Advanced Programming Language

Presentation on

Programming Concepts

&

Fundamentals of

C++ Programming

Contents

- Compilation process
- Program Model
- Introduction to C
- Preprocessors
- Functions & Macros
- Data Types & Type casting
- Precedence & Associativity
- Control Statements

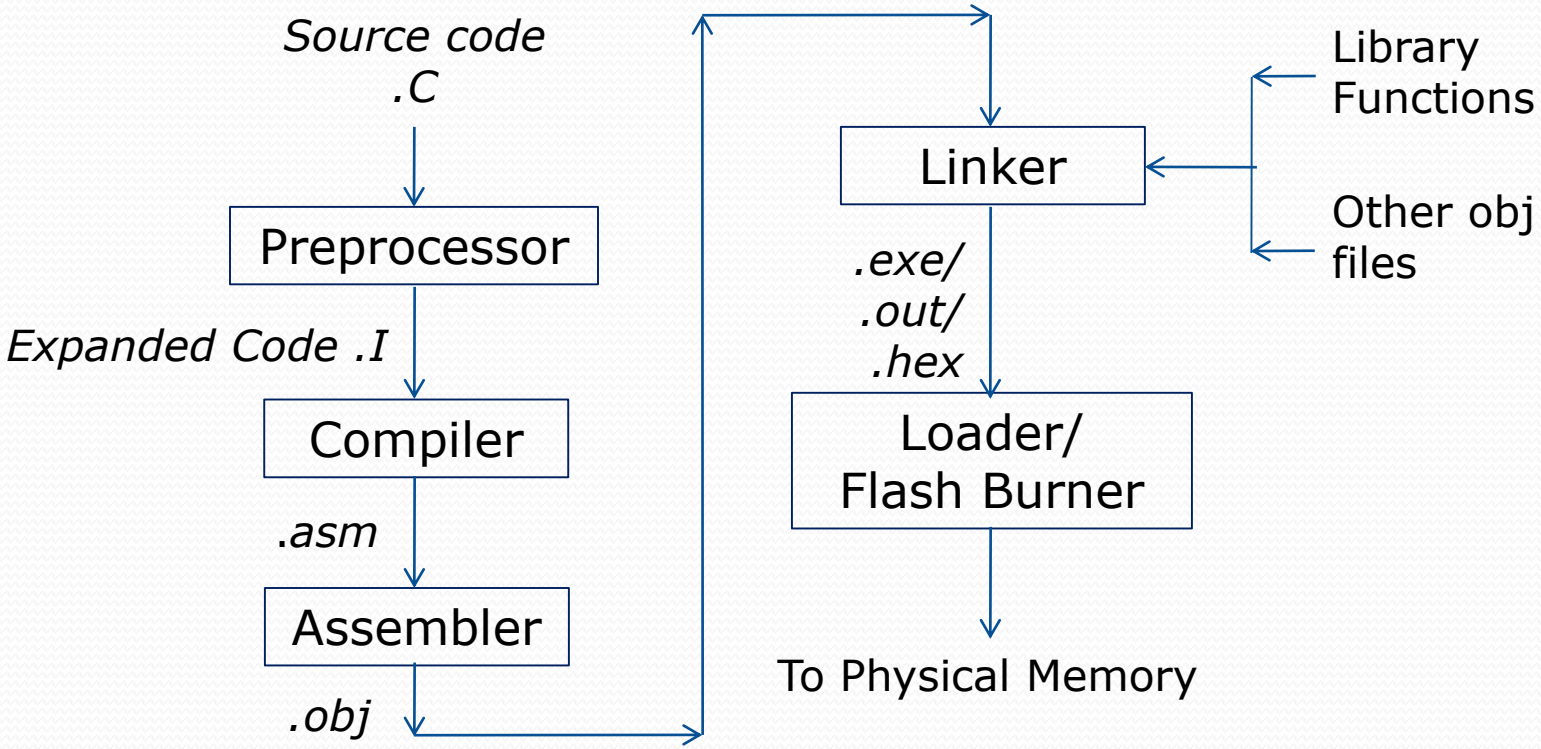
Introduction to C

Characteristics of C

- Compiler language
- It has simplicity of High level language and Power of Low level language
- Includes 32 keywords
- Structured programming language

Compilation Process

Process of converting High Level Language program into executable file.



