28 MONDAY
Aug 2023 Week 35 / 240-125

AUGUST 2023

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

C++ Language preliminaries

2.1. Introduction

- Each lang. will have its own set of permissible characters.
- characters form the most basic building blocks of the language.
- Using only these characters, the language constructs are formed.

e.g. (i) English Language 52 symbols → 26 upper case, 26 lower case. } character set is called its alphabet.

(ii) Symbols of decimal no. system. 0 ..... 9.

So special symbols in programming.

. Dot	+ Plus sign
, Comma.	& Ampersand
; Semicolon	^ Caret
: Colon.	* Asterisk
? Question mark	- Minus sign.
' Single quote	+ Plus sign
" Double quote.	< Opening pointed bracket
! Exclamation symbol.	> Closing " "
| Vertical bar	(Opening parenthesis
/ Right slash.	) Closing parenthesis
\ Back slash.	[Opening square bracket
~ Tilde symbol.	] Closing square "
_ Underscore	{ Opening brace
$ Dollar sign.	} Closing brace
% Percentage symbol.	# Number sign
	= Equal symbol.

* - class of characters, which are not printed, but instead cause some action are commonly referred to as the control characters or white spaces.
- Represented by backslash followed by another character

Backspace — \b
Horizontal tab — \t
Vertical tab — \v
Carriage return — \r
New line — \n
Form feed — \f
Alert character — \a.

2.2. Tokens

- Tokens are the smallest elements of a programming language.
- It can be categorized as (i.e. supported in C++ language)
  - Identifiers.
  - Keywords
  - Constants
  - Operators.
  - Special characters.

2.2.1 Identifiers

→ The names given to the program elements such as variables, constants, functions, arrays, classes, etc. are known as identifiers.
In C++ each of the files, programs, classes, functions, objects, constants and variables have unique identifiers or names.
Here are some rules and conventions

1. Naming Rules:
   - Start with a letter (uppercase or lowercase), underscore (_) or a special character (in some cases)
   - Can be followed by letters, digits or underscores.
   - C++ is case-sensitive, so "myVar" and "myvar" are treated as different identifiers.

2. Reserved Words:
   You can't use C++ keywords as identifiers.
   For E.g., "int", "class", "if", etc., are reserved words.

3. Length
   The length of an identifier varies by compiler but is typically limited to a few hundred characters.

4. Naming conventions
   - It's a good practice to use meaningful names to improve code readability
   - Variables and functions often follow a "camelCase" or "snake_case" convention, where words are separated by capitalization or underscores, respectively.
   - Class name typically use "CamelCase" with the first letter of each word capitalized.

Here are some examples:

```
int myVariable;          // Variable with a meaningful name
void calculateValue();   // Function with a meaningful name
class MyClass { };       // class with a meaningful name

// Remember to choose descriptive identifiers to make your code more understandable and maintainable.
```

2.2.2 Keywords

- C++ has standard identifiers called keywords
- Some commonly used C++ keywords are as follows :-

08.00

09.00

10.00

11.00

12.00

01.00

02.00

03.00

04.00

05.00

06.00

EVE

(1) Fundamental Types:
int, double: (double float), char, bool, void (represents the absence of type

(2) Control Flow:
if, else, for, while, do, switch

(3) Functions:
return: Used to return a value from a function
void: Indicates a function that doesn't return a value

(4) Classes and Objects.
- class: keyword to define a class
- struct: keyword to define a structure
public, private, protected: Access specifiers for class members.

(5) Memory mngmt.
- new: Allocates m/m on the heap
- delete: Deallocates m/m on the heap.

(6) Exceptions
try: Marks a block of code where exceptions might be thrown
catch: Catches and handle exceptions
throw: Throws an exception.

(7) Pointers and References
*: Pointer declaration and dereferencing
&: Reference declaration and address-of operator.

(8) Miscellaneous:
- const: specifies that a variable's value can't be changed.
- static: specifies static storage duration for variables.

Notes

- namespace : Declares a namespace
- typename : Used to declare type-dependent names in templates.
- template : Used to define templates for generic programming.

2.2.3. Constants

- The value of a constant that does not change during the execution of the program enclosing it.
- Depending on the types of data they represent, constants are broadly classified into three classes.

