University of Duhok
College of Science
Department of Computer
Second Year

Data Structures
Function and Stack

Avin J. Kovli
2022 - 2023

OUT-LINES:

- Allocation of memory.
- Functions.
- Function Calls.
- Stack.

ALLOCATION OF MEMORY:

- **Static allocation** is the allocation of memory space at **compile time**.
- **Dynamic allocation** is the allocation of memory space at **run time** by using **operator new.**
- **Dynamically Allocated Data:**

```
int *pint=new int;
*pint=5;
cout<<*pint;
```

```
char* ptr;
ptr = new char;
*ptr = 'B';
cout << *ptr;
```

- To de-allocate the memory the key word delete is used as follow:

```
delete ptr;
```

DYNAMICALLY ALLOCATED DATA:

- You can dynamically allocate an entire array:

```
int *p = new int[5];
for(int cnt=0; cnt<5; cnt++){
*(p+cnt)=cnt;
cout<<*(p+cnt)<<'\t';}
```

- Free the array space using delete:

```
delete []p;
```

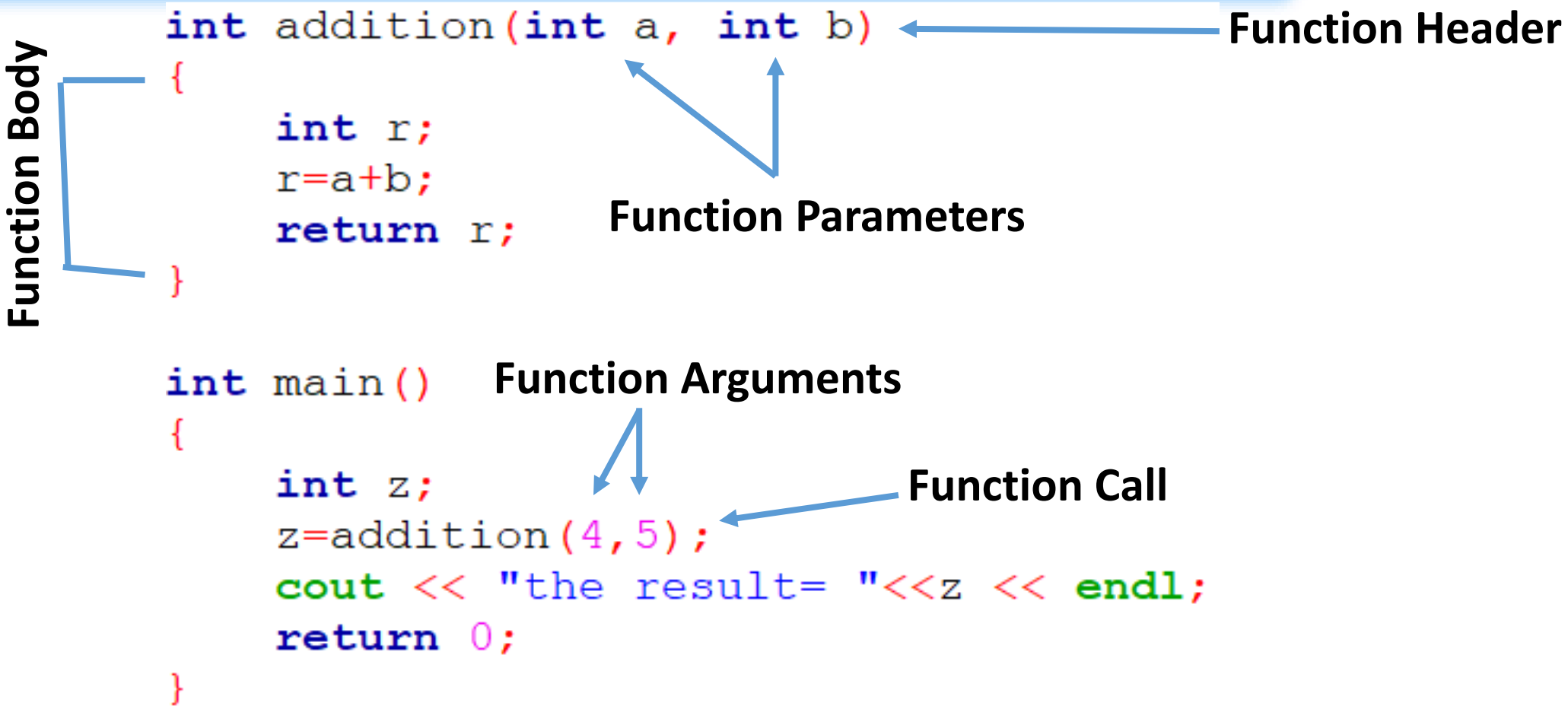
*p		*(p+2)		*(p+4)
a[0]	a[1]	a[2]	a[3]	a[4]
15	3	21	5	1

