5.1 Void Functions

- **function that do not return a value.**
- **examples:**

```
void f1( int x)
{
}
void f2( )
{
}
void f3 ( )
{
   return;
}
```

5.2. CALL-BY-REFERENCE

During a function call memory is allocated for the formal parameters and local variables. Memory is also allocated for the address of the instruction to return to after the function has executed. There are two ways arguments are passed to a function during a function call, call-by-value and **call-by-reference.** Being able to call functions has many advantages which include: 1) allows us to write complex programs; 2) allows the user to decompose a program into smaller manageable components; 3) makes it easier to find errors; 4) makes it easier to maintain the program.

Definitions and Important Terms (You Need To Know To Understand Call By Reference and Call By Value)

We will define several terms that you need to know to understand function calls. They are as follows:

a. **Actual arguments** are the arguments used in the function call statement.

b. **Formal parameters** are the parameters used in the function header.

c. During a **call-by-value** a copy of the actual argument is made and placed into its corresponding formal parameter.

d. During a **call-by-reference** the address of the actual argument is copied into the formal parameter, when pointers are used.

e. During a **call-by-reference** the formal parameter is an alias for the actual argument when a reference is used in C++.

f. A formal parameter is called an **output parameter** when it is passed-by-reference. A formal parameter is called an **input parameter** if it is passed-by-value

g. If the value of a formal parameter is changed inside a function, and if this changed value is needed outside the function, it should be labeled as an output parameter or as a parameter that is passed by reference.

h. If the value of a formal parameter is only used inside a function, it should be called an input parameter or a parameter that is passed by value.

i. A **prototype** is a function declaration.

j. **References** are implemented internally as pointers

Declaration Syntax

```
return_type  function_name(formal_parameter_list);
```

Example:

```
void input(int &a, double c, char * m);
```

Explanation: a and m are passed-by-reference, and c is passed-by-value.

SEE PAGES 259-273
MEMORY MAPS ALWAYS HERE

GO OVER THE FOLLOWING SWAP CODE:

```
void swap(int a, int b)  //CALL-BY-VALUE
{
   int temp = a;
        a = b;
        b = temp;
}

void swap(int *a, int *b) //C-STYLE CALL-BY-REFERENCE
{
   int temp =*a;
        *a = *b;
        *b = temp;
}

void swap(int &a, int &b) //C++ CALL-BY-REFERENCE
{
   int temp = a;
        a = b;
        b = temp;
}
```

- **Precondition – condition that is true before the function executes**

  Example for Input function in Assignment#2:

  The variables cell_number, relays, and call_length have not been initialized

- **Postcondition – condition that is true after the function has executed**

  Example for Process function in Assignment#2:

  The variables net_cost, call_tax, and call_total_cost have been initialized using the determine tax rate.