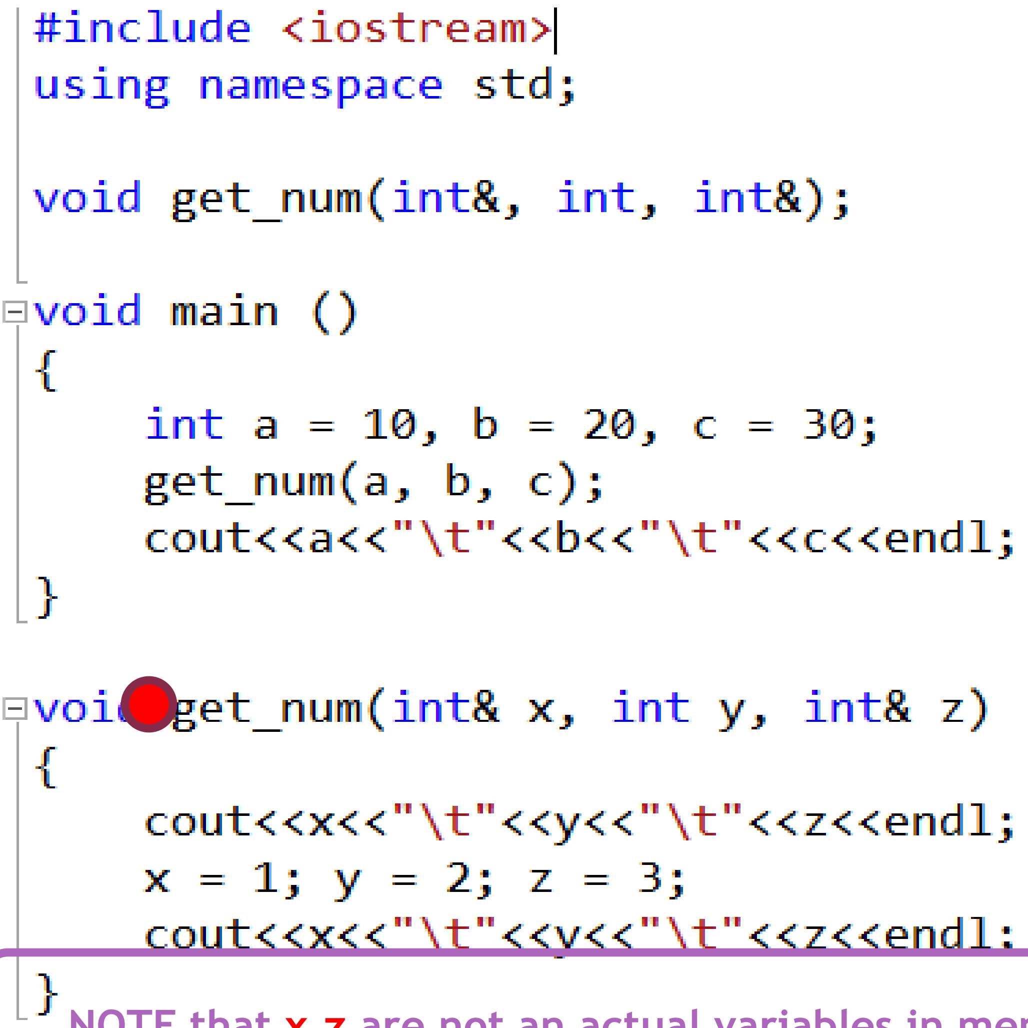


```
#include <iostream>
using namespace std;

void get_num(int&, int, int&);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int& x, int y, int& z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

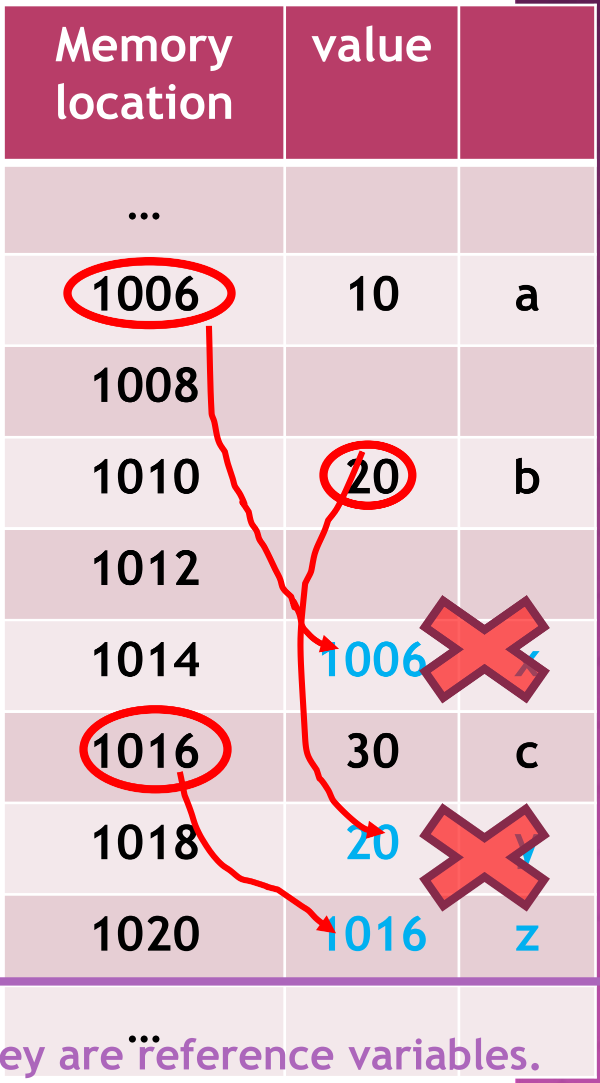
Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014		
1016	30	c
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value



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

void get_num(int&, int, int&);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int& x, int y, int& z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