FUNCTIONS:

- A function is a sequence of C++ code executed from another part of the program by a function call. Every function's definition consists of two parts, the header and a compound statement.

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

EXAMPLE:

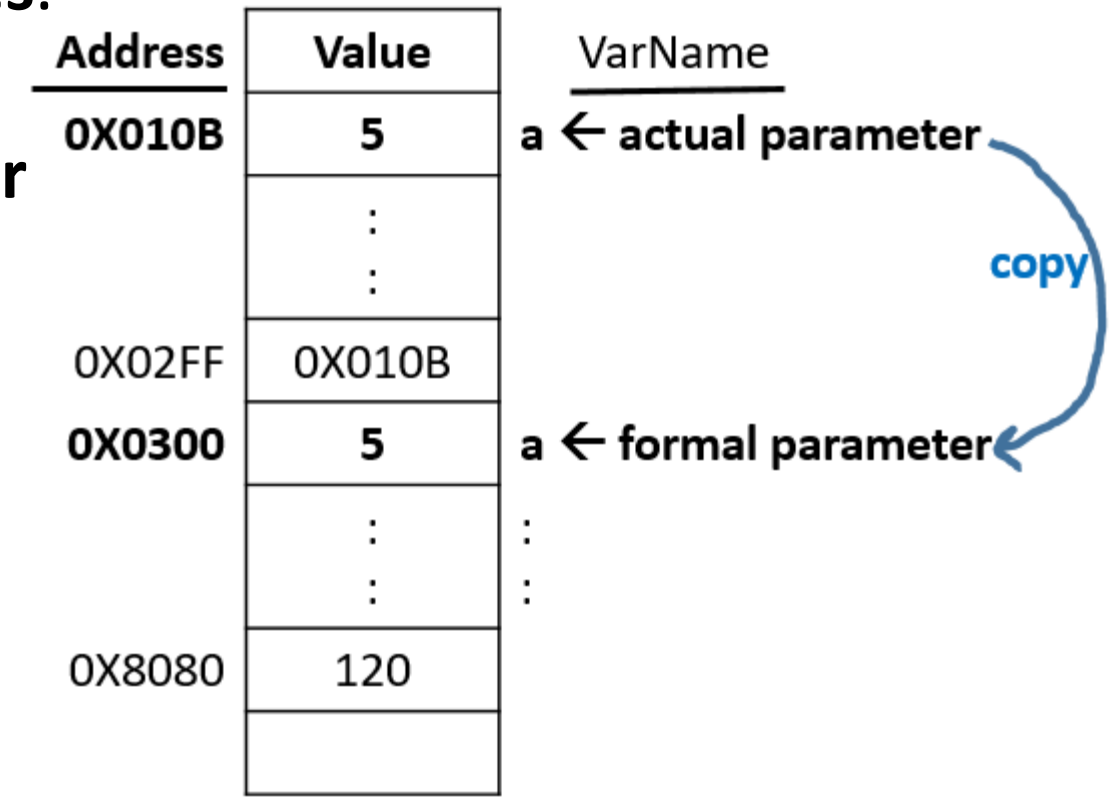

int addition(int a, int b)
{
int r;
r=a+b;
return r;
}
int main()
{
int z;
z=addition(4,5);
cout << "the result= "<<z << endl;
return 0;
}
Function Header
Function Body
Function Parameters
Function Arguments
Function Call

FUNCTION CALLS:

- There are two mechanisms used generally in C++ to call the functions.
  1. Call– by – value.
  2. Call– by – reference.
     - Pass – by – pointer.
     - Pass – by – reference.
- The **parameters passed** to function are called **actual parameters (Arguments)** whereas the **parameters received** by function are called **formal parameters**.

