Day 1

Classfication of languages:

1. Machine level languages
    - Binary language(1, 0)
2. Low level languages
    - Assembly
3. High level languages
    - C, C++, java

Chracteristics of Language

1. It has own syntax
2. It has its own rule(semantics)
3. It contain tokens:
    1. Identifier
    2. Keyword
    3. Constant/literal
    4. Operator
    5. Seperator / punctuators
4. It contains built in features.
5. We use language to develop application(CUI, GUI, Library)
6. If we want to implement business logic then we should use language.

Classfication of high languages:

1. Procedure Oriented Programming Languages
    - PASCAL, FORTRAN, COBOL, C, ALGOL, BASIC etc
    - FOTRAN is first high level pop language.
2. Object Orineted Programming Languages
    - Simula, Smalltalk, C++, Java, Python, C# etc.
    - Simula is first object oriented programming language. It is developed in 1960 by Alan kay.
    - Smalltalk is first pure object oriented programming language which is developed in 1967.
    - More 2000 languages are object oriented.
3. Object based programming languages
    - Ada, Modula-2, Java Script, Visual Basic etc.
    - Ada is first object based programming language.
4. Rule based programming languages
    - LISP, Prolog etc
5. Logic Orineted programming languages
6. Constraint oriented programming languages
7. Functional programming languages
    - Java, Python etc.

C Language Revision

History

- Inventor of C language is Dennis Ritchie
- It is developed in 1969-1972
- It is develped at AT&T Bell Lab USA
- It is developed on DEC-PDP11(Hardware)
- It is developed on Unix(Operating System)

ANSI Standards

- Set of rules is called standard and standard is also called as specification.
- American National Standard Institute(ANSI) is an organization which is responsible for standardization of C/C++ and SQL.
- ANSI is responsible for updating language ie. adding new features, updating existing features, deleting unused features.
- ANSI C standards:

1. Before 1989 : The C Prog Lang Book
2. C89 : 1989
3. C90 : 1990
4. C95 : 1995
5. C99 : 1999
6. C11 : 2011
7. C18 : 2018

C Language Basics

```c
#include<stdio.h>
int main( void )
{
    printf("Hello World!!!");
    return 0;
}
```

- Set of statement is called program.
- An instruction given to the computer is called statement.
- Every instruction is made up of token.
- Token is basic unit of program.
- Tokens in C:

1. Identifiers
    - Name given to variable, array, function, pointer, union structure, enum etc is called identifier.
    - "main" is name of function hence it is considered as identifier.
2. Keyword
    - It is reservered word that we can not use as a identifier.
    - Kewords in C:
        1. The C Prog Language (1st Edition) : 28 keywords

2. The C Prog Language (2nd Edition) : 27 keywords(entry keyword was removed)
3. C89 : 5 keywords
4. C99 : 5 keywords
5. C11 : 7 Kewords

3. Constant / Literal
    - An entity whose value we can not change is called constant.
    - Types:
        1. Character constant. e.g 'A'
        2. Integer constant
            1. Decimal Constant
            2. Octal Constant
            3. Hexadecimal Constant
        3. Floating Point Constant
            1. Float constant. e.g 3.14f
            2. Double constant. e.g 3.14
        4. String constant. e.g "CDAC"
        5. Enum Constant

```
enum ShapeType
{
    EXIT, LINE, RECT, OVAL  //Enum constant
};
```

4. Operator

- If we want to create expression then we should use operator
- Types:
    1. Unary Operator e.g ++, --, ~,!, sizeof,& etc
    2. Binary Operator
        1. Arithmetic operator e.g +, -, *, /, %
        2. Relational Operator e.g <, >,>=, <=, ==, !=
        3. Logical Operator e.g &&, ||
        4. Bitwise operator e.g &, |, ^, <<, >>
        5. Assignment operator e.g =, Shorthand operators
    3. Ternary OPerator e.g Conditional operator(? :)

5. Punctuator / Seperator

- ; : , space, tab, { }[ ]< > etc

Software Development Kit

- SDK = Language tools + Documentation + Supporting Library + Runtime Env.
- Language tools
    1. Editor
        - Notepad, Edit Plus, gedit, vim, TextEdit, MSVS Code etc
        - It is used to develop/edit source code.