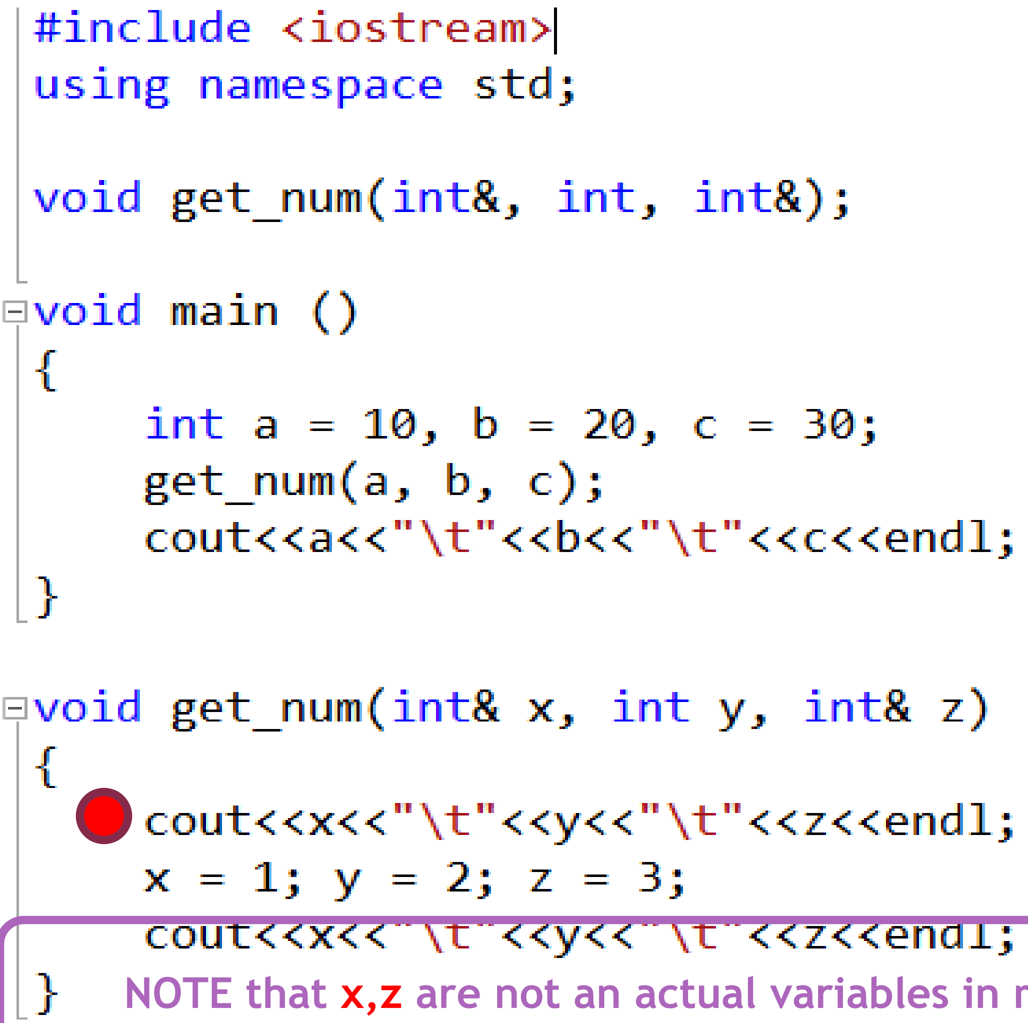


Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014	1006	x
1016	30	c
1018	20	y
1020	1016	z
…		

NOTE that **x,z** are not an actual variables in memory. They are reference variables.

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value



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

void get_num(int&, int, int&);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int& x, int y, int& z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

x →

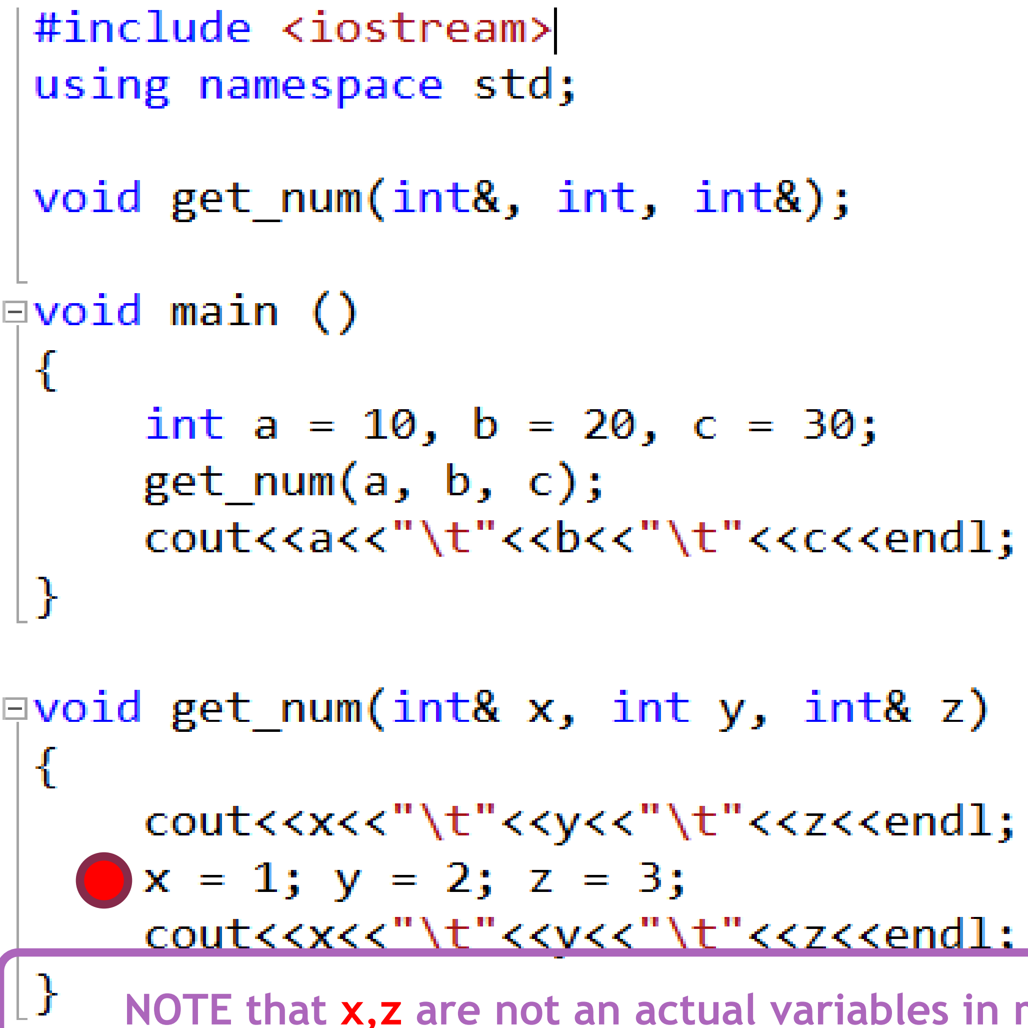
z →

Memory location	value	
…		
1006	10	a
1008		
1010	20	b
1012		
