REFERENCES

1. Wakerly, J. F. (2007). Digital design: Principles and practices (4th ed.). Pr
Hall.
2. Patterson, D. A., & Hennessy, J. L. (2017). Computer organization and design
hardware/software interface (5th ed.). Morgan Kaufmann Publishers.
3. Osborne, A. (1981). An introduction to microcomputers (Vol.
Osborne/McGraw-Hill.
4. Gaonkar, R. S. (2001). Microprocessor architecture, programming,
applications with the 8085 (5th ed.). Prentice Hall.
5. Mazidi, M. A., Mazidi, J. G., & McKinlay, R. D. (2005). The 80x86 IBM PC
compatible computers: Assembly language, design, and interfacing (Vol.
Pearson Prentice Hall.
6. Mano, M. M. (2010). Digital design (5th ed.). Pearson Education.
7. Morris Mano, M. (2013). Digital design with an introduction to the verilog hdl
ed.). Pearson Education.
8. Ciletti, M. D. (2012). Digital design: A systems approach (2nd ed.). Newnes.
9. Roth, J. F. (2008). Digital system design using VHDL (2nd ed.). CRC Press.

Online resources:

10. Harris, D. M., & Harris, S. L. (2017). Digital design and computer architecture
Elsevier. https://www.elsevier.com/books/digital-design-and-compute
architecture/harris/978-0-12-385477-6
11. Brey, B. B. (2017). The intel microprocessors: 8086/8088, 80186/80188, 80
80386, 80486, pentium, pentium pro processor, pentium ii, iii, 4 (9th ed.). Pearso
https://www.pearson.com/us/higher-education/program/Brey-The-Intel-
Microprocessors-80
12. (2023). || Available at: https://www.dcaclab.com/blog/sr-flip-flop-explained-i
detail/ (Accessed: 11 February, 2023).
13. (2023). || Available at: https://www.coursehero.com/file/p5hr3abh/The-latch-i
said-to-be-disabled-Fig9-Logic-Diagram-Clocked-RS-Flip-Flop-When/ (Accesse
11 February, 2023).

PROGRAMMING LANGUAGE AND ITS APPLICATIONS (AEtE03)

3.1 INTRODUCTION TO C PROGRAMMING

Introduction to C Programming:

a) C is a general-purpose, high-level programming language that was first developed in the early 1970s.

b) C is designed to be a low-level, system programming language that provides a lot of control over the computer hardware.

c) C is a compiled language, which means that the source code is translated into machine code before it is executed.

d) C is a structured programming language that uses a top-down, modular approach to problem-solving.

C Tokens

- Predefine/reserved words
- Used by compiler, cannot be used as variable names
- Supports 32 keywords

Examples include "int," "char," and "while."

Operators

- Special symbols used to perform functions
- Operands are defined as the data items on which the operators are applied.
- Types (Based on the number of operands)

Unary Operator	Binary Operator
Operator applied to the single operand	Operator applied between two operands
Eg: Increment operator (++), decrement operator (--), sizeof, (type)*	Eg: Addition, subtraction, multiplication, and division

List of Binary Operators

- Arithmetic Operators
- Logical Operators
- Relational Operators
- Shift Operators

- Conditional Operators
- Bitwise Operators
- Assignment Operator
- Misc. Operator

Summary of operators in C programming:

a) Arithmetic operators: These operators perform arithmetic operations such as addition (+), subtraction (-), multiplication (*), division (/), and modulo division (%).

b) Relational operators: These operators compare two values and return a boolean value (true or false) based on the comparison. Examples include equal to (==), not equal to (!=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).

c) Logical operators: These operators perform logical operations such as AND (&&), OR (||), and NOT (!).

d) Assignment operators: These operators are used to assign a value to a variable. The most common assignment operator is the equal sign (=).

e) Increment and decrement operators: These operators increase or decrease the value of a variable by 1. Examples include ++ (increment) and -- (decrement).

f) Conditional operator: This operator is also known as the ternary operator and is used to select one of two values based on a condition. The syntax is (condition) ? value1 : value2.

g) Bitwise operators: These operators perform operations on the individual bits of an integer value. Examples include AND (&), OR (|), XOR (^), and NOT (~).

h) Sizeof operator: This operator returns the size of a variable or data type in bytes.

i) Comma operator: This operator is used to separate multiple expressions and execute them in order.

Formatted and Unformatted Functions

Formatted Function	Unformatted Function
Allow to supply input or display output in user desired format	Do not allow to supply input or display output in user desired format.
Store data more user friendly	Store data more compactly
Eg: printf(), scanf()	Eg: getch(), gets(), puts()

Control Statements:

- Help user to specify a program control's flow
- Specify the order of execution of the instruction that are present in the program
- Helps to make decision, perform tasks repeatedly, jump from section of one code to another.

