**CPP NOTES – DAY 08**

**Functions**

Can be classified into two categories:

* Library functions – predefined by compiler itself. eg: printf()
* User-defined functions – defined by the user according to requirement.

It is possible to code any prgms utilizing only main functions, it leads to a number of problems like complexity which results in hard debugging, testing and maintaining the code. Each functionality can be splitted into different functions which creates a modular code by improving the efficiency and reducing the complexity. Those set of codes are called Functions. Also known as self-contained block of code.

*return\_type name() {*

*// Function body*

*}*

3 Parts in functions:

1. Declaration of functions / prototyping of functions //int a=10;

Eg*: int adIntegers(int,int);* , *float divide(int,int);*

1. Definition of functions: where body of the function is defined.

Eg: *int addIntegers(int v1, int v2){*

*int retValue = 0;*

*retValue = v1+v2;*

*return retValue;*

*}*

1. Calling of functions: where functions are called for execution.

Eg: addIntegers(3,4);

Functions are classified into:

1. Function with no input args and no return type – void display(void);
2. Function with input args but no return type – void display(int);
3. Funciton with both input args and return type – int display(int);

**The Process of a Function Call (Step-by-Step)**

**a. Function Call**

When a function is called, the following sequence of actions occurs:

1. **Return Address:** The address of the next instruction after the function call (i.e., the instruction to return to) is pushed onto the stack.
2. **Function Parameters:** The arguments passed to the function are pushed onto the stack (if applicable).
3. **Stack Frame Allocation:** A new frame is created for the function, and space is reserved for its local variables and saved registers.
4. **Frame Pointer Update:** The frame pointer is updated to point to the current function's stack frame.

**b. Function Execution**

Now the function executes, using the local variables, performing calculations, etc.

**c. Function Return**

When the function finishes:

1. The return value (if any) is placed into the return location (usually a specific register or memory location).
2. The stack frame is "cleaned up":
   * The return address is popped from the stack.
   * The frame pointer is restored.
   * The stack pointer is updated, effectively removing the current function's stack frame.
3. Control is transferred back to the return address, and execution continues from where the function was called.

**Execution Flow and Return Address:**

1. **Initial Call to main()**:
   * The main() function is called, and its stack frame is pushed onto the stack.
   * The program starts executing inside main(), and it reaches the call to functionB().
2. **Calling functionB()**:
   * Before functionB() executes, the **return address** (the address of the next instruction to execute in main()) is stored in the stack. This is the memory address where the program should return after functionB() finishes.
   * A new stack frame is created for functionB(), and the program starts executing inside functionB().
3. **Calling functionA() from functionB()**:
   * Before functionA() executes, the **return address** (the address of the next instruction in functionB() after functionA() is called) is pushed onto the stack.
   * A new stack frame is created for functionA(), and the program starts executing inside functionA().
4. **Returning from functionA()**:
   * When functionA() finishes execution, the **return address** for functionB() is popped off the stack, and control is transferred back to the location in functionB() where functionA() was called.
   * The program resumes executing the rest of functionB().
5. **Returning from functionB()**:
   * When functionB() finishes execution, the **return address** for main() is popped off the stack, and control is transferred back to main() where functionB() was called.
   * The program resumes executing the rest of main().

*Fyi:*

* *in C return is not compulsory, but in cpp return is must.*
* *can’t declare variables inside switch case.*
* *Memory leak: Allocated but not used*
* *When function is declared and not used, it will remain as a text segment in the memory.*
* *Dynamic allocation aka Heap(Explicit) consists of keywords such as new, malloc, realloc etc…*
* *Static allocation aka Stack(Implicit) is activated whenever we do declaration with int a=1,float b etc…*
* *Stack Frame is allocated whenever is function is called out to execute.*
* *Goto can’t be executed if the label is declared outside the function(outer function) because of the unavailability of stack frame.*

*Book to refer – Dennis Ritchie-C programming*