C++ Notes Day-1&2 Date: 05-06 June 2025

Setting-Up Environment C/C++

- Toolchain: gcc
  - Install MingW Gcc/G++
  - Setup path of bin folder to env
- IDE: Eclipse
  - Install CDT extention from eclipse marketplace
  - Change perspective to C/C++

Introduction to C/C++

- C History
  - Founder: Denis Richhe
  - Bell Lab
  - Year:1972
  - PDP 11
  - C is statically type checked and strongly type checked general purpose programming language.
  - ANSI is looking after the standered of C/C++.

```
method1{
int Num1=100; //OK
//Num1=100;       //NOT OK
Num1="Makeet"; //NOT OK
}
method2{
int Num1=100;
}
method3{
int Num1=100;
}
```

- standards of C
  - C89
  - C95
  - C99
  - C11
  - C17
  - C23

Data Types and Working of C/C++

- Tokens: Every single entity of the program is known as Token.
- Keywords: The language reserved words are known as Keywords.

- Identifiers: Words or the names to the variables, methods, classes given by programmer itself are known Identifiers.
- Literals: The constant values used in the expressions or assigments is known as literals.
- Data Types in C: It helps the programmer to keep the specific type of the data inside the program.
    - Basic Data Types
        - int
        - char
        - float
        - double
        - void
        - bool is also a datatype since 2023
    - Derived Data Types: It is derived from basic data types.
        - Arrays
        - Pointers
        - Functions
    - User Defined Data Types
        - structure

```
struct ABC
{
    int Num1;
    char Name[50];
}
```

        - union

```
union ABC
{
    int Num1;
    char Name[50];
}
```

- Type Modifiers
    - short
    - long
    - signed
    - unsigned
- Type Specifiers/Qualifiers
    - const
    - volatile
- Variable: It is a container of a specific data type which hold single entity of the that data type.

```
int Num1=450;  //Here Num1 is a variable of type int having value 450 or using 4 byte memory.
```

- A variable is specifying
    - Type
    - Size
    - State
- Entry point function
- Syntax of main
- Example-1:

```
int main(int args, char *argv[], char *envp[])
{

}
```

```
void main(int args, char *argv[], char *envp[])
{

}
```

```
void main(int args, char *argv[])
{

}
```