2.2.3.1 Numerical constants.

- It use to represent fixed value or literals.
- Some common numerical constants include

(1) Integer Constants:
Ex→ int myInteger = 42;

(2) Floating-Point constants:
Ex→ float myFloat = 3.14

(3) Double Precision Constants:
Ex→ double myDouble = 2.71828

(4) Character Constants
Ex→ char myChar = 'A';

(5) Boolean Constants
Ex→ bool myBool = true;

(6) Hexadecimal Constants:
Ex→ int hexValue = 0x1A; (Hexadecimal values start with "0x" or "0X")

(7) Octal Constants:
Ex→ int octValue = 075;
(Octal values start with "0")

(8) Binary Constants (C++14 and later):
Ex→ int binaryValue = 0b1010; (Binary values start with "0b")

2.2.3.2 Real Constants

08.00 The real constants, also known as floating point constants

09.00 (or) 2 main types of real constants in C++:

1. Float: Float constants are represented with the float data and 'F' or 'f' suffix
   → float myFloat = 3.14f. (containing 1 digit & 1 decimal point)

11.00

2. Double: representation → 'double' data type
   by default real numbers without a suffix are treated as 'double' constants.
   For ex→ double myDouble = 3.14
   we can also use scientific notation
   double scientificNotation = 1.23e4

12.00 01.00 02.00 03.00 [Exponential → $23 * 10^8$ → 0.23E8 or 0.23e8] // 1.23×10^4.

~~2.2.3.3 Character Constants~~

04.00

2.2.3.3. Character Constants

05.00

In C++ programming, character constants, or character literals, are single characters enclosed within single quotation marks (' ')

06.00 Sunday 03

E.g→ (1) 'A' : represents the character 'A'
(2) '1' ; " " " '1'
(3) '\n' ; " " newline character
(4) '%' ; " " percentage symbol
(5) 'x' ; " " lowercase letter 'x'

String constants: consists of one or more characters and it is enclosed within a pair of double quotes

E.g. => "abc"
"126n"
"123" and "1" are also string constants.

2.2.4. Operators in C++.

-> used to perform different operations on data.

Here's an overview of most common operators in C++.

Fundamental operators in C++.

(1) Arithmetic Operators:
- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)

(2) Relational Operators:
- Equal to (==)
- Not equal to (!=)
- Greater than (>)
- Less than (<)
- Greater than or equal to (>=)
- less than or equal to (<=)

(3) Logical Operators:
- Logical AND (&&)
- Logical OR (||)
- Logical NOT (!)

(4) Assignment Operators:
- Assignment (=)
- Addition assignment (+=)
- Subtraction assignment (-=)
- Multiplication assignment (*=)
- Division assignment (/=)

- Modulus assignment (%=)

5. Increment and Decrement Operators:
   - Increment (++)
   - Decrement (--)

6. Bitwise Operators:
   - Bitwise AND (&)
   - Bitwise OR (|)
   - Bitwise XOR (^)
   - Bitwise NOT (~)
   - Left shift (<<)
   - Right shift (>>)

7. Conditional (Ternary) Operator:
   - (conditional expression) true expression: false expression.

8. Member Access Operators:
   - Dot operator (.)
   - Arrow operator (->)

9. Sizeof Operator:
   - Used to determine the size of a data type or object.

10. Comma Operator:
   - Used to separate expressions and evaluate them from left to right.

EVE 2.25

Special characters

- For formatting o/p, the C++ compiler recognizes some commonly used special characters.
- These characters can be used by typing the backslash (called the escape character), followed by the special character.

Eg:

Notes

\a bell (alert
\b backspace
\f form feed
\n New line.
\r Carriage return
\t Tab
\v Vertical tab
\' Single quote
\" Double quote
\? Question mark
\\ Backslash.
\000 Octal notation
\xhhh. Hexadecimal notation.

2.3 Variables

- Variable is a named memory location, the variable of which can change during the execution of the program containing it.
- Naming of a variable can reflect the purpose of variable, it is only to increase the readability of the variables.

Exp, int a;
data type variable 1, variable 2, ---
Ex int a, b, c.

2.4 Data Type. in c++.

Notes: Programmer can select appropriate data types, to the needs of application.

07 THURSDAY

Sep 2023 Week 38 / 251 115

SEPTEMBER 2023

Name	Cycle	Description	Range
char	1	character or integer 8 bit length	signed: -128 to 127 unsigned 0 to 255
08.00 short	2	integer 16 bit length	signed: -32768 to 32767 unsigned: 0 to 65535
09.00 long	4	" 32 " "	signed: -2147483648 to 2147483648 unsigned: 0 to 4294967295
10.00			
11.00 float	4	floating point number	$3.4e^{-38}$ to $3.4e^{+38}$
12.00 double	8	double precision floating point number	$1.7e^{-308}$ to $1.7e^{+308}$
01.00 long double	10	long double precision floating point number	$1.2e^{-4932}$ to $1.2e^{+4932}$
02.00			

03.00 (1) Integer types.

int: 32 bit

04.00 short: 16 bit

long: 32 bit or 64-bit

05.00 long long: 64 bit.