Types:

1. **Decision Making**
   - Also called IF statements
   - Get executed when the evaluation of the condition is true. In case of the evaluation condition is false, there will be an execution of a different instruction set.
   - Types: IF, IF Else, Nested IF

2. *Selection*
   - Also called Switch Case
   - Can use when more than three alternatives or conditions exist in a program.
   - In every block there is a use of the case keyword
   - Break Statement and Default block statement are optional

3. *Loop:* A loop is a control flow statement in C programming that allows a program to repeat a set of statements while a certain condition is true. There are three types of loops in C programming
   - Includes For, While and Do-While
   - Used to execute a particular set of instructions repeatedly until a particular condition is met or for a fixed number of iterations.
   - Save time, reduce errors, make code more readable.
   - Debugging and error handling

For Loop

- Initialization statement will be executed only one time. After that, the condition will be checked and if result of condition is true it will execute the loop. If false, then for loop will terminated

Syntax The syntax for a for loop in C programming is as follows:

```
for (initialization; condition; increment/decrement)
{
  // code to be executed
}
```

While Loop

Performed only if the condition is valid and will not executed if the condition is incorrect.

Syntax:

```
while (Condition)
{
// code to be executed
}
```

Do While Loop

Do while loop is guaranteed to be done once at a time

The code inside the loop is executed at least once, even if the condition is false, because the condition is evaluated after each iteration of the loop. If the condition is true, the loop continues to execute; if it is false, the loop terminates.

Syntax:

```
do
{
```

```
// code to be executed}
while(Condition);
```

4. **Jump Statement**: A jump statement in C programming is a control flow statement that allows the flow of execution to jump to another part of the program. There are three types of jump statements in C programming:
   a) break statement: terminates a loop or switch statement and transfers control to the next statement outside of the loop or switch.
   b) continue statement: skips the current iteration of a loop and moves on to the next iteration.
   c) goto statement: transfers control to another part of the program specified by a label.

User Defined Functions:

User-defined functions are used to improve code readability, reuse code, and modularize the program.

Block of code that performs a specific task and can be called multiple times from different parts of the program

Also called programmer's defined function.

Programmers write their own functions according to their need

Main Components: Function Prototype or Function Declaration, Function Definition and Function Call

Syntax

```
Syntax: return_type function_name(parameter_list) {
  // function body
}
```

E.g.: you need to create a triangle and color it based upon size and color:

Createtriangle()

Color()

Note:

Recursive Functions:

Function call itself repeatedly until some condition has been satisfied.

A recursive function is a function that calls itself, either directly or indirectly, to solve a problem.

a) Reduce unnecessary calling of function
b) Reduce size of code
c) Solve data structure problem
d) Reusability


```
#include <stdio.h>

int factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}

int main() {
  int n = 5;
  int result = factorial(n);
  printf("The factorial of %d is %d\n", n, result);
  return 0;
}
```

In this example, the function factorial calculates the factorial of a given number n by calling itself recursively until the base case (n == 0) is reached. The base case returns a value of 1, and the recursive case multiplies n by the result of the recursive call with n - 1.

The factorial of 5 is 120

Note: Recursive functions are useful when a problem can be divided into smaller subproblems that are similar in nature to the original problem. They are often used to solve problems related to data structures like trees, linked lists, and graphs.

However, recursive functions can be memory-intensive and may cause stack overflow if the recursion depth is too large. Therefore, it is important to ensure that the base case is reached within a reasonable number of recursive calls.

The basic structure of a recursive function includes a base case and a recursive case, which involves calling the function itself

Array (1-D, 2-D, Multi-dimensional), and String manipulations

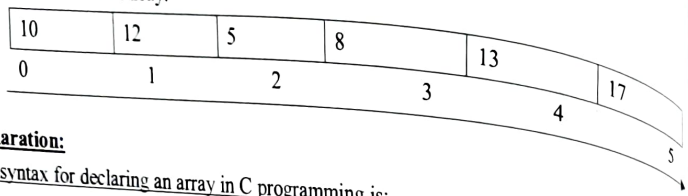
Array

An array is a collection of elements of the same data type, stored in contiguous memory locations.

a) Used to store multiple values in single variable
b) Elements of array stored in contiguous memory locations and lowest address corresponds to the first element and the highest address to the last element.
c) Can have one or more dimensions

Representation of Array:

Representation of an Array.



Declaration:

The syntax for declaring an array in C programming is:

data_type array_name[array_size];

For example, to declare an array of int values with the name arr and size 10, the syntax would be:

int arr[10];

The size of the array is specified within square brackets [] after the name of the array. The size must be a positive integer constant.