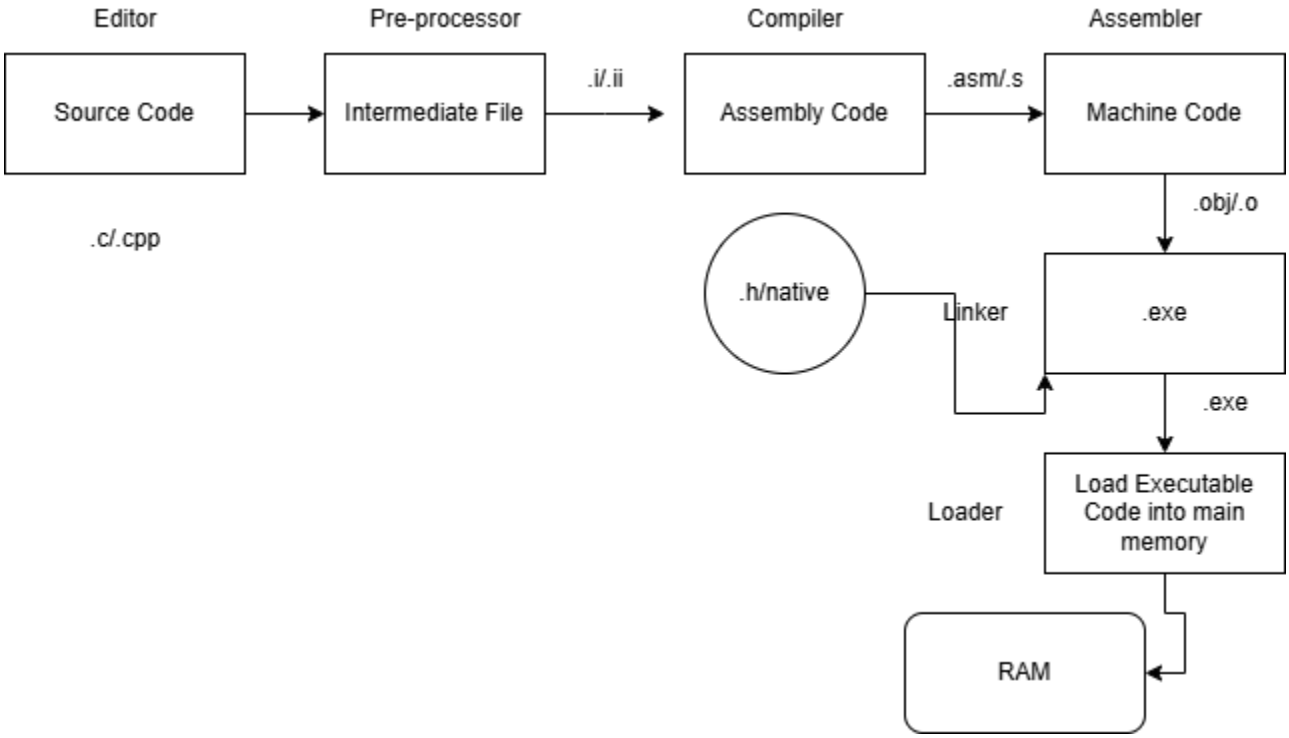
```
int main(int args, char *argv[])
{

}
```

```
void main()
{

}
```

```
int main()
{
    return 0;
}
```

- Reference: https://en.cppreference.com/w/c
- Software Development Kit: Development Tools + Documentation + Runtime Environment + Supporting Libraries
- Development Tools
    - Editor: It used to create or edit the program file.
    - Example:
        - Windows: Notepad, Notepad++, Visual Code, Wordpad etc.
        - Linux: Nano, vi, Vim, VS Code, TextEditor etc.
        - Mac OS: Vim, Vi, ExCode etc.
    - Pre-Processor
        - Primary role of pre processor is to remove comments from the source file.
        - To extent macros.
        - Example:
            - CPP(C/C++ Pre-Processor)
        - It generates intermediate file. (.i/.ii)
    - Compiler
        - It is a system software which primary role is to:
            - To check the syntax of intermediate source code.
            - It converts HLL code into LLL code. (Assembly Code).
        - Example:
            - tcc.exe
            - MS Vidual Studio: cl.exe
            - Linux: gcc
        - Compiler generate .asm/.s file.
    - Assempler
        - It is a system program which convert Assembly Level Language code into Machine level language code.
        - Example:
            - Turbo: Tasm
            - MS Visual Studio: Masam.
            - Linux: as
        - It generates .obj/.o file.
    - Linker
        - It is a system software whose primary role is to link machine code with system libraries.
        - It generates .exe file.
        - Example:
            - TurboL Tlink
            - MS Visual Studio: link.exe
            - Linux: ld
    - Loader
        - It is an OS API.
        - It is used to load executable file from HDD to Main Memory(RAM).
    - Debugger
        - Logical error is known as bug.
        - To identify such error we need to debug our code with the help of Debugger
        - Example: gdb, ddd

- Documentation
  - The role of documentation of the any technology or language or SDK is to provide full fledge and authentic informatiob about the language or technology or SDK to the end users/firms and organizations.
  - It can be provided in the form of pdf/html/txt format.
- Runtime Environment
  - It is responsible for the entire execution of the program.
  - Example:
    - Java runtime is JRE
    - C# runtime is CLR
    - C/C++ is C Runtime
- C Flow of execution



- Reference : https://www.tenouk.com/ModuleW.html
- Comments in C
  - Comments are used to add authentic and full fledge information about the source code.
  - Hence commets are used to design the official documentation of the source code.
  - Type:
    - Single Line comments

```
// Use Double // to add single line comments
```