Program Model

- ✓ Preprocessors
  - *#include*
  - *#define*
  - *#ifndef*
  - *#endif*
  - *#pragma*
- ✓ Header Files
  - *#include <std_header_files>*
  - *#include "user_defined_header_files"*
- ✓ Global declarations
- ✓ Extern declarations

Program Model Continued…

- ✓ Function prototype declaration (Exempted in CPP)

  Return_data_type function_name(parameter list);

- ✓ Function definition

  Return_data_type function_name(parameter list with name);

  {

  Local variables

  Input/invoke statements

  Logic implementation

  Output/return statements

  }

Programming Skill

- Requirement
- Modularisation
- Algorithm
- Coding
- Testing
- Validation
- Upgradation / Revisions

Preprocessors

Directives are:

- Macro expansion:
  - *#define PI 3.1415*
  - *#define AND &&*
  - *#define AREA(x)* (3.14*x*x)
- File inclusion
  - *#include <std_header_files>*
  - *#include "user_defined_header_files"*

Preprocessors Continued…

- Conditional Compilation
  - *#ifdef – if defined*
  - *#ifndef – if not defined*
  - *#else – else condition*
  - *#if – if condition*
  - *#elif – else if condition*
  - *#endif - end if condition*
- Miscellaneous directives
  - *#undef – undefine a macro already defined.*
  - *#pragma – Suppresses a specific warning.*
    - *-rlv – return value*
    - *-par – parameter not used*
    - *-rch – unreachable code*

Functions & Macros

Advantages of Functions

- ✓ Program can be divided into small modules.
- ✓ Avoids repetition of code as same function can be called many times.
- ✓ Program becomes easily understandable, modifiable.
- ✓ Debugging and testing becomes easier.
- ✓ Functions can be stored in library and reused in other programs.

Advantages of Macros

- ✓ Program becomes more readable and understandable.
- ✓ Avoids typing errors.
- ✓ Easily modifiable.

Comparison of Functions & Macros

Functions	Macros
Memory optimized	Execution optimized
Compiled with source code	Preprocessed
Local temporary variables can be declared	Local temporary variables can not be declared

Data Types

Basic Data Type	Data type with qualifier	Size (Bytes)
Char	Signed	1
	Unsigned	1
Int	Signed	2
	Unsigned	2
	Short int	1
	Unsigned short	1
	Long int	4
	Unsigned long	4
Float	Float	4
Double	Double	8
	Long double	10

Properties of Variable

- Name of variable
- Data type of variable
- Storage class/modifier
  - Automatic – Garbage value
  - Static - Zero
  - Extern - Zero
  - Register - Garbage
- Default value or assigned value

Conversion / Format specifications

Specification	Meaning
%c	Print/scan character
%d	Print/scan signed decimal integer
%i	Print/scan signed integer (Base depends on prefix)
%u	Print/scan unsigned decimal integer
%o	Print/scan unsigned octal integer
%x	Print/scan unsigned hex integer using a, b, c, d, e, f
%X	Print/scan unsigned hex integer using A, B, C, D, E, F

Conversion / Format specifications continued…

Specification	Meaning
%f	Print/scan floating point number (6 digits after decimal point)
%e	Print/scan floating point number in exponential format
%E	Print/scan floating point number in exponential format (uses E)
%g	Print/scan floating point number in %f or %e whichever is smaller
%G	Print/scan floating point number in %f or %E whichever is smaller
%s	Print/scan string
%%	Print % sign
%p	Print pointer in hex format

Operator Precedence and Associativity

Operator	Description	Precedence	Associativity
() [] -> .	Function call Array subscript Arrow operator Dot operator	1	Left to Right
+ - ++ -- ! ~ * & (datatype) sizeof	Unary plus Unary minus Increment Decrement Logical Not 1's Complement Indirection Address Typecast Size in bytes	2	Right to Left