1014	~~1006~~	x
1016	30	c
1018	20	y
1020	~~1016~~	z
…		

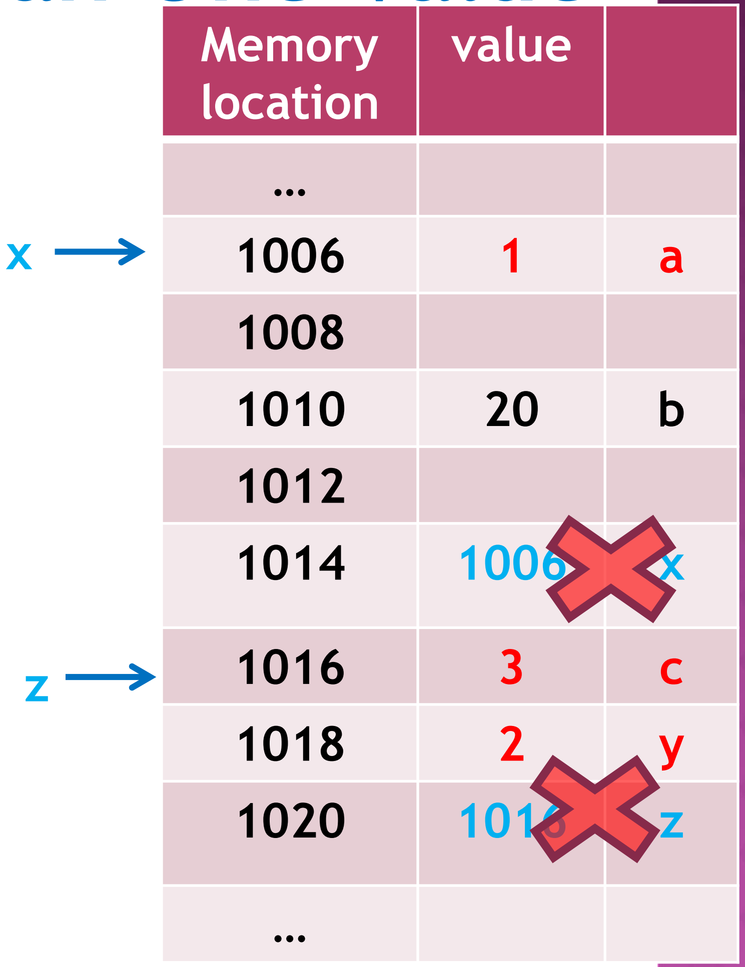
NOTE that x,z are not an actual variables in memory. They are reference variables.

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value



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

void get_num(int&, int, int&);

void main ()
{
     int a = 10, b = 20, c = 30;
     get_num(a, b, c);
     cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int& x, int y, int& z)
{
     cout<<x<<"\t"<<y<<"\t"<<z<<endl;
     x = 1; y = 2; z = 3;
     cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