    - Block comments

```
/*
This is My Program
I am .Net Developer
*/
```

        - Document Comments

```
/**
Auther Name

*/
```

- Working with Functions
    - Declaration and Definition of Functions
        - With no return type and with no parameters
        - With no return type and with parameters
        - With return type and with no parameters
        - With return type and with parameters
    - Stages to design the method
        - Method Declaration
        - Method Definition
        - Method Call
- Initialization and Assignment
    - Todo
        - Do it for variables of type int, char, float etc.
        - Work with variables of every datatypes with their pointers
        - Use the concept of derefferencing

Some basic concepts of C/C++

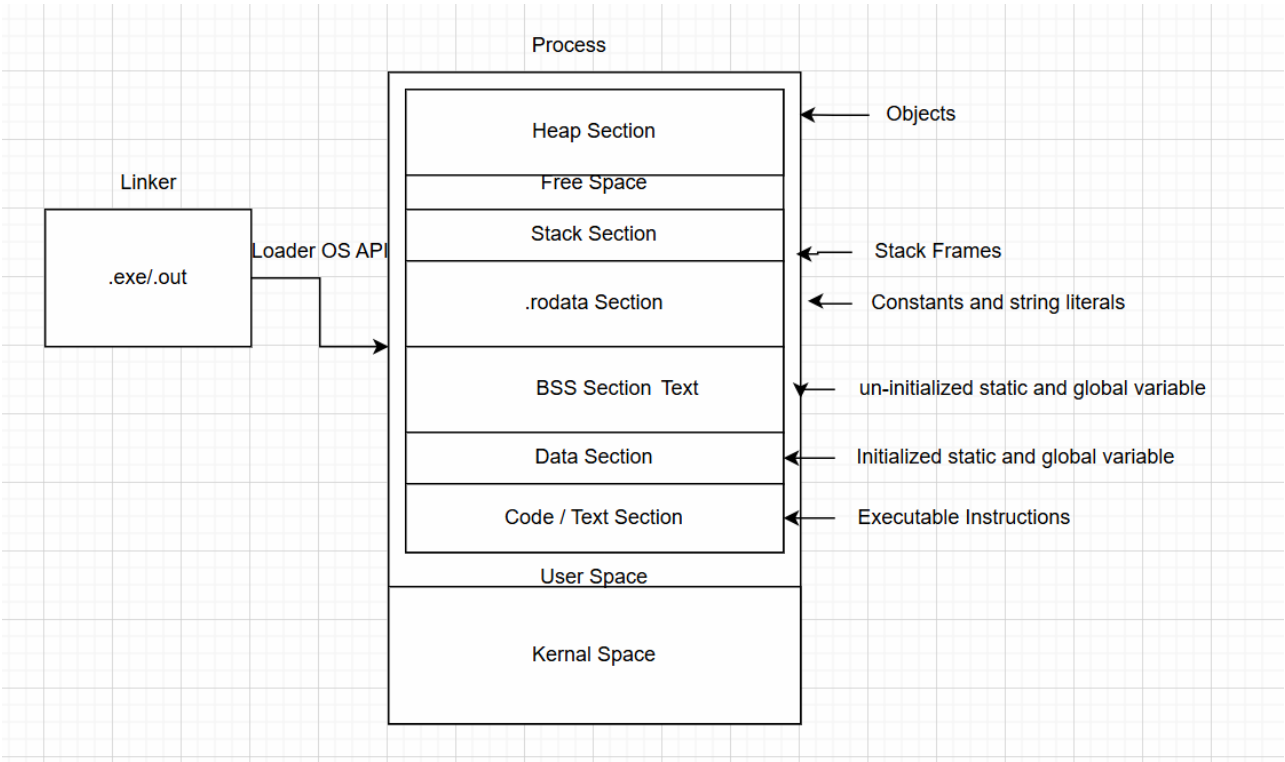
- use of 'typedef'
- use of 'sizeof' operator and size_t typedef
- use of '&' operator to get address of the variable
- Function arguments vs Function parameters

```
#include<cstdio>  //C style Header File
using namespace std;
void Add(int, int); //Function Prototype
int main()
{
    int Num1=100;
    printf("Value of Num1:%d\n",Num1);
    printf("Address of Num1:%p\n",&Num1);
    Add(100,200);   //Here 100 and 200 are arguments
    return 0;
}
void Add(int x, int y)  //Here X and Y are Parameters
{
    int res=x+y;
    printf("The Sum of X and Y:%d\n",res);
```