Arrays can also be initialized at the time of declaration, using the following syntax:

data_type array_name[array_size] = {value1, value2, value3, ...};

For example, to declare and initialize an array of int values with the name arr and size 5, the syntax would be:

int arr[5] = {1, 2, 3, 4, 5};

Types of 1-D and 2-D array:

1-dimensional array:

A 1-dimensional array is a collection of elements stored in contiguous memory locations. The syntax for declaring a 1-dimensional array is:

data_type array_name[array_size];

2-dimensional array:

A 2-dimensional array is an array of arrays, with each inner array representing a row of the 2-dimensional array. The syntax for declaring a 2-dimensional array is:

data_type array_name[row_size][column_size];

For example, to declare a 2-dimensional array of int values with the name arr and size 3x4, the syntax would be:

int arr[3][4];

Declaration

Datatype arrayName[arraySize]; Array Size=6

int publication[6];

size and type of array cannot be changed once it is declared.

initialization

can initialize array during declaration.

```
int publication[6] = {10,12,5,8,13,17} //1-d
also
int publication[] = {10,12,5,8,13,17}

int author[3][4]={{1,2,3},{4,5,6}} //2-d
```

loop through an array

```
int publication[] = {10, 12, 5, 8};
    int i;
    for (i = 0; i < 4; i++) {
     printf("%d\n", publication[i]);
    }
```

Advantages of Array

- Can sort data efficiently
- Ease of Sorting
- Ease of Traversing
- Random Access

Disadvantage

- Fixed size

Multi dimensional array

Array of Arrays is called multi dimensional array.

e.g.: Float Publication[4][5];

Here Publication is two dimensional array. It can hold 20 elements.

We can declare a 3-dimensional array

Eg: Float Publication[2][4][3]

Publication can hold 24 elements.

First Method: Int Author[3][4]={{1,2,3},{4,5,6}}

Second Method: Int Author {0,1,2,3,4,5,6,7,8,9,10,11}

The complexity increases as the number of dimension increases.

Difference between 1-D and 2-D Array

a) In programming, a 1-D array is a linear array that stores elements of the same data type in a contiguous memory location. It is also known as a vector or a one-dimensional array. The elements in a 1-D array can be accessed using a single index or subscript.

b) On the other hand, a 2-D array is an array of arrays, where each element is itself an array of the same data type. It is also known as a matrix or a two-dimensional array. The elements in a 2-D array are arranged in rows and columns and can be accessed using two indices or subscripts.

c) The main difference between 1-D and 2-D arrays is the way their elements are organized in memory. In a 1-D array, the elements are stored in a single contiguous block of

memory, while in a 2-D array, the elements are organized in rows and columns, with each row being stored in a separate block of memory.

String Manipulations

a) **String manipulation in C programming refers to the various operations that can be performed on strings, such as concatenating, comparing, copying, and formatting strings.**

b) Process of handling and Analyzing strings.

c) All string handling functions are defined in the header file called string.h

d) Major string handling functions are: strcat(), strlen(), strcpy(), strcmp() etc.

e) String manipulation can be used to clean and preprocess text data, extract relevant information, and transform data for analysis.

f) It's important to handle strings carefully to avoid errors such as buffer overflows or injection attacks.

3.2 POINTERS, STRUCTURE AND DATA FILES IN C PROGRAMMING

Pointer

- A variable which stores the address of another variable rather than values.
- Can be declared using * Symbol
- Also known as indirection pointer used to dereference a pointer.
- Pointers are commonly used to manipulate arrays, strings, and other complex data structures.
- Dereferencing a pointer means accessing the value stored at the memory location pointed to by the pointer, and can be done using the "*" symbol.
- Memory leaks can occur if pointers are not properly managed and deallocated when no longer needed.
- Syntax:

int* p; [declare a pointer p of int type]

Declaring Pointer

```
int *q1;
int *q2;
```

Or by int * q1, q2; here we have declared a pointer q1 and normal variable q2

Assigning addresses to pointers

```
int* AC, C;
C=5;
AC= &C;
```

Get value of thing pointed by pointers

```
int* AC, C;
C=5;
AC= &C;
printf("%d", *AC);
```

Change value pointed by pointers

```
int* AC, C;
C=5;
AC= &C;
printf("%d", C);
printf("%d", *AC);
```

Types of Pointers

Null Pointer

Create by assigning null value during pointer declaration

Null pointer is represented by the value 0 or by the macro NULL, which is defined in the stdio.h header file.

Null pointers can be used to represent situations where a pointer does not currently point to a valid object, or to initialize pointers to a default or sentinel value. They are also used in some standard library functions, such as calloc and realloc, to indicate that memory allocation has failed.

int *p = NULL;

Void pointer

A void pointer is a special type of pointer in C that can store the address of any object, but does not have a specific type associated with it.

