C++Programming

DAY 1:

INTRODUCTION TO OBJECT ORIENTED PROGRAMMING

Index

- Session Objectives
- Object Oriented Programming
- C++ Language
- Writing Functions in C++
- Writing a class & Creating its Object
- Adding members to the class
- Access specifier considerations
- Conclusion

Session Objectives

Session Objectives

- Awareness on the evolution of C++ language
- To review C++ programming basics
- To demonstrate the use of C++ compiler for simple C++ programming
- To make participants comfortable in creating / using classes and objects
- To initiate the object oriented thought process among the participants
- To develop the problem solving abilities using object oriented programming techniques
- To make participants aware about the additions in modern C++ standards

Installation of C++ IDE

Object Oriented Programming

Object Oriented Programming

- What is OOP?
- Major Pillars of object oriented programming
- Why OOP?
- Introduction to VC++ IDE
- Steps in creating a simple C++ program

What is OOP?

In nutshell, the Object Oriented Programming is an effective way of solving complex programming problems

What are the difficulties in software development?

The software development process is inherently chaotic

Comprehension is difficult to achieve

Refinements in created software are unavoidable

There are various approaches to tackle the programming problem, the two major approaches are as follows:

- Procedural Approach
- Object Oriented Approach

Programming Problem

- Any program has following two things:
  - Procedure
  - Data
- To put in different way, Program = Algorithm + Data
- Hence programming problem can be approached either first considering procedure and then data or vice versa
- When you focus first on algorithm i.e. process/procedure to be executed then the resulting approach is procedure oriented programming approach
- When you focus first on data then the resulting approach is object oriented programming approach.

Procedural Approach

- The Procedural Approach helps to some extent to solve the programming problem, by modeling a programming problem as a set of processes and subroutines etc.
- The major aspect of Procedural Approach is that it considers process/procedure first and data comes later. In short, initially data is somewhat neglected. And indeed this is fine if programming problem is process oriented with limited data
- But as complexity grows, especially when time comes to handle voluminous data this approach of solving programming problem shows limitations
- These limitations are as follows:
  - Modelling the problem domain is difficult
  - Insecurity of data
  - Inflexible software
- Example programming languages, which are following procedural approach are: C, Pascal, Fortran etc.

Object Oriented Approach

- The **Object Oriented** Approach helps to solve the programming problem, by modeling a programming problem as a set of objects and their interactions etc.
- The essential aspect of **Object Oriented** approach of solving programming problem is that it first considers data and then possible operations on that data
- The limitations observed in procedural programming are easily overcome by **Object Oriented** approach, and it provides following advantages:
  - Realistic modeling
  - Reuse of code
  - Resilience to modification
- Example programming languages, which are following **Object Oriented** approach are: C++, Python, Java, Eiffel etc.

Demonstration #1

HELLOWORLD

Major Pillars of Object Model

The follow the object oriented approach the object model was proposed by a well known software researcher, Grady Booch,

This object model focuses on various principle of object oriented programming, and are considered as Major Pillars of OOP

Following are four Major Pillars of OOP

- Abstraction
- Encapsulation
- Hierarchy
- Modularity

Grady Booch has also proposed three minor pillars-Typing, Persistence & Concurrency

Abstraction

An abstraction is the process of deciding an essential attributes of an object, while at the same time ignoring the non-essential attributes of an object, from the perspective of the viewer.

Encapsulation

- Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from the outside interference and misuse
- Typically, the public parts of an object are used to provide a controlled interface to the private elements of the object
- Encapsulating the data and behavior into a single object is of primary importance in OO development.
- A single object contains both its data and behaviors and can hide whatever it wants from other objects.