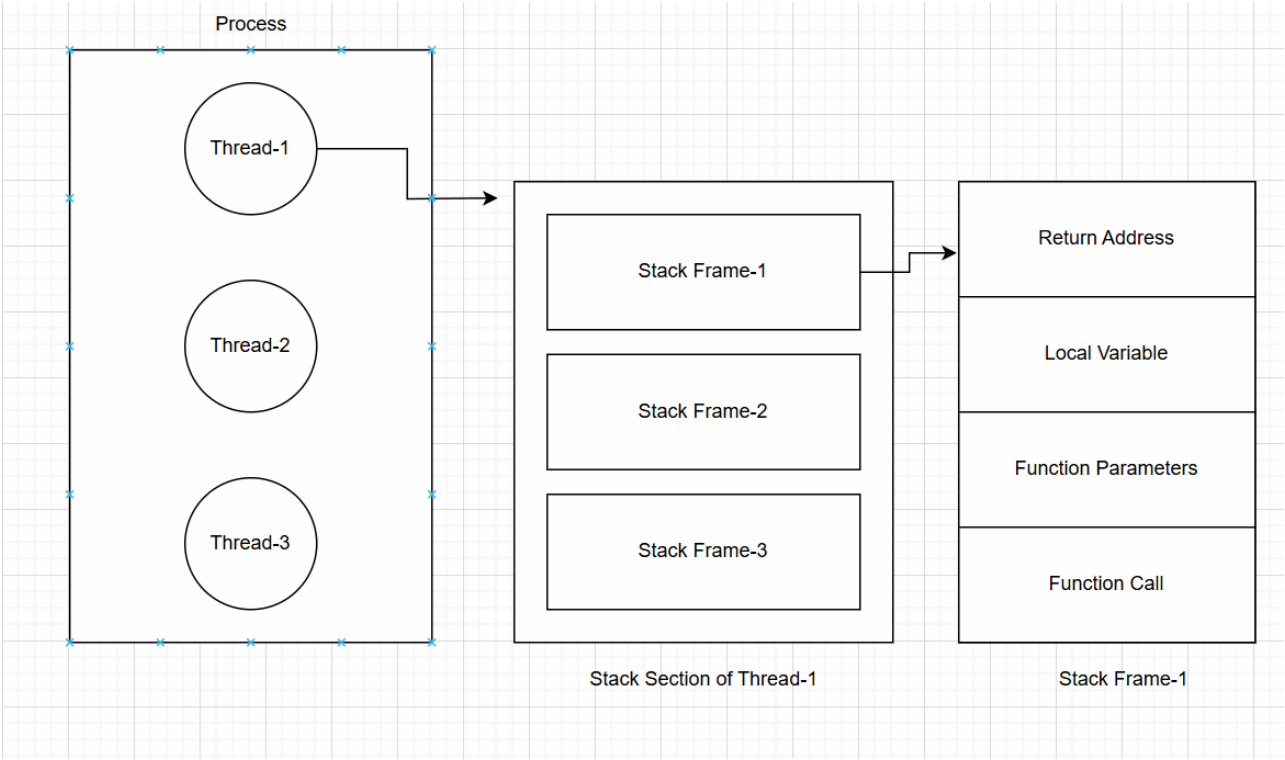
- Declaration and Definition + Initialization and Assignment of variables

```cpp
#include<cstdio>  //C style Header File
using namespace std;
int main()
{
    int Num1;   //Declaration + Definition
    printf("Value of Num1:%d\n",Num1);
    printf("Size of Num1:%d\n",sizeof(Num1));
    printf("Address of Num1:%p\n",&Num1);

    int Num2=90;    //Declaration + Definition + Initialization
    Num2=100;       //We can do assignment multiple time
    Num2=89;

    return 0;
}
```

