BITG 1223: Function (Part 2)
At the end of this lecture, you should be able to:

- explain parameter passing in programs using: **Pass by Value** and **Pass by Reference**.
- use **reference variables** as parameters.
Passing Variables to a Function

- A variable has both a value and a unique address (memory location)

- You can pass/send either the variable’s value or its address to the receiving function using:
  - Pass by value
  - Pass by reference
Sending Data into a Function

- Values are passed to a function during the function call: e.g.: `pow(a, b)` -> passes values of `a` and `b` to function `pow`.

- Values passed to function are arguments. An argument can also be called an actual parameter.

- Variables in a function header that hold the values passed as arguments are parameters. A parameter can also be called a formal parameter.

- An argument will be matched to a parameter.
Passing Multiple Arguments

When calling a function and passing multiple arguments:

- the number of arguments in the call must match the function prototype and definition.

- the first argument will be used to initialize the first parameter, the second argument to initialize the second parameter, etc.
Passing Variables *By Value*

- In a **pass by value**, the computer passes the content of each actual variable (argument) to the parameter of the receiving function (creates a **copy/duplicate** of the actual value).
- Receiving function is **not given access** to the actual variable in memory.
- It **cannot change value** stored inside the **actual variable**.
- By default, variables are passed **by value** in C++. 

**LECTURE 5**
Pass By Value Concept

Pass by value
(num1 is a copy of a, num2 is a copy of b)

Purpose of Function exchange:
To exchange the values of two numbers

The values of the two numbers have NOT been exchanged by function exchange

Output ???

The diagram illustrates the exchange of two numbers, a and b, using a function called `exchange`. The values of the two numbers are passed by value, meaning that copies of the original values are passed to the function, and the function works on these copies. The original values of a and b remain unchanged. The code snippet demonstrates this by assigning different values to `num1` and `num2` and then using the `exchange` function to swap them. The output shows the final values of `num1` and `num2` after the function call.
```cpp
// prototype declaration
#include <iostream>
void fun (int x);
int main ()
{
    int a = 5;
    cout << a << endl;
    fun(a);
    cout << a << endl;
    return 0;
} // main

void fun (int x)
{
    x = x + 3;
    return;
} // fun
```

**Pass by value**: Programming Example

**Output**: 5

**One-way communication**

(x is a duplicate of a)

**Only a copy**
Passing Variables *By Reference*

- Passing a variable’s address is referred to as **passing by reference**
- Receiving function has access to the passed variable (the actual variable)
- Use when you want the receiving function to change contents of actual variable
- To pass a variable *by reference* in C++:
  - Include an & (the *address-of operator*) before the name of the corresponding formal parameter in receiving function’s header
Using Reference Variables as Parameters

- A mechanism that allows a function to work with the **original** argument (value) from the function call, **not a copy** of the argument (value)
- Allows the function to **modify values** stored in the calling function (**modify original values**)
- Provides a way for the function to give **more than one value** as the RESULT of using the function.
Passing by Reference

- A **reference variable** is an **alias** (different name) for another variable.
- Its ‘formal parameter’ (known as **reference parameter**) declared with an **ampersand** (\&).
- Example:
  ```
  void getDimensions(int&, int&);
  ```
- Changes to a reference variable value are actually changing the original value for which it refers to.
- Use **reference variables** to implement passing parameters *by reference*
Pass By Reference

Concept

The values of the two numbers have been exchanged by function `exchange`.

Output ???

```c
void exchange ( int& num1, int& num2 )
{
    // Local Declarations
    int hold;

    // Statements
    hold = num1;
    num1 = num2;
    num2 = hold;

    return;
} // exchange
```

Variables are called `a` and `b` in main ...

Pass by Reference (num1 is an alias for a, num2 is an alias for b)

... num1 and num2 in exchange

Purpose of Function exchange:
To exchange the values of two numbers

Local Variable

LECTURE 5
Pass By Reference: Programming Example

//prototype declaration
#include <iostream>
void fun (int& x);
int main ()
{
    int a = 5;
    cout << a << endl;
    fun(a);
    cout << a << endl;
    return 0;
} // main

void fun (int& x)
{
    x = x + 3;
    return;
} // fun

Output:
5
8

Pass by Reference (x is an alias for a)
Reference Variable: TAKE NOTE

- Each **reference parameter** must contain `&`
- Space between type and `&` is unimportant
- Must use `&` in both function prototype and header
- **Argument** passed to reference parameter **must be a variable** – cannot be an expression or constant
- **Use when appropriate** – don’t use when argument should not be changed by function, or if function needs to return only 1 value
Parameter passing: TAKE NOTE
(pass by value vs. pass by reference)

- **Pass by value**
  - a copy of data is created and placed in a local variable in the called function
  - ensure that regardless of how the data is manipulated and changed in the called function, the original data in the calling function are safe and unchanged

- **Pass by reference**
  - sends the address of a variable to the called function
  - use the address operator (&) in the parameter of the called function
  - anytime we refer to the parameter, therefore we actually referring to the original variable
  - if the data is manipulated and changed in the called function, the original data in the calling function is changed