Hierarchy

- As a result of abstraction there may be various hierarchies-
  - inheritance hierarchies or
  - composition hierarchy
- Inheritance is the process by which one object can acquire the properties of another object.
- Inheritance is important because it supports concept of classification
- Inheritance—A class can inherit from another class and take advantage of the attributes and methods defined by the class
- Composition means when one object is contained in another object
- Both the forms of hierarchies gives rise to reuse of code

Modularity

Polymorphism

- Polymorphism—Polymorphism means that similar objects can respond to the same message in different ways. For example, you might have a system with many shapes. However, a circle, a square, and a star are each drawn differently.
- Using polymorphism, you can send each of these shapes the same message (for example, draw), and each shape is responsible for drawing itself
- In simple terms polymorphism is the attribute that allows one interface to control access to a general class of actions
- The specific action selected is determined by the exact nature of the situation
- C++ supports both, compile time and run time polymorphism

OO Terminology

- Object
- Class
- Data member
- Member function
- Static member
- Inheritance
- Polymorphism
- Composition
- Aggregation
- Association
- Message passing
- Method
- overloading
- Abstract class
- Virtual/pure virtual function
- Dynamic polymorphism
- Multiple inheritance
- Constructor
- Destructor
- Accessor
- Mutator
- Public interface
- Overriding
- Class scope

C++ Language Basics

C++ Language Basics

Keywords,

data types,

Identifiers,

operators,

control statements,

Comments,

Header files,

Keywords

Following are the keywords in C++ language *(before C++11 standard)*

asm	auto	bool	break	case	catch	char
class	const	const_cast	continue	default	delete	do
double	dynamic_cast	else	enum	explicit	export	extern
false	float	for	friend	goto	if	inline
int	long	mutable	namespace	new	operator	private
protected	public	register	reinterprete_cast	return	short	signed
sizeof	static	static_cast	struct	switch	template	this
throw	true	try	typedef	typeid	typename	union
unsigned	using	virtual	void	volatile	wchar_t	while

C++ keywords
(after introducing C++11 standard)

- Keywords are reserved words of the language
- All C++ keywords are in lower case
- Also, some keywords involve underscore as a word separator
- You can not use any keyword as an identifier
- There are totally 97 keywords in C++, as of 6-Jan-2024

A - C	D - P	R - Z
alignas (C++11)	decltype (C++11)	reflexpr (reflection TS)
alignof (C++11)	default (1)	register (2)
and	delete (1)	reinterpret_cast
and_eq	do	requires (C++20)
asm	double	return
atomic_cancel (TM TS)	dynamic_cast	short
atomic_commit (TM TS)	else	signed
atomic_noexcept (TM TS)	enum (1)	sizeof (1)
auto (1) (2) (3) (4)	explicit	static
bitand	export (1) (3)	static_assert (C++11)
bitor	extern (1)	static_cast
bool	false	struct (1)
break	float	switch
case	for (1)	synchronized (TM TS)
catch	friend	template
char	goto	this (4)
char8_t (C++20)	if (2) (4)	thread_local (C++11)
char16_t (C++11)	inline (1)	throw
char32_t (C++11)	int	true
class (1)	long	try
compl	mutable (1)	typedef
concept (C++20)	namespace	typeid
const	new	typename
consteval (C++20)	noexcept (C++11)	union
constexpr (C++11)	not	unsigned
constinit (C++20)	not_eq	using (1)
const_cast	nullptr (C++11)	virtual
continue	operator (4)	void
co_await (C++20)	or	volatile
co_return (C++20)	or_eq	wchar_t
co_yield (C++20)	private (3)	while
	protected	xor
	public	xor_eq

- (1) — meaning changed or new meaning added in C++11.
- (2) — meaning changed or new meaning added in C++17.
- (3) — meaning changed or new meaning added in C++20.
- (4) — new meaning added in C++23.

https://en.cppreference.com/w/cpp/keyword

Data Types

Following are the brief classification of fundamental data types in C++

Integral types -numeric	Integral types -character	Floating-point types
bool short int unsigned short int int unsigned int long int unsigned long int long long int unsigned long long int	char signed char unsigned char char16_t char32_t wchar_t	float double long double

Demonstration #2

INITIALIZATIONS, CONSTANTS, VARIABLES AND TYPE CASTING

Operators

- C++ is very rich in its built-in operators
- There are four main categories of operators and some special category of operators

Main Operator categories in C++:

- Arithmetic
- Relational
- Logical
- Bitwise

Special categories of operators:

- Assignment
- Ternary
- Address of and de-referencing operator
- The sizeof operator
- Other operators

Demonstration #3

SAMPLE OPERATOR USAGE- UNARY, COMPOUND ASSIGNMENT, INSERTER, EXTRACTOR, NEW AND DELETE

Writing Functions in C++

Function

Function prototype

Function definition

Function call

Passing arguments to a function

Returning a value from a function

identifier scopes in C++

Review of Control statements in C++

- **Decision making statements**
  - if … else
  - switch … case
- **Iteration statements**
  - while
  - do … while
  - for
- **Unusual control flow statements**
  - break
  - continue
  - goto

Review of Arrays and pointers in C

- **Arrays**
  - Single dimensional numeric and characters arrays
  - Multidimensional arrays
- **Pointer**
  - Pointer to fundamental data type
  - Pointer to array
  - Pointer to pointer
  - Pointer manipulations and its Limitations
  - Dynamic memory allocation
  - Passing and returning pointer to and from function
  - Pointer initialization, Null pointer and Dangling pointer

Review of Structures and Unions in C

- Structures
  - Defining structure and declaring structure variables
  - Passing structure to and returning it from a function
  - Pointer to structure
  - Passing pointer to structure structure to and returning it from a function
- Unions
  - Pointer to fundamental data type
  - Pointer to array
  - Pointer to pointer
  - Pointer manipulations
  - Limitations on Pointer manipulations
  - Dynamic memory allocation
  - Passing and returning pointer to and from function

Programming Problems

1. Draw a flowchart and then write a program to generate the following pattern of symbols and numbers:

```
*
* *
* * *
* * * *
```

```
1
2 3
3 4 5
4 5 6 7
```

```
1
2 1
3 2 1
4 3 2 1
```

```
   1
  2 1
 3 2 1
4 3 2 1
```

```
1
1 2
1 2 3
1 2 3 4
```

```
      1
    2 3 2
  3 4 5 4 3
4 5 6 7 6 5 4
```

Writing class & Creating Object

WRITING CLASS SYNTAX

CREATING OBJECT SYNTAX

WRITING DRIVER PROGRAM FOR USING OBJECTS

What is class?

A Class is a collection of objects having same attributes, i.e. properties and behavior

Any real world object we can classify based on two things:

- Properties
- Behavior

Example: A car has seating capacity, maximum speed, steering wheel, brake etc. as properties of the car,; whereas start, stop, accelerate, idle and turn etc. are behaviors of a car