Also called generic pointer

Does not contain standard data type

To use a void pointer, it must be cast to a specific pointer type before it can be dereferenced or used in pointer arithmetic

It can be used to bypass type checking and can lead to hard-to-debug errors if used incorrectly

Created by using void keyword

void *p = NULL;

Wild pointer

Wild pointers are also known as uninitialized pointers or dangling pointers, and can cause unpredictable behavior when dereferenced or used in pointer arithmetic.

Not being initialized to anything

Can point to unknown memory location, so that program can crash

int *p;

Other types of pointers exists in C like

- Near Pointer
- Huge Pointer
- Dangling Pointer
- Far Pointer
- Complex Pointer
- Pointer Arithmetic

Following are the few operations that are allowed to perform on pointers in C language.

1. Increment/Decrement
2. Addition of integer
3. Subtraction of integer
4. Subtracting two pointers
5. Comparison of pointers

Pointer and Array

Pointer:

Already Discussed.

Array

Already discussed.

Array of structures

Collection of multiple structures variables where each variable contains information about different entities.

Known as collection of structures.

There are two ways of declaring the structure variable with and without the structure declaration

Multiple values with a single variable

Structure student s1, s2, s3;

By using array of structure, we can write,

struct student s[3];

Passing Pointer to Function

Pointer can be passed as an argument to a function like other argument.

Declare the function parameter as a pointer type.

when we pass a pointer as an argument then the address of that variable is passed instead of the value

So if any change made to the pointer by the function is permanently made at the address of the passed variable

Also called as call by reference

Example of a function that takes a pointer to an integer as an argument, and modifies the value of the integer:

```
void increment(int *p) {
  (*p)++;
}
```

Declaration:

```
return_type function_name(int*);
```

If a function wants to accept an address of two integer variable then declaration will be

```
return_type function_name(int*, int*);
```


Structure and Union

Structure

To create a structure 'struct' keyword is used.

Used to represent a record

A structure can contain many different data types(int, float, char etc.)

Defining a structure:

```
Struct students {
char name[100];
char address[100]
char subject[100];
int std_id;
};
```

We can use this method if numbers of variable are not fixed

Structure as function arguments:

Same way as you pass variable or pointer.

Union

- Collection of variables of different datatypes in same memory location
- Can define with many members, only one member can contain a value at a given point of time.
- Used to save memory
- Allows data members which are mutually exclusive to share the same memory

Syntax:

```
Union Union_name{
    Datatype field_name;
    Datatype field_name;
    ...
};
```

Comparison of structure and union

Structure	Union
Keyword struct is used	Keyword union is used
Each member is assigned a unique storage area of location	Shared by individuals members
Individual member can be accessed at a time	Only one member can be accessed at a time
Several member can be initialized at once.	Only first member can be initialized

Passing structure to function

To pass a structure to a function, the function must be declared to take a pointer to the structure as an argument, and the address of the structure must be passed to the function.

Two ways

Passing all the elements to the function individually

Passing the entire structure to the function

- Call by reference
- Call by value

Call by value

value of the actual parameters is copied into the formal parameters

can not modify the value of the actual parameter by the formal parameter

Actual Parameter- used in the function call

Forma parameter- used in the function definition

Call by reference

The address of the variable is passed into the function call as the actual parameter

The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed

memory allocation is similar for both formal parameters and actual parameters

structure and pointer

The pointer which points to the address of the memory block that stores a structure

It tells the address of a structure in memory by pointing the variable to the structure variable

There are two ways to access the members of structure with the help of a structure pointer

1. With the help of (*) asterisk or indirection operator and (.) dot operator.
2. With the help of (->) Arrow operator

Input/Output operations on files.

File types: Text, Binary

Files are needed to preserve the data even if the program terminates.

use a structure pointer of file type to declare a file.

```
FILE *fp;
```

Following are the functions that helps to perform basic file operations.

fopen()- create a new file or open a existing file

syntax:

```
*fp = FILE *fopen(const char *filename, const char *mode);
```

fclose() – close a file

```
int fclose( FILE *fp);
```

fprintf() - writes a set of data to a file

fscanf() - reads a set of data from a file

getc() - reads a character from a file

getw() - reads a integer from a file

putw() - writes a integer to a file

rewind() - set the position to the begining point

ftell() - gives current position in the file

fseek() - set the position to desire point

Opening modes in I/O

r Open for reading

rb Open for reading in binary mode

w- Open for writing.