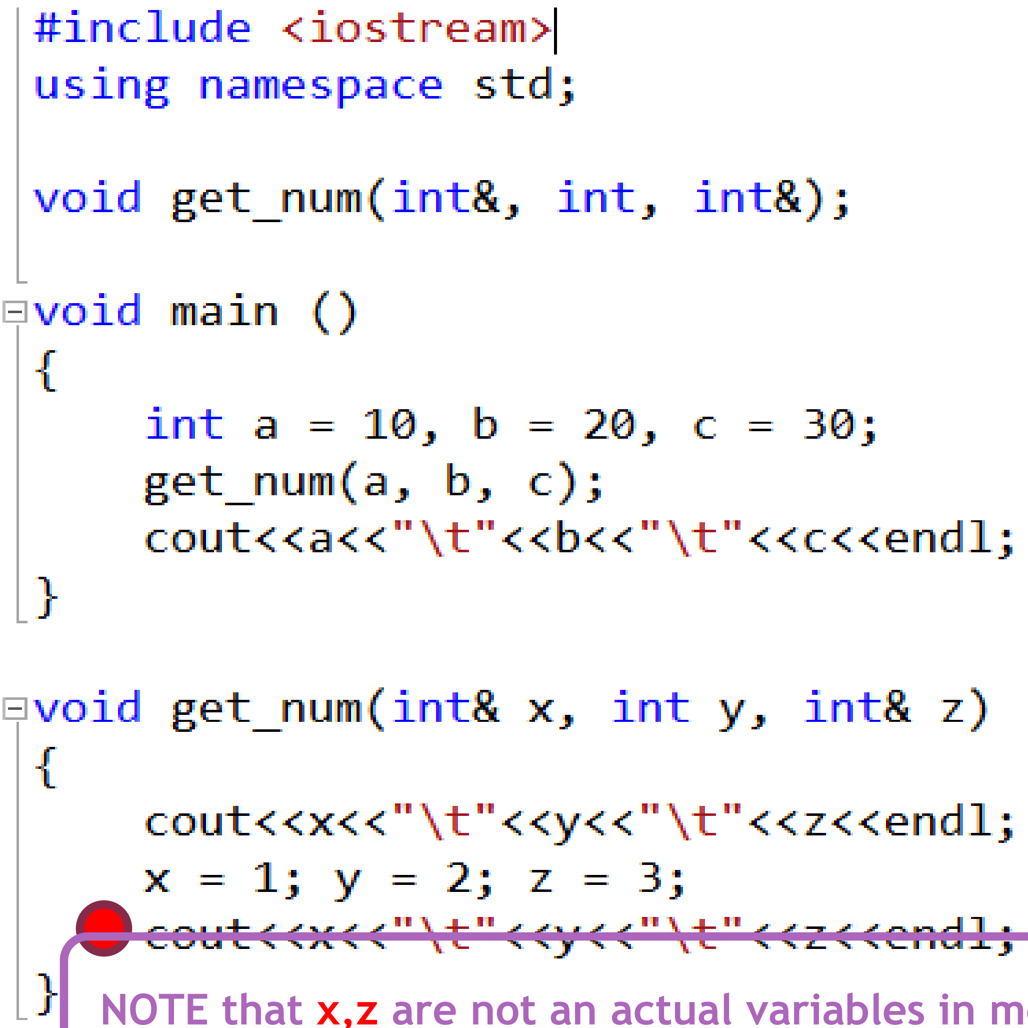


Memory location	value	
…		
1006	1	a
1008		
1010	20	b
1012		
1014	~~1006~~	x
1016	3	c
1018	2	y
1020	~~1016~~	z
…		

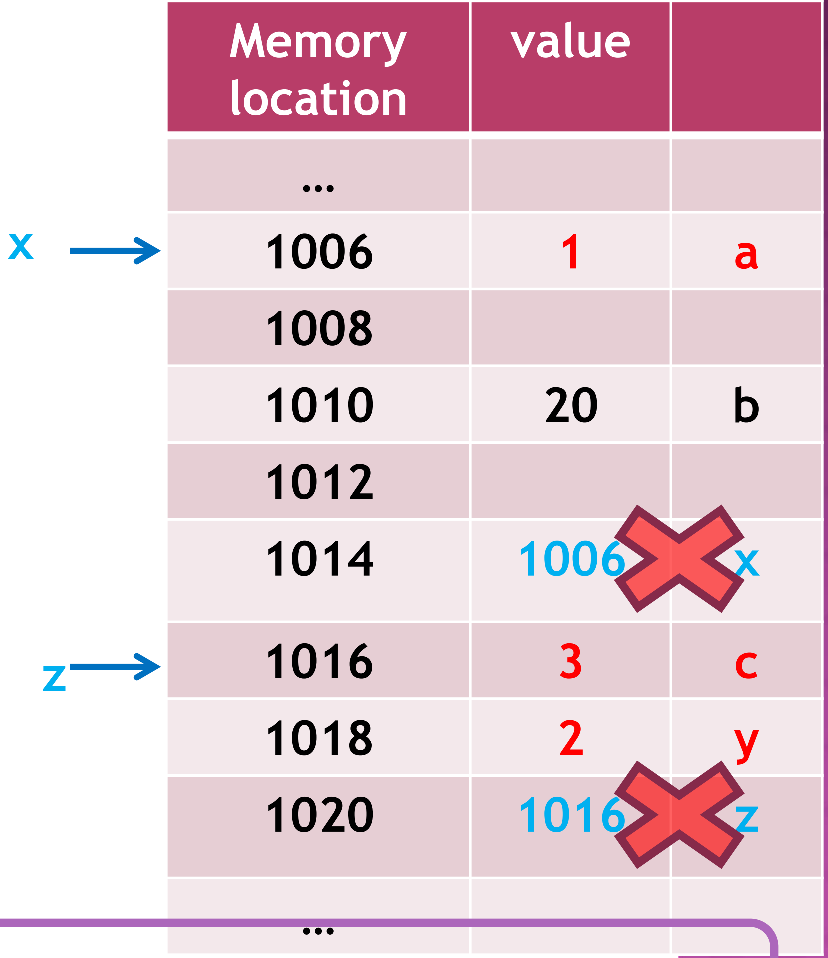
NOTE that x,z are not an actual variables in memory. They are reference variables.

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value



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

void get_num(int&, int, int&);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int& x, int y, int& z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

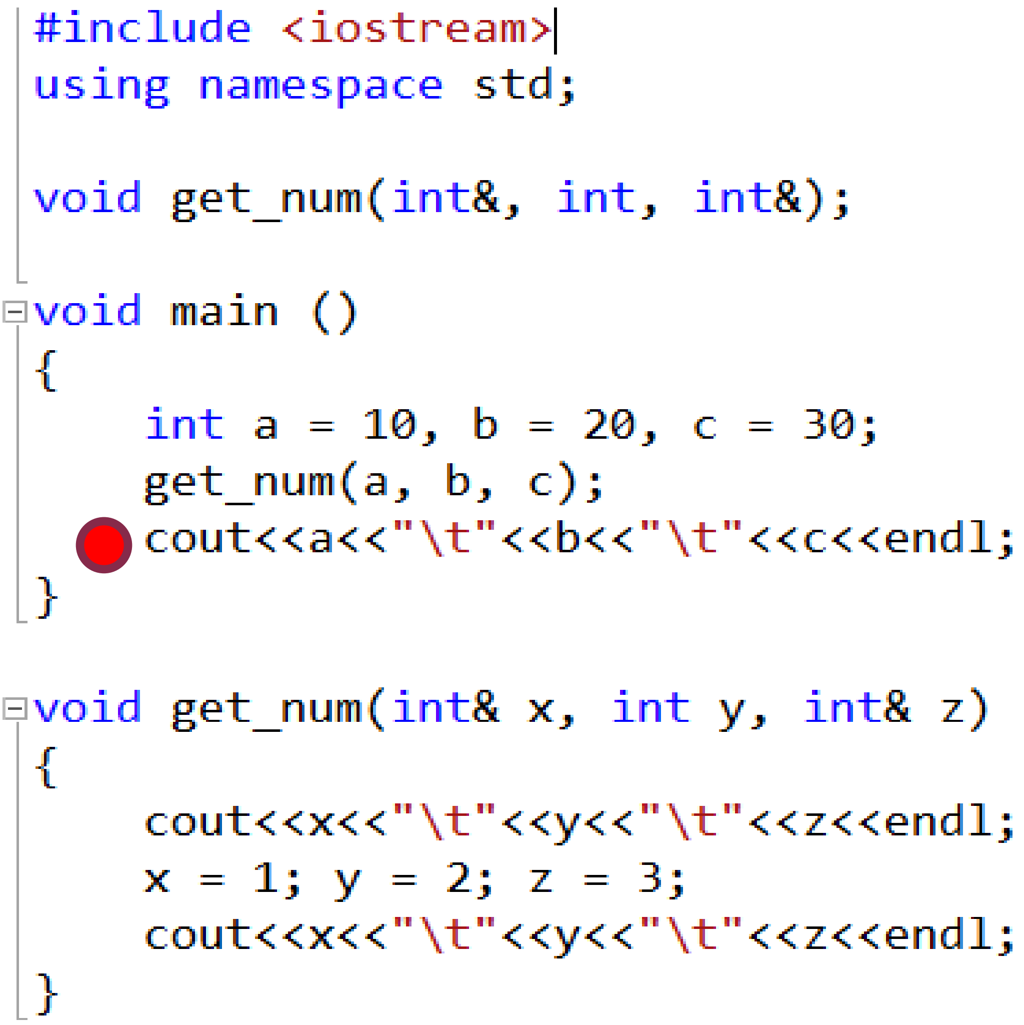


Memory location	value	
…		
1006	1	a
1008		