Operator Precedence and Associativity continued…

Operator	Description	Precedence	Associativity
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
== !=	Equal to Not equal to	7	Left to Right

Operator Precedence and Associativity continued…

Operator	Description	Precedence	Associativity
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
? :	Conditional operator	13	Right to Left
= *=, /=, %= +=, -= &=, ^=, |= <<=, >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right

Control Statements

- If-else and else if
- While loop
- Do while
- For loop
- Switch case
- Continue statement
- Break statement

Structural and Object Oriented Paradigm

Structural	OOP
Program is divided into small parts called **functions**.	Program is divided into small parts called **objects**.
Top down approach.	Bottom up approach.
Less secure.	More secure.
Function is more important than data.	Data is more important than function.
Solution inspired	**Real world** inspired
Examples: C, FORTRAN, Pascal, Basic etc.	Examples: C++, Java, Python, C# etc.

Object Oriented Programming

- Built on Structure programming and Data Abstraction Concepts.
- Combines both data and functions into single unit (Object).
- Object's functions are called member functions or methods.
- The program is first divided into Objects instead of Functions.
- Namespace is declared to avoid conflict of names.

Properties of C++

- Abstraction
- Encapsulation
- Reusability
- Inheritance
- Overloading
- Interface
- Polymorphism
- Overriding

Classes & Objects

- Class & Object:
  - Class is a group of data & functions.
  - Creating class does not consume memory.
  - Objects are the instances of class.
  - Attributes are variables where information is stored

Syntax:

```
Class class_name
{
int x;
}; obj1, obj2
```

Access Specifiers

- Public:
  - ✓ The members declared as Public are accessible from outside the Class through an object of the class.
- Protected:
  - ✓ The members declared as Protected are accessible from outside the class **BUT** only in a class derived from it.
- Private:
  - ✓ These members are only accessible from within the class. No outside Access is allowed.

Abstraction

- Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.

- Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object.

Encapsulation & Reusability

- Data encapsulation is a mechanism of binding the data, and the functions that use them.
- Keeps both data and function safe from external access.
- Lead to data hiding concept

- Property by which the classes declared can be reused in other programs.

Inheritance

- It is a property by which classes can be derived from other classes (Base class).
- Derived class acquires all property of base class.
- Additional properties also can be added.
- Class can be derived from more than one base classes – Multiple Inheritance
- Syntax
  - ✓ *class derived_class : access_specifier base_class*
  - ✓ *class derived-class: access baseA, access baseB....*

Property Derived in Inheritance

- Public
  - ✓ Public members => public members
  - ✓ Protected Members =>Protected Members
  - ✓ Private Members => Not accessible
- Protected
  - ✓ Public members => Protected members
  - ✓ Protected Members =>Protected Members
  - ✓ Private Members => Not accessible
- Private
  - ✓ Public members => Private Members
  - ✓ Protected Members => Private Members
  - ✓ Private Members => Not accessible

Polymorphism

- Occurs when there is hierarchy of inherited classes.
- More than one function is defined with same name.
- Function call depends on Object calling the function.
- Virtual functions are used to avoid early binding/static Linkage and perform Late binding/dynamic Linkage.

Overloading

- Defining multiple functions with same name is termed as Overloading.
  - ✓Function overloading:

```
Eg.   void print(int x, int y){
      cout<<"Printing int:"<<i<<endl; }
      void print(float i){
      cout<<"Printing float:"<<i<<endl; }
```

  - ✓Operator overloading:

```
Eg.   complex operator +(complex c1, complex c2)
      {body}
```

Function Overriding

- Defining multiple functions with same name and parameter list.
- But they should be in different classes.
- Function is identified based on the object calling the function.

Concepts of C++

- Flexible declaration: Variables can be declared at point of usage.
- Typecasting:
  - ✓ (Data_type) var_name //C type
  - ✓ Data_type(var_name) // C++ type
- The :: Operator : Scope resolution operator

```
int a=10;
void main(){  int a=15;
              cout<<a<<::a;::a=20;
              cout<<a<<::a;}
```

- Const pointer & references : Avoid accidental modification of variable.