(2) Floating point types

06.00 float: A single precision floating point number

double: A double " " " "

EVE long double: An extended precision " " "

(3) Character types:

char: represents a single character

wchar_t: " " wide "

Notes char16_t and char32_t: For unicode characters

(4) Boolean type:

bool: represents true or false values

5. Enumeration Types:
- enum: User-defined integer types with named values.

08.00 6. Void Types:
- void: represents the absence of a type.

09.00 7. User-Defined types
- struct: defines a composite data types with members.
- class: defines a composite data type with encapsulation
- union: defines a data type that can hold different data types in the same memory location.
- typedef: Alias for existing data types
  (nickname of something)
- enum class: Scoped enumerations for better type safety.

Type compatability

- In C++, type compatability refers to whether diff data types can be used together in various operations or assignments without causing errors or requiring explicit type conversions.

1. Implicit type conversion (Type coersion)

→ C++ allows some level of automatic type conversion between compatible types.

For eg., if you assign an integer to a float, C++ will perform an implicit conversion.

Ex: int myInt = 5;
float myFloat = myInt;
(result value = 5.0000)

Notes

OCTOBER NOVEMBER DECEMBER

2. Type Compatability in Expressions:
- In expressions, C++ follows a set of rules for type promotion and conversion to determine the resulting type.
E.g → int a = 5; double b = 2.5
double result = a + b;
- Here, a is promoted to a double before the addition operation.

(int - 2@ (can be put into float type))

3. Explicit type conversion:

You can use casting operators like static_cast, dynamic_cast, const_cast and reinterpret_cast to explicitly convert one type to another when needed.
Ex→ double myDouble = 3.14;
int myInt = static_cast<int>(myDouble);

Ex→ double a = 3.14
~~int b = int(a)~~ ; ✓ int b = (int) a

~~double~~ int b = a // this will give error.

Sunday 10

EVE

Notes

2023 OCTOBER

S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28
29	30	31											

Operator Precedence

MONDAY 11

Week 37 / 254-111 Sep 2023

FUNCTIONS

08.00 → A function groups a number of program statements into a unit and gives it a name. This unit can
09.00 then be invoked from other parts of the program.

10.00 → In simple terms, a function is a block of code that runs only when it is called
11.00

Example:- C++ program to demonstrate working of
12.00 a function.

01.00
```
#include <iostream>
using namespace std;
```
02.00
```
// Following function that takes two parameters 'x' and 'y'
// as input and returns max of two input numbers
```
03.00
```
int max (int x, int y)
{
```
04.00
```
    if (x>y)
        return x;
```
05.00
```
    else
        return y;
```
06.00
```
}
```

Time complexity: O(1)
Space: O(1)

```
// main function that doesn't receive any parameter
// and returns integer
int main()
{
    int a = 10, b = 20;
    // Calling above function to find max of 'a' & 'b'
    int m = max(a, b);
    cout << "m is" << m;
    return 0;
}
```

o/p: m is 20

Good things come to those who wait

OCTOBER NOVEMBER DECEMBER

12 TUESDAY

Sep 2023 Week 37 / 255-110

SEPTEMBER 2023

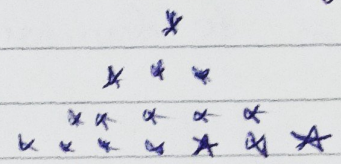
S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

Syllabus for function

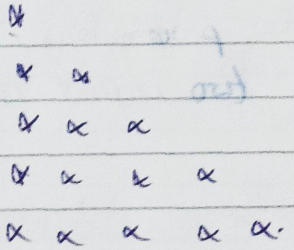
The Main Function, Function Prototyping, Call by Reference, Return by Reference, Inline Functions, Function Overloading, Friend Functions, default parameter value.

Today's Practice Program

(1) Program to print a full pyramid using *.



(2) WAP to print a half pyramid using *.



(3) WAP to add two complex numbers.

(4) WAP to find the size of fundamental data types.

(5) WAP using functions that takes two parameters 'x' & 'y' as input and returns maximum of two input numbers.

Notes

S3 - DLCD - 1:20pm
NEHA (ECE Dept.)

Friend function

08.00 → Data hiding is a fundamental concept of OOPs. It restricts the access of private members from outside of the class.

09.00 →

10.00 → Similarly, protected members can only be accessed by derived classes and are inaccessible from outside

11.00

For ex →

private
public
protected

12.00

```
class MyClass {
private:
    int member1;
};

int main() {
    MyClass obj;
    // Error! Cannot access private members from here.
    obj.member1 = 5;
}
```

01.00

02.00

03.00

04.00

05.00

06.00

EVE → However, there is a feature in C++ called friend functions that break this rule and allow us to access member functions from outside the class.

Similarly, there is a friend class which we'll learn later.

Notes

friend Function in C++

A friend function can access the private and protected data of a class. We declare a friend function using the "friend" keyword inside the body of the