2. Preprocessor
    - CPP(C/C++ preprocessor)
    - Job of preprocessor:
        1. To remove the comments
        2. To expand macros
3. Compiler
    - For Mrcrosoft Visual Studio : cl.exe
    - For Linux : gcc
    - For Intel : icc
    - For Borland : tcc
    - Job of Compiler:
        1. To check syntax
        2. To convert high level source code into low level code(Assembly)
4. Assembler
    - For Borland : TASM
    - For MSVS : MASM
    - For Linux : as
    - Job of Assembler:
        1. To convert low level code into machine code.
5. Linker
    - For Borland : TLINK.exe
    - For MSVS : Link.exe
    - For Linux : ld
    - Job of linker
        1. .obj/.o file contains machine code. This file is also called as almost executable. Linker is responsible for linking .o file to glibc.so.
6. Loader
    - It is operating system API, which is responsible for loading executable file from HDD into RAM.
7. Debugger:
    - For Linux : gdb
    - For Windows : windbg
    - Job of Debugger:
        1. It is used to find the bug.
8. Profiler:
    - For Linux : valgrind
    - Job of profiler:
        1. To debug the memory and detetcting memory leakage.

- Documentation:
    1. For Windows : MSDN
    2. For Linux : man pages
- Supporting Library:
    1. glibc.so
    2. BOOST, QT
- Runtime Environment
    - It is responsible for managing execution of C application.

    - Runtime Environment for C is "C runtime".

Data Type

- It describes 3 things about variable / object

1. Memory : How much memory is required to store the data.
2. Nature : Which type of data memory can store
3. Operation : Which operations are allowed to perform on data stored inside memory.

- Types of data types:
    1. Fundamental Data Types
    2. Derived Data Types
- Fundamental Data Types(5)
    1. void : Not Specified
    2. char : 1 byte
    3. int : 4 bytes
    4. float : 4 bytes
    5. double : 8 bytes
- Derived Data Types(5)
    1. Array
    2. Function
    3. Pointer
    4. Union
    5. Structure

Type Modifiers(4)

```
1. short
2. long
3. signed
4. unsigned
```

Type Qualifiers(2)

```
1. const
2. volatile
```

Constant and variable

- An entity whose value we can not modify is called constant.
- constant is also called as literal.
- e.g 'A', "Pune", 3.14, 0 etc.
- An entity whose value we can modify is called variable.
- Variable is also called as object/instance.

- e.g. int number; Here number is variable.

Comments

- If we want to maintain documentation of source code then we should use comments.
- Types:
    1. //Single line comment
    2. /* Multiline comment. */

Main function

- According to ANSI, main should be entry point function of C/C++.
- Programmer is responsible for defining main function hence it is considered as user defined function.
- Calling/invoking main function is responsibility of operating system. Hence it is also called as Callback function.
- Since main function is responsible to give call to the other functions, it is also called as calling function.
- Signature of main function;

1. void main( );
2. void main(void);
3. int main(void);
4. ☐);
5. ☐);

- Standard Syntax of main function is:

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

Function Declaration and Definition

```
//Function Definition
int main( void )        //Calling Function
{
        void print( void );     //Local Function Declaration

        print( );       //Function Call
        return 0;
}
//Function Definition
void print( void )      //Called Function
{
        printf("Inside print function\n");
}
```

- Implementation of function is called function definition.
- Local definitions are not allowed in C/C++. In other words, we can not define function inside another function.
- If we use function before its definition then it is mandatory to provide its signature to the compiler. It is called function declaration.
- It is possible to declare function locally as well globally.
- Without definition, if we try to access any element then linker generates error.

```
//Function Definition
int main( void )        //Calling Function
{
        //Local Function Declaration
        void print( void );

        //Function Call
        print( );       //Linker error

        return 0;
}
```

- If we try to build and execute project without main function then linker generates error.