C++ Syntax to create a class

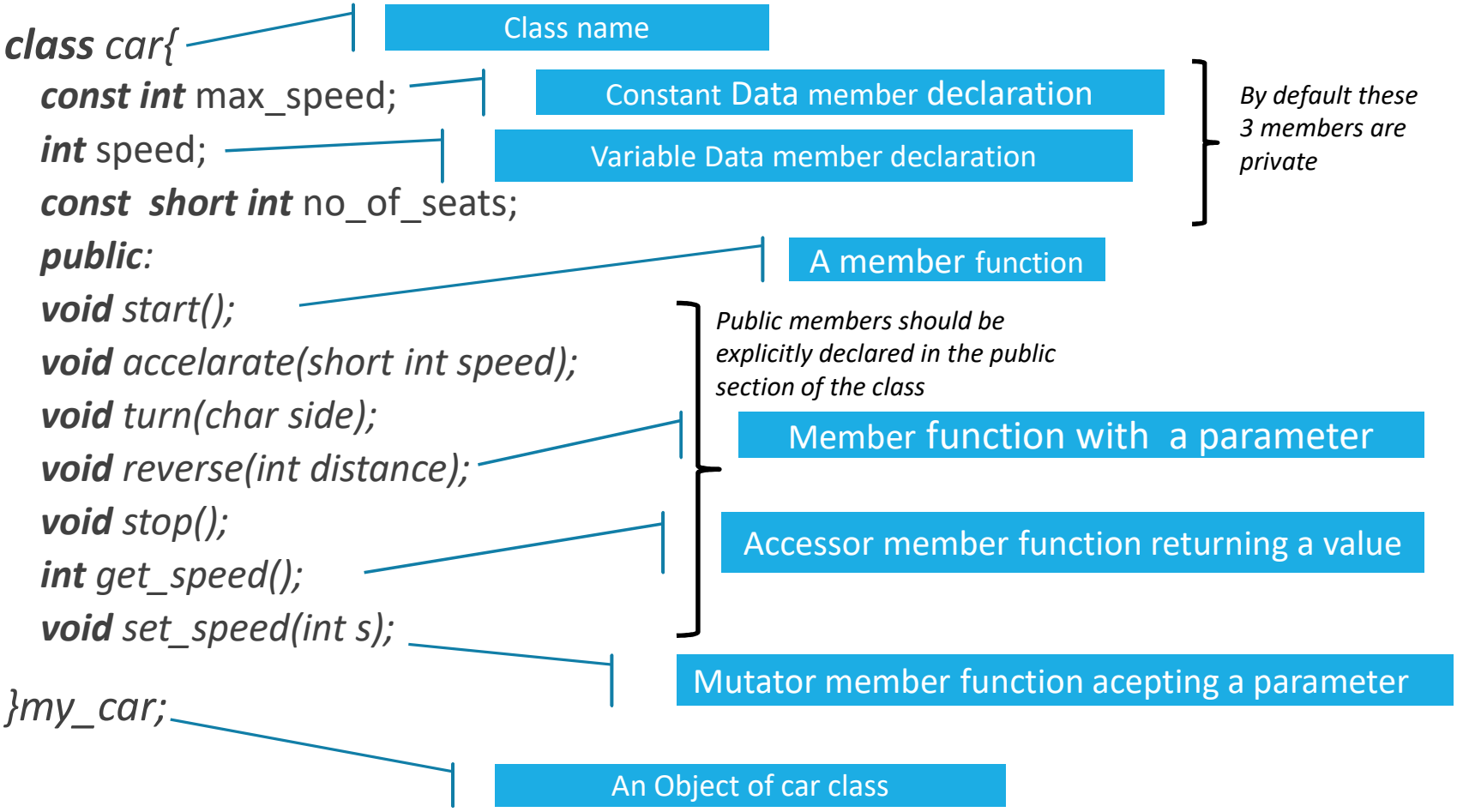
```
class <class_name>{
   [private]
   data_member(s) declaration(s);
   :
   member_function(s) ([parameter_declaration_list]) declaration(s)
   public:
   data_member(s) declaration(s);
   :
   member_function(s) ([parameter_declaration_list]) declaration(s)
}[object_declaration_list];
```

Writing a C++ class

```
class car{
   const int max_speed;
   int speed;
   const  short int no_of_seats;
   public:
   void start();
   void accelarate(short int speed);
   void turn(char side);
   void reverse(int distance);
   void stop();
   int get_speed();
   void set_speed(int s);

}my_car;
```

Writing a C++ class



Demonstration #4

CREATING A CLASS AND ITS OBJECT

Adding various members to the class

CONSTRUCTORS

GETTER-SETTER METHODS

STATIC MEMBERS

CONSTANT MEMBER FUNCTION

CONSTANT OBJECT

THIS POINTER CONSIDERATIONS

Class Members

Data Members

- variable Class field
- constant Class field
- static Class field

Member Functions

- Constructors
- Destructor
- Getters
- Setters
- Non-static member function
- Static member function
- Conversion function
- Operator overloading function

Constructor

- The constructor is a special member function of a class, which is exactly invoked once only immediately after the creation of object.
- Following are the specialties of a constructor member function
  - The purpose of the constructor is to initialize the object, immediately after its creation.
  - A constructor has the same name as that of its class name
  - A class can have one ore more constructors
  - The concept of return value is not applicable to a constructor
  - Usually, a constructor is a public member function

Destructor

- A destructor is also a special member function similar to a constructor, but it has different specialties, as mentioned below:
  - A class can have only one destructor
  - A destructor name is same as that of class, but prefixed with ~ (tilda symbol)
  - A destructor is executed exactly once for an object just before its removal from the memory, or when it goes out of scope
  - The purpose of destructor is to release the resources allocated, such as file, network connection etc. by the object before its being removed from the memory.