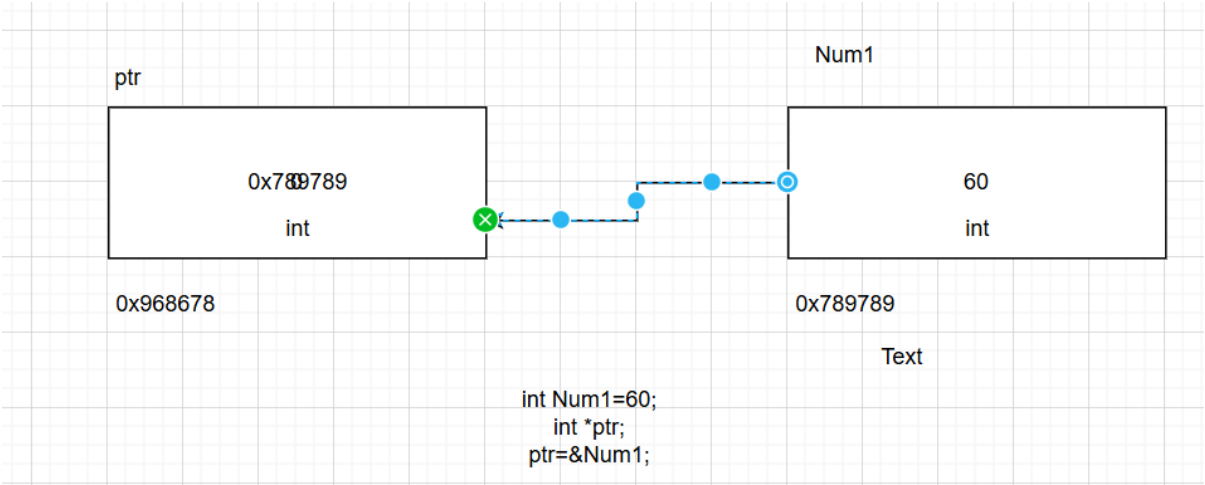
- Memory structure of C/C++ program



- Function Activation Record (Stack Frames)



- Pointer
  - Concept: Pointer is a variable which is used to store the address of other variable. With the help of pointer variable we can not only save address of the other variable, we can also access and manupulate value saved at given memory address



  - Declaration
    - Syntax: datatype *variableName.
    - Example: int *ptr, here ptr is pointer variable of type int
    - Example-1:

```
int* ptr;    //OK
```

    - Example-2:

```
int * ptr;    //OK
```

- Example-3:

```
int *ptr;   //OK
```

- Pointer Example: Initialization and Assignment

```
int main1()
{
    //Num1 is a variable of type int
    int Num1=60;  //Every defined variable has a memory address
    printf("Value of Num1:%d\n",Num1);
    printf("Address of Num1:%p\n",&Num1);
    printf("Size of Num1:%d\n",sizeof(Num1));

    int *ptr;   //ptr is pointer variable of type int

    ptr=&Num1;  //Assignment of the address of Num1 to the pointer variable ptr

    printf("Value of ptr:%p\n",ptr);
    printf("Address of ptr:%p\n",&ptr);
    printf("Size of ptr:%d\n",sizeof(ptr));

    printf("Value of Num1 using ptr variable:%d\n",*ptr);  //Dereferencing

    *ptr=100; //Dereferencing
    printf("Value of Num1:%d\n",Num1);
    return 0;
}
```