CALL– BY- VALUE:

- When you specify the **parameters** in the function definition as **ordinary variables**.
- This method **copies** the **actual value** of an **argument** into the **formal parameter** of the function, and the **two** types of **parameters** are **stored** in **different** memory **locations**.
- In this case, **changes** made to the **parameter inside** the function have **no effect** on the argument.



EXAMPLE:

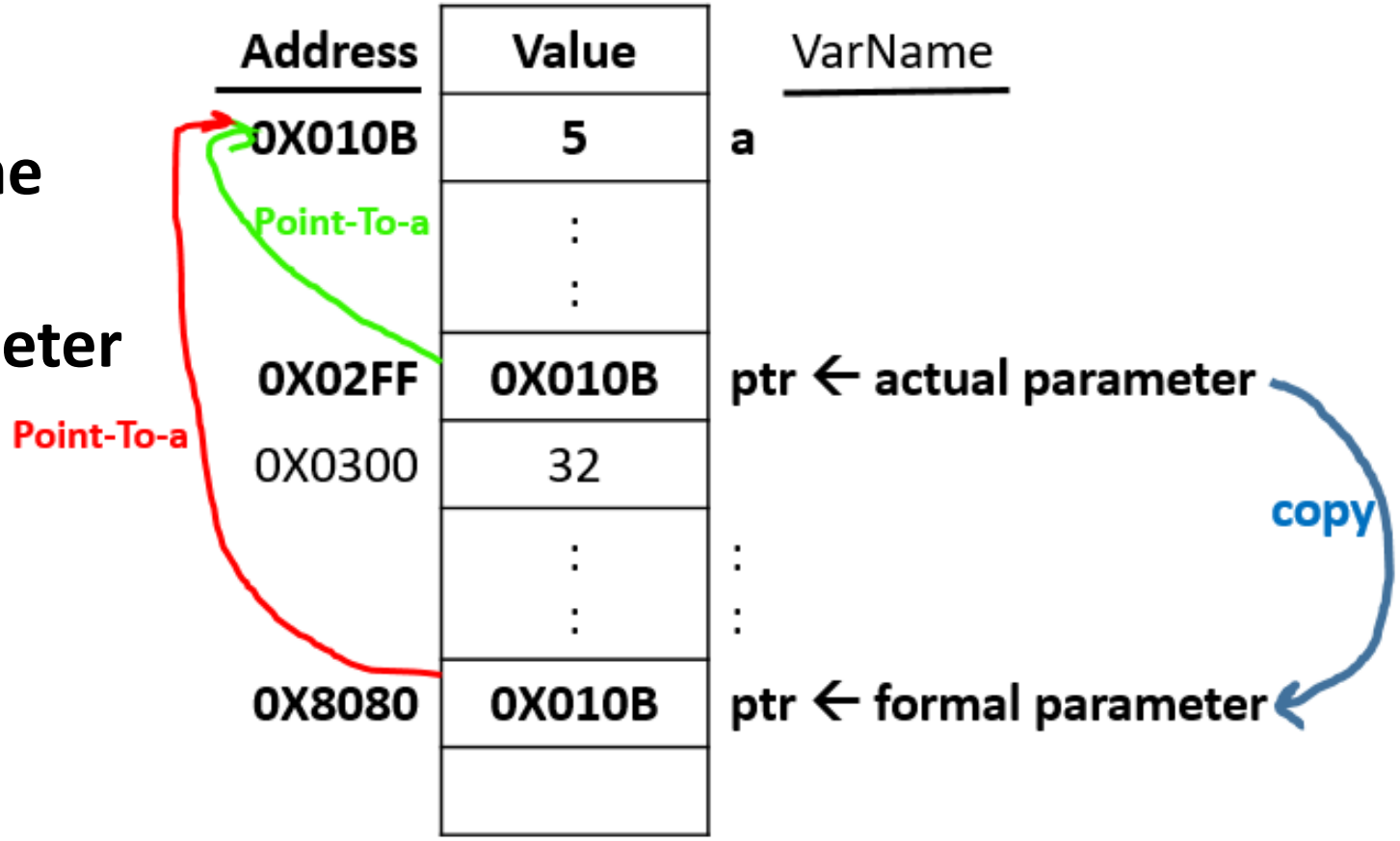
```
int addBy10(int a)
{
    a+=10;
    return a;
}
int main()
{
    int a=5;
    cout<<"addBy10(a)= "<< addBy10(a)<<endl;
    cout << "a= "<<a << endl;
    return 0;
}
```