1010	20	b
1012		
1014	~~1006~~	~~x~~
1016	3	c
1018	2	y
1020	~~1016~~	~~z~~
…		

NOTE that x,z are not an actual variables in memory. They are reference variables.

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value



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

void get_num(int&, int, int&);

void main ()
{
    int a = 10, b = 20, c = 30;
    get_num(a, b, c);
    cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}

void get_num(int& x, int y, int& z)
{
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
    x = 1; y = 2; z = 3;
    cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
```

Memory location	value	
…		
1006	1	a
1008		
1010	20	b
1012		
1014		
1016	3	c
1018		
1020		
…		

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value

- In passing a reference to a variable you are not actually returning anything, you are just changing the values in that address (which are the actual arguments of the function).
- Any other function “looking” at that location will see the changed value (as if you returned it).
- Notice that you didn't need to dereference the address. It’s done for you in the called function.

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value

- Passing by reference is used to provide access to caller's actual argument.
- Caller's data can be modified by called function.
- Typically used for input functions.
- Specified by ampersand &, after the type in formal parameter list.
- Remember that passing an array variable is passing by reference without explicitly specifying.

1. Programmer-defined Functions – (cont.)

(11) Returning More than One Value

Example 3 (*BONUS) - Lec8Ex3.cpp

A function to return the sum, average, max, and min of three numbers.

Example 4 - Lec8Ex4.cpp

A function that swaps two numbers (mathematically☺).

Example 5

A function to input an array of struct variable.

Thank
you.