Variable Declaration and Definition

- Declaration refers to the place where nature of the variable is stated but no storage is allocated.
- Definition refers to the place where memory is assigned or memory is allocated.

```
int main( void )
{
    int num1;   //Declaration as well as defintion

    int num2 = 20; //Declaration as well as definition

    extern int num3;    //Declaration
    return 0
}
int num3 = 30;  //Declaration as well as definition
```

Variable Initialization and Assignment

```
int num1 = 10;  //Initialization
```

- Initialization is the process of storing value inside variable during its declaration.
- We can initialize variable only once.

```
int num1 = 10; //Initialization
num1 = 20;  //Assignment
num1 = 30;  //Assignment
```

- Assignment is process of storing value inside variable after its declaration.
- we can assign value to the variable multiple times.

Day 2

L-Value(Locator Value)

- Non constant(editable/modifiable) memory location which is available at left hand side of assignment operator is called locator value(L-Value).
- Consider Following code:

```
2 + 3 = 5;  //Error - L-Value Required
```

- Consider Following code:

```
5 = 2 + 3; //Error - L-Value Required
```

- Consider Following code:

```
const int number = 10;
number = number + 5; //Error - L-Value Required
```

- Consider Following code:

```
int number = 10;
number = number + 5; //OK - number is L-Value
```

R-Value(Reference Value)

- A constant, variable or expression which is used at right hand side of assignment operator is called R-Value.

```
int num1 = 10;  //10 - R-Value
```

```
int num1 = 10;  //10 - R-Value
int num2 = num1;//num1 - R-Value
```

```
int num1 = 10;  //10 - R-Value
int num2 = num1;//num1 - R-Value
int num3 = num1 + num2; //(num1 + num2) - R Value
```

5 keywords introduced in C89

```
1. const
2. volatile
3. void
4. enum
5. signed
```

Constant in C

```
int num1 = 10;
num1 = num1 + 5;    //15
```

- Once initialized, if we dont want to modify value/state of the variable/object then we should use const keyword.
- const keyword is introduced by ANSI in C89.

```
const int num1 = 10;
num1 = num1 + 5;        //Not OK
```

- Constant variable is also called as read only variable.
- In C, it is optional to initialize constant variable.
- In C, we can declare variable constant but we can not declare function constant.

Pointer in C

- Named memory location is called variable.
- & is a unary operator which is used to get address of variable/object.
- If we want to store address then we need to declare pointer in a program.
- A pointer is a variable which is used to store address of another variable.
- Size of any type of pointer on 16-bit compiler is 2 bytes, on 32-bit compiler 4 bytes and on 64 bit compiler 8 bytes.
- Uninitialized pointer is called wild pointer

```
int main( void )
{
        int *ptrNum1;   //Wild POinter
        return 0;
}
```

- NULL is a macro whose value is 0 address

```
#define NULL ((void*) 0)
```

- If pointer contains NULL value then such pointer is called NULL pointer.

```
int main( void )
{
        int *ptrNum1 = NULL;//ptr1Num1 : NULL Pointer
        return 0;
}
```

- Pointer initialization:

```
int main( void )
{
        int num1 = 10;  //Initialization
        int *ptrNum1 = &num1;   //Initialization
        return 0;
}
```

- Pointer assignment:

```
int main( void )
{
        int *ptrNum1 = NULL; //Initialization
        int num1 = 10;  //Initialization
        ptrNum1 = &num1;        //Assignment
        return 0;
}
```

- Process of accessing value of the variable using pointer is called dereferencing.