Getter member functions

- A getter function is an interface of an object for other classes, which are not the part of the class under consideration
  - A getter function is invoked by the other, outside class member function to access any class field of a class under consideration
  - A getter function is always a public member function
  - A getter function must return a value of exactly one data member of the class
  - A getter function never accept any parameter, hence it does not require any parameter declaration
  - If required, a getter function can have other applicable logic as per C++ syntax
  - A getter function can be a static member function for a static data member

Setter member functions

- A setter function is an interface of an object for other classes, which are not the part of the class under consideration
  - As like a getter, setter function is always a public member function
  - Unlike getter function, a setter function never return a value of any type, hence its return type must be void
  - Unlike getter function, a setter function must accept a parameter and require a parameter declaration of respective type
  - If required a setter function can have other applicable logic as per C++ syntax, especially validation logic for a respective class field
  - A setter function can be a static member function for a static data member

C++ operators

- Following are C++ operators in addition to the operators inherited from C language:
  - Scope resolution operator, **::**
  - Reference operator, **&**
  - Insertion and extraction operators, **<<** and **>>**
  - Dynamic memory allocation and deallocation operators, **new** and **delete**
  - Pointer casting operators, static_cast, reinterprete_cast, dynamic_cast

The this pointer

- Every member function has an implicit pointer parameter passed known as the **this** pointer
- Using the **this** pointer, a member function can access other members of the class.
- Especially, when parameter passed and the data member name is same then that data member can be referred by **this** pointer

Static class members

- A member of a class, may it be data member or member function, can be a static member.
- The static members have following specialties:
  - A static data member is shared by all the objects of the class, hence its memory allocation happens once only after loading the class
  - A static data member is accessed using class name along with scope resolution operator
  - A static function invocation is done with class name along with scope resolution operator. But, since it is shared by all objects of the class, it can also be invoked with object
  - A static data member initialization is done in global scope with class name along with scope resolution operator
  - A static function can use only static data (i.e. any static variable or constant in the class)

Static class members

- The static members have following specialties (contd. . .) :
  - The memory is not allocated for static members
  - A static member variable exists before the any object of the class is created
  - The static member is initialized with zero before creating first object of the class
  - A static function can access global functions and data
  - There should can not be a static and non-static versions of the same function
  - A static member function can not be **const** or **volatile**
  - A static member function does not have **this** pointer
  - A friend static function can not be a **virtual** function

Access specifier considerations

ACCESS SPECIFIERS IN C++

- public
- private
- protected

Access specifiers in C++

The access specifier are used to declare the accessibility of a class member, and that decides who can access certain class member.

Access specifiers are very important from security point of view

Following are the access specifiers used in C++

- **public** – The class members which should be accessed from the other class are must be public
- **private** - The class members which must be accessed from within the class are declared private
- **protected** – The base class members which you want to keep accessible from the child /derived class are declared as protected

Demonstration #5

A FULLFLAGED CLASS CONTAINING:

CONSTRUCTORS,

DESTRUCTOR,

STATIC MEMBERS,

GETTER AND SETTERS

Conclusion

SUMMARY

REFERENCES

Summary

- As computer program consists of algorithm and data, the programming problem can be approached by two ways- either by considering algorithm first, known as procedural approach or data first, known as an object oriented approach
- A procedural approach shows limitations when data volume in program increases, and all those limitations can be effectively tackled using an object oriented approach
- The real life objects can be classified to create a template, which in turn creates software objects which maps to real life object
- Such a template, which can create objects with same attributes is known as a class
- A class has static-nonstatic variables/constant data members, constructors, destructor, accessors, mutators, friend functions, conversion functions, along with static-nonstatic business logic functions
- A static data member is shared by all the objects of that class

References

- C++: The Complete Reference 3rd edition by Herbert Schildt
- C++: How to program 10th edition by Dietel & Dietel
- Object oriented thought process by Matt Weisfeld