output

```
addBy10(a)= 15
a= 5
```

CALL – BY – REFERENCE (PASS-BY-POINTER):

- This method **copies** the **address** of an **argument** into the **formal parameter**.
- Both the **actual and formal parameters** refer to the **same locations**, This means that **changes** made to the **parameter affect** the **argument**.



EXAMPLE (USING POINTER):

```
int addBy10(int *a)
{
    *a+=10;
    return *a;
}
int main()
{
    int a=5;
    int *ptr;
    ptr=&a;
    cout<<"addBy10(a)= "<< addBy10(ptr)<<endl;
    cout << "a= "<<a << endl;
    return 0;
}
```

```
cout<<"addBy10(a)= "<< addBy10(&a)<<endl;
```

```
addBy10(a)= 15
a= 15
```

CALL – BY – REFERENCE (PASS-BY-REFERENCE):

- It allows a function to **modify** a **variable without** having to create a **copy** of it.
- In the header of function, we have to **declare reference variables**.
- The **memory location** of the passed variable and parameter is the **same** and therefore, any **change** to the parameter **reflects** in the variable as well.

EXAMPLE (USING ADDRESS):

```
int addBy10(int &a)
{
    a+=10;
    return a;
}
int main()
{
    int a=5;
    cout<<"addBy10(a)= "<< addBy10(a)<<endl;
    cout << "a= "<<a << endl;
    return 0;
}
```

```
addBy10(a)= 15
a= 15
```

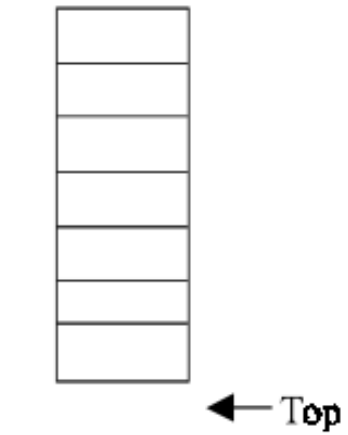
Stack

- A stack is a **non-primitive** linear data structure.
- It is an **ordered collection** of items into which new data items is **added** or **deleted** at only **one end**, called the **top** of the stack.
- The last added element will be first removed from the stack. That is why the stack is also called **Last-in-First-out (LIFO).**

Stack of Books

Stack of rings

Stack:



Stack is empty

Stack Operations

- **Push (newItem)** -- Adds newItem to the top of the stack.
- **Pop (item)** -- Removes the item at the top of the stack and returns it in item.
- **IsEmpty** -- Determines whether the stack is currently empty.
- **IsFull** -- Determines whether the stack is currently full.
- **MakeEmpty** -- Sets stack to an empty state.

Push algorithm (add New Item):

1. If TOP = SIZE – 1, then:
    (*a*) Display "The stack is in overflow condition"
    (*b*) Exit
2. TOP = TOP + 1
3. STACK [TOP] = ITEM
4. Exit

Pop algorithm (delete items):

1. If TOP < 0, then
    (*a*) Display "The Stack is empty"
    (*b*) Exit
2. Else remove the Top most element
3. DATA = STACK[TOP]
4. TOP = TOP – 1
5. Exit

Any questions?

THANK YOU