Wb Open for writing in binary mode.

a Open for append.

ab Open for append in binary mode

r+ Open for both reading and writing.

w+ Open for both reading and writing

a+ Open for both reading and appending

ab+ Open for both reading and appending in binary mode

Sequential and Random Access to a file

Data stored in a file can be accessed in two ways:-

Sequential Access

If file size is too big, sequential access is not best for reading the record in the middle of the file

Random Access

Enable access to any record at any point in the file

Enables to read or write any data in disk without reading or writing every piece of data

File Mode combinations allow us to simultaneously accomplish reading and writing operations

Three functions helps to random access the file

fseek()

ftell()

rewind()

3.3 C++ LANGUAGE CONSTRUCTS WITH OBJECTS AND CLASSES

Namespace

collection of related names or identifiers (functions, class, variables)

Help to separate identifiers from similar identifiers in other namespaces or global namespace

Begins with keyword namespace

std is a namespace whose members are used in the program

members of 'std' namespace are cout, cin, endl etc.

Overloading

Creation of two or more members having the same name but different in number or type of parameter, known as overloading

Two types-

Function overloading

Having two or more function with the same name but different in parameters.

Increases the readability of program because need not to use different names for the same action.

```
#include <iostream>
using namespace std;
class calculation {
  public:
static int add(int x,int y){
     return x + y;
  }
static int add(int x, int y, int z)
  {
     return x + y + z;
  }
};
int main(void) {
   calculation c;
   cout<<c.add(10, 20)<<endl;
   cout<<c.add(12, 20, 23);
  return 0;
}
```

Operator overloading

operator is overloaded to provide the special meaning to the user-defined data type

Adv: To perform different operations on the same operand.

Syntax:

```
return_type class_name : : operator op(argument_list)
{
   // body of the function.
}
```


Operators that cannot be overloaded are

scope operator (::)
sizeof
member selector(.)
member pointer selector(*)
ternary operator(?:)

Inline Functions

When we make the function inline, compiler places a copy of the code of that function at each point where the function is called at compile time

Saves memory space

If we make any changes to an inline function, that function need to be re compiled again.

Syntax: inline return_type function_name(parameters)

```
{
// function code?  }
```

Default Arguments

Used when we provide no arguments or few arguments while calling a function

It is that type of argument on which if the calling function does not pass any value to the arguments then the compiler assigned a value in the function declaration automatically.

We will not get compilation error, while assigning default value to an argument, the subsequent arguments must have default values assigned to them.

Introduction to Class and Object

Object

Real world entity.eg: car, pen, pencil etc.

It is and entity that has state and behavior. State means data and behavior means functionality.

Instance of Class

e.g.: Student s1; //creating an object of student

Class

User defined data type has data members and member functions

Group of similar objects

Blueprint for an object

No memory is allocated when a class is defined but memory is allocated when object is created.

e.g:

```
Class student
{
Public:
Int id;
String name;
String address;
}
```

Access Specifiers

Define how the members (attributes and methods) of a class can be accessed.

Used to implement data hiding

Types:

Private

members cannot be accessed from outside the class

Public

members are accessible from outside the class

Protected

members cannot be accessed from outside the class, however, they can be accessed in inherited classes

All members of class are private, if we don't specify any access specifier

Objects and the member Access

- It determines if a class member is accessible in an expression or declaration
- In Class the default access is private but in structure and union the access is public.

Defining Member Function

Member functions can be defined within the class definition or separately using scope resolution operator.

Scope resolution operator is used to qualify hidden names so that we can use them.

Types

- Simple Functions
- Const functions.
- Friend functions.
- Static functions.
- Inline functions.

Constructor and its types

Constructor

a) Special type of function with no return type
b) Constructor's name should be same as class name
c) Member functions that get invoked when an object of class is created
d) Can defined inside or outside the class definition

Types:

- Default constructor: doesn't take any arguments, use to initialize data members (variables) with real values
- Parameterized constructor: contains parameters (or arguments) in the constructor definition and declaration
- Copy constructor: is a member function that initializes an object using another object of the same class

Constructor Overloading:

The constructors with the same name and different parameters (or arguments)

Having more than one constructor with same name.

Destructor

Destructor destroys the class objects created by constructor

preceded by a tilde (~) symbol.

Not possible to define more than one destructor

Automatically called when object goes out of scope.

Dynamic Memory Allocation for objects and object array

Divided into two parts: The Stack and The Heap

Memory allocation can be done in two ways: Static allocation or compile-time allocation and Dynamic allocation or run-time allocation

Stack: All variables declared inside the function will take up memory from it.

Heap: Used to allocate the memory dynamically when program runs

New Operator: Denotes a request for memory allocation on the Free Store

Syntax: pointer-variable = **new** data-type;

e.g.: int *p = new int;

Delete Operator:

Syntax: delete pointer_variable_name

Note: C++ has new and delete for allocating dynamic memory but in c we use functions like malloc, calloc, realloc and free.