Constant and pointer combination

```
int *ptr
```

- In above statement, ptr is non constant pointer variable which can store address of non constant integer variable.
- Consider following example:

```
int main( void )
{
        int *ptr = NULL;
        int num1 = 10;
        ptr = &num1;
        *ptr = 50;      //Dereferencing
        printf("Num1    :       %d\n", *ptr);//Dereferencing

        int num2 = 20;
        ptr = &num2;
        *ptr = 60;      //Dereferencing
        printf("Num2    :       %d\n", *ptr);//Dereferencing
        return 0;
}
```

```
const int *ptr
```

- In above statement, ptr is non constant pointer variable which can strore address on constant integer variable.

```
int main( void )
{
        const int *ptr = NULL;

        const int num1 = 10;
        ptr = &num1;    //OK
        //*ptr = 50;    //Not OK
        printf("Num1    :       %d\n", *ptr);//Dereferencing : OK

        const int num2 = 20;
        ptr = &num2;    //OK
        //*ptr = 60;    //Not OK
        printf("Num2    :       %d\n", *ptr);//Dereferencing : OK
        return 0;
}
```

```
int const *ptr
```

- Above is 100% same as "const int *ptr"

```
const int const *ptr
```

- Above state is same as "const int *ptr" or "int const *ptr"
- For above compiler will generate warning: "Duplicate const qualifier".

```
int *const ptr
```

- In above statement, ptr is constant pointer variable which can store address of non constant integer variable.

```
int main( void )
{
        int num1 = 10;
        int *const ptr = &num1; //OK
        *ptr = 50;      //Dereferencing : OK
        printf("Num1    :       %d\n", *ptr);//Dereferencing : OK

        int num2 = 20;
        //ptr = &num2;  //Not OK
        return 0;
}
```

```
int *ptr const
```

- Above syntax is invalid.

```
const int *const ptr
```

- In above statement, ptr is constant pointer variable which can store address of constant integer variable.

```
int main( void )
{
        const int num1 = 10;

        const int *const ptr = &num1;   //OK
        //*ptr = 50;    //Not OK
        printf("Num1    :       %d\n", *ptr);//Dereferencing : OK

        const int num2 = 20;
```

```
        //ptr = &num2;  //Not OK
        return 0;
}
```

```
int const *const ptr
```

- This statement is same as "const int *const ptr"

Structure

- If we want to group related data elements together then we should use structure. Related data elements may be of same type or different type.
- Structure is derived data type.
- if we want to define structure then we should use struct keyword.

```
struct Employee
{
    char name[ 30 ];
    int empid;
    float salary;
};
```

- We can declare structure inside function. It is called local structure.
- We can not use object and pointer of local structure outside function.
- We can define/declare function inside structure.
- If we want to store value inside structure then we must create its object.

```
struct Employee emp;
```

- If we create object structure then all the variables declared inside structure get space inside it.
- If type allows us to initialize its element using initializer list then it is called aggregate type and object is called aggregate object.

```
struct Employee emp = {"Abc",33,45000.50f};
```

- Following types are aggregate types:
    1. Array
    2. Structure
    3. Union
- Using object, if we want to access members of the structure then we should use dot/member selection operator.

```
printf("Name    :       %s\n", emp.name );
printf("Empid   :       %d\n", emp.empid);
printf("Salary  :       %f\n",emp.salary);
```

- Using pointer, if we want to access members of structure then we should use arrow/dereferencing operator.

```
printf("Name    :       %s\n", ptr->name );
printf("Empid   :       %d\n", ptr->empid);
printf("Salary  :       %f\n",ptr->salary);
```

- Parameter and argument

```
//a,b -> Function parameter / parameter
void sum( int a, int b )
{
    int c = a + b;
    printf("Result  :   %d\n",c);
}
```

```
sum( 10,20 );   //Function Call
//10,20 -> Function argument / argument
```

```
int x = 10, y = 20;
sum( x, y );    //Function Call
//x,y -> Function argument / argument
```

- In C language, we can pass argument to the function using 2 ways:
    1. By Value
    2. By Address/Reference
- If we declare structure outside function then it is called global structure. We can create object and pointer of global structure anywhere in the program.
- Procedure oriented programming is a kind of programming in which we try to solve real world problems using structure and function.
- Object oriented programming is a kind of programming in which we try to solve real world problems using class and object.
- If we want to control visibility of members of structure/class then we should use access specifier.
- Access specifiers in C++
    1. private(-)
    2. protected(#)
    3. public(+)