- Size of the Pointer - 16-bit: 2 Byte - 32-bit: 4 Byte - 64-bit: 8 Byte
- Wild pointer: Un-Initialized pointer variable is known as Wild Pointer

```
int main()
{

    int *ptr;   //Un-inialized pointer variable is known as wild-pointer,
becouse it will pointing to a any memory address

    printf("Value of ptr:%p\n",ptr);        //Value of ptr is always a
address of the memory
    printf("Address of ptr:%p\n",&ptr);

    //printf("Value by using ptr:%d\n",*ptr);

    return 0;
}
```

- NULL and Null Pointer - Null is a macro whose value is 0. - If we assign NULL to a pointer while declaring it, then the pointer is know as Null-Pointer
- Dereferencing in using pointer variable - If we assign a memory address to a pointer variable and we are accessing the value of the given memory address by using pointer variable name prefixing it with * this is know as Dereferencing

```
int main()
{
    int Num1=90;
    int *ptrNum1;

    int* ptr;   //OK
    int * ptr1; //OK
    //int ptr*;  //Not OK

    ptrNum1=&Num1;

    printf("%d\n",Num1);    //90
    printf("%p\n",&Num1);   //Address of Num1
    printf("%p\n",ptrNum1);     //Value of ptrNum1 which is actually address
of Num1
    printf("%p\n",&ptrNum1);    //Address of ptrNum1 variable
    printf("%d\n",*ptrNum1);    //90, value at the memory whose address is
stored inside ptrNum1 (Dereferencing)
    return 0;
}
```

- const qualifier
  - const is keyword in C/C++ and its is known as const/type qualifier.
  - If we do not want to change xthe value of the particuler varibale then we should use const qualifier.

    ```
    int X=90;   //X is avraible whose value can be changed at any point of
    time
    print(X);
    X=890;  //OK
    print(X);

    const int Y=100;
    print(Y);
    Y=560; //NOT OK, as Y is decalared using const qualifier so its value
    can not be changed/modified
    ```

  - variable decalared using const keyword its also known as read-only variable
  - constant variable should be assigned value at the time of declaration of the variable
- Constant and Pointer Combination
  - int *ptrNum

- ptrNum is non-constant local pointer variable who can store the address of non-contant integer variable
- Non-constant pointer variable can not store the address of contant variable.

```
int main1()
{
    int Num1=100;       //Num1 is non-constant local variable
    int Num2=200;
    int *ptrNum;    //ptrNum1 is non-constant local pointer variable
which is ready to store address of non-constant integer variable

    ptrNum=&Num1;
    printf("Value of ptrNum:%p\n",ptrNum);

    ptrNum=&Num2;
    printf("Value of ptrNum:%p\n",ptrNum);

    return 0;
}
```

- const int *ptrNum / int const *ptrNum / const int const *ptrNum

```
int main()
{
    const int Num1=100;     //Num1 is non-constant local variable
    const int Num2=200;
    const int *ptrNum;  //ptrNum1 is non-constant local pointer
variable which is ready to store address of constant integer variable

    int const *ptrNum1;
    //const int const *ptrNum2;   //Not Valid

    ptrNum=&Num1;
    printf("Value of ptrNum:%p\n",ptrNum);

    ptrNum=&Num2;
    printf("Value of ptrNum:%p\n",ptrNum);

    return 0;
}
```

Will be discussed tomorrow (07-06-2025)

- Const qualifier
- Constant and Pointer Combination.
- Structure in C
  - Structure in C++
  - Access Specifiers

    - Class, object and related c concepts
    - Multiple demo
    - Header Guard
    - #include<abc.h> versus #include"abc.h"
    - Namespace

Stream concept

- Standard stream objects associated with console.
- std namespace
- cin, cout, cerr and clog objects

Object Oriented Concepts

- class and object concepts
- characteristics of object.