The main advantage of new over malloc() is that new doesn't just allocate memory, it constructs objects which is prime purpose of C++.

For Arrays:

Pointer Initialized with NULL

char* pvalue = NULL;

Request memory for the variable

pvalue = new char[20];

This Pointer

- Refers to the current instance of the class
- used to pass current object as a parameter to another method.
- used to refer current class instance variable.
- used to declare indexers.

Applications:

- Return Object
- Method Chaining
- Distinguish data members

Static Data Member and Static Function

Static Data Member

- Declared using static keywords
- visible only within the class
- only one copy of the static data member in the class, even if there are many class objects

Syntax: static data_type data_member_name;

Static Function

- Declared using static keyword
- Used to access only static data members
- Can be called even if no objects of the class exist.

Constant Member Functions and Constant Objects

Constant Member Functions

- Declared as constant in the program
- Constant member function can only read or retrieve the data members of the calling object without modifying them

Constant Functions:

- Specifies that function is a read-only function and does not modify the object for which it is called

Friend Function and Friend Classes

Friend Function

A function that can access private, protected and public members of a class.

Declared using the friend keyword inside the body of the class.

Friend functions in C++ have the following types

- Function with no argument and no return value
- Function with no argument but with return value
- Function with argument but no return value
- Function with argument and return value

Friend Classes

Can access private and protected members of other class in which it is declared as friend

Allowing to extend storage and access its part while maintaining encapsulation

3.4 FEATURES OF OBJECT-ORIENTED PROGRAMMING

Operator Overloading

Already Discussed

operator overloading can be done using three approaches

- Overloading unary operator.
  1. no arguments should be passed
  2. works only with one class objects
- Overloading binary operator.
  1. one argument to be passed
  2. overloading of an operator operating on two operands
- Overloading binary operator using a friend function
  1. friend operator function takes two parameters in a binary operator
  2. implemented outside of the class scope.

Type Conversion

Conversion of data of one type to another called as type conversion

Two types:

- Implicit Conversion
  Conversion that is done automatically by the compiler
  Also called Automatic Conversion
- Explicit Conversion
  When user manually changes data from one type to another
  Also called type casting

Three ways to use explicit conversion in C++

Cast Notation

Function Notation

Type Conversion operators

Inheritance

Process in which one object acquires all the properties and behaviors of its parent object automatically

Derived class is one which inherits the members of another class

Base Class is the class whose members are inherited

To inherit from a class, we use the: symbol.

The derived class is the specialized class for the base class.

Five types of inheritance is supported by C++

- Single inheritance
  Derived class is inherited from only one base class
- Multiple inheritance
  Deriving a new class that inherits the attributes from two or more classes
- Hierarchical inheritance
  Deriving more than one class from a base class
- Multilevel inheritance
  Process of deriving a class from another derived class
- Hybrid inheritance
  Combination of more than one type of inheritance

Constructor/Destructor in single/multilevel inheritances.

- Constructor is invoked implicitly when an object is created.
- When an object of derived class is created then constructor of derived class get executed and then it calls the constructor of base class
- If there is no default constructor present in parent class then not only we have to create constructor in child class but also we will have to call the constructor of parent class.
- The order of constructors calling is from child class to parent class
- The order of constructors execution is from parent class to child class
- The order of destructors calling is from child class to parent class
- The order of destructors execution is from child class to parent class

Encapsulation

Encapsulation is a programming concept that refers to the practice of bundling data and the methods that operate on that data within a single unit, such as a class in object-oriented programming. The data is kept hidden from the outside world and can only be accessed through the methods provided by the class. This technique is used to protect the data and ensure that it is only modified in controlled ways, which helps to prevent unintended errors and ensure the correctness and consistency of the code.

E.g.:

```
#include <iostream>
using namespace std;

class Person {
  private:
    string name;
    int age;

  public:
    void setName(string name) {
      this->name = name;
    }

    void setAge(int age) {
      this->age = age;
    }

    string getName() {
      return name;
    }

    int getAge() {
      return age;
    }
};

int main() {
  Person person;

  person.setName("RAM");
  person.setAge(15);

  cout << "Name: " << person.getName() << endl;
  cout << "Age: " << person.getAge() << endl;

  return 0;
}
```

Polymorphism

Polymorphism is a key concept in object-oriented programming that refers to the ability of objects of different classes to be used interchangeably.

Polymorphism enables different objects to be treated as if they are the same type, allowing for more flexible and extensible code. This is typically achieved through method overloading, method overriding, subtyping, or interface-based programming. Polymorphism allows for more modular and reusable code, as well as easier maintenance and updates.

Types of Polymorphism

1. **Compile-time polymorphism**
   - Static polymorphism or method overloading
   - resolved by the compiler at compile-time
   - It allows different methods to have the same name but different parameters, allowing the appropriate method to be selected based on the types and number of arguments passed to it.
2. **Runtime polymorphism**
   - dynamic polymorphism or method overriding
   - resolved at runtime
   - Runtime polymorphism allows objects of a subclass to be treated as objects of its superclass, and for methods to be called on those objects even if they have been overridden in the subclass.

3.5 PURE VIRTUAL FUNCTION AND FILE HANDLING

Virtual Function

It is a member function in the base class that we expect to redefine in a derived class.

Declared using virtual keyword

Cannot have a virtual constructor, but we can have a virtual destructor

It cannot be static members.

Syntax:

```
virtual<func_type><func_name>()
{
    // code
}
```

Pure Virtual Function

Also Known as do-nothing function

It is a function declared in the base class that has no definition relative to the base class

It can be defined as:

```
virtual void display() = 0;
```

Syntax:

```
virtual<func_type><func_name>() = 0;
```

Binding

Converting identifiers into addresses

linking function definition with the function call.

Takes place at compile time or run time

Types: Early binding and Late binding

Early Binding:

When compiler acknowledges all the information required to call a function or all the values of the variables during compile time

Late Binding:

Calling a function or assigning a value to a variable at runtime

Opening and Closing a file

Three types of stream:

Input

Syntax: ifstream fin;

Output

Syntax: ofstream fout;

Input/Output

Syntax: fstream fio;

Opening a file(two ways)

- Using the constructor function of the stream class

  ifstream fin("file", ios::in);

- Using the function open

Closing a file

It is closed by disconnecting it with the stream it is associated with.

stream_object.close();

Input / Output operations on files

Ofstream: Stream class to write on files

Ifstream: Stream class to read from files

Fstream: Stream class to both read and write from/to files.

To achieve file handling we need following steps.

a. Naming a file
b. Opening a file
c. Writing data into the file
d. Reading data from the file
e. Closing a file

Classes for file stream operations:

1. Ios
2. Istream
3. Ostream
4. Streambuf
5. Fstreambase
6. Ifstream
7. Ofstream
8. Fstream
9. Filebuf.

Error handling during input/output operations

Errors can occur during file operations with different reasons while working with files.

C++ provides built in functions to handle errors during file operations. They are:-

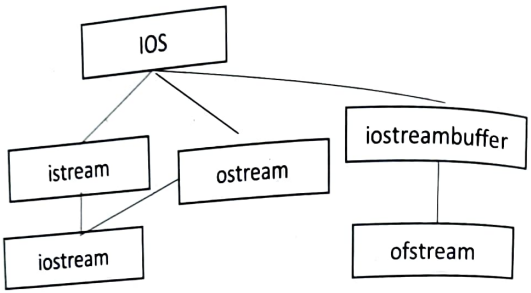
int bad(), int fail(), int good(), int eof().

Stream Class Hierarchy for Console Input /Output

Stream classes in C++ are used to input and output operations on files and io devices.

iostream.h library holds all the stream classes in the C++ programming language.

Hierarchy



IOS: base class for all stream classes

Istream: handles the input stream

Ostream: handles the output stream

Types of stream classes

- IOS
- Ostream
- istream_withassign
- iostream_withassign.
- Istream
- Iostream
- ostream_withassign

Unformatted Input /Output Formatted Input /Output with ios

Formatted Input/ Output:

These functions take various inputs from the users and also display multiple outputs to the users. Format specifiers can be used so it is called formatted I/O.

Examples: printf(), scanf(), sprintf(), sscanf()

Unformatted Input/ Output:

Read single input from the user at the console and it allows to display the value at the console.

Unformatted I/O are only used for character and string datatype but not for other datatypes.

Examples: getch(), getche(), getchar(), putchar(), gets(), puts(), putch()

In OOP, two ways to perform the formatted IO operations.

- Using the member functions of ios class.
- Using the special functions called manipulators defined in iomanip.h.

Formatted IO using ios class memebers:

The ios class contains several member functions that are used to perform formatted IO operations. The ios class also contains few format flags used to format the output. It has format flags like showpos, showbase, oct, hex, etc. The format flags are used by the function setf( ). [1]

functions of ios class used to perform formatted IO in C++. [2]

- width(): The width method is used to set the required field width. The output will be displayed in the given width

- precision(): The precision method is used to set the number of the decimal point to a float value
- fill(): The fill method is used to set a character to fill in the blank space of a field
- setf(): The setf method is used to set various flags for formatting output
- unsetf(): The unsetf method is used To remove the flag setting

Formatted IO using manipulators

The iomanip.h header file contains several special functions that are used to perform formmated IO operations.

Details of the special manipulator functions used to perform formatted IO in C++:

- setw(int): set the width in number of characters for the immediate output data.
- setfill(char): fill the blank spaces in output with given character.
- setprecision(int): set the number of digits of precision.
- setbase(int): set the number base.
- setiosflags(format flags): set the format flag.
- resetiosflags(format flags): clear the format flag.

The iomanip.h also contains the following format flags using in formatted IO in C++.

Flag	Description
endl	move the cursor position to a newline.
ends	print a blank space (null character).
dec	set the decimal flag.
oct	set the octal flag.
hex	set the hexadecimal flag.
left	set the left alignment flag.
right	set the right alignment flag.
showbase	set the showbase flag.
noshowbase	set the noshowbase flag.
showpos	set the showpos flag.
noshowpos	set the noshowpos flag.
showpoit	set the showpoit flag.
noshowpoint	set the noshowpoint flag.

Member functions and Flags

Formatting with Manipulators.

Manipulators: Manipulators are operators that are used to format the data display.

Types of Manipulators.

Two types one taking arguments and other without arguments

1. Manipulators with Arguments

Require iomanip header

Eg: setprecision, setw, setfill, setbase, setiosflags and resetiosflags

2. Manipulators without arguments

   Require iostream header

   Eg: edl, ws, ends, fixed, showpoint, left and flush

3.6 GENERIC PROGRAMMING AND EXCEPTION HANDLING

Template:

Define the generic classes and generic functions

Provides support for generic programming

Two Ways:

Function Templates

Class Templates

Function Templates

Special functions that can operate with generic types

Format for declaring function templates with type parameters

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

Both prototype use of either the keyword class or the typename and both expressions have the same meaning and behave exactly the same way.

Syntax: template < class Ttype> ret_type func_name(parameter_list)

```
{
  // body of function.
}
```

Class Templates

When a class uses the concept of Template, then the class is known as generic class.

```
template<class Ttype>
class class_name
{ ....
}
```

Creating a class template object

```
className<datatype> classObject;
```

Function Definition of Class Template

```
Template <class T>
Class ClassName {
...
returnType functionName();          //Function prototype
};
Template <class T>                  // Function Definition
returnType ClassName<T>::functionName() {
//Code

}
```

Overloading Function Template

Can be overloaded by a non-template function or by using ordinary function template

When the name of the function templates are same but called with different arguments known as overloading function template

Standard Template Library (Containers, Algorithms, Iterators)

It is a library of container classes, algorithms and iterators.

It has Four Components

- Algorithms

  Defines a collection of functions specially designed to be used on a range of elements.

- Containers

  It stores objects and data.

- Functions
- Iterators

Used to step through the elements of collections of objects.

Exception Handling Constructs

Exception is a problem that arises during the execution of a program.

Consists three keywords: try, throw and catch

Try: Represents a block of code that can throw an exception.

Throw: Throws an exception when a problem is detected, which lets us create a custom error.

Catch: It indicates the catching of an exception.

Syntax:

```
try {
    // Block of code to try
    throw exception;
  }
  catch () {
    // Block of code to handle errors
  }
```

Multiple Exception Handling

Multiple catch exception statements are used when a user wants to handle different exceptions differently.

Syntax:

```
Try {
  body
```

```
}

catch (type1 argument1)
{
  statement;
  ... ... ...
}

catch (type2 argument2)
{
  statement;
}
... ... ...
catch (typeN argumentN)
{
  statement;
}
```

Rethrowing Exception

We can rethrow the exception if a catch block cannot handle the particular exception it has caught

```
try {
  // Block of code to try
  throw exception; // Throw an exception when a problem arise
}
catch (exception& e) {
  // Block to do something with the error (e.g. log it).
  throw;    // rethrows the error.
}
```

Catching All Exceptions

To catch all exceptions we specify ellipses in the catch block.

it is difficult to know what error occurs in the program if all the exceptions are caught using catch(...).

Syntax:

```
try{
  // ...
} catch (const std::exception& ex) {
  // ...
```



```
} catch (const std::string& ex) {
  // ...
} catch (...) {
  // ...
}
```

Exceptions Specification for Function

A function is limited to throwing only a specified list of exceptions called as exception specification.

An exception specification at the beginning of any function acts as a guarantee to the function's caller that the function will throw only the exceptions contained in the exception specification.

Handling Uncaught and Unexpected Exceptions

If an exception is thrown and no exception handler is found means the exception is not caught then the program calls a termination function.

We can specify own termination function with set_terminate.

If a function specifies which exceptions it throws and it throws an unspecified exception, an unexpected function is called. We can specify own unexpected function with set_unexpected.

If we do not specify an unexpected function, the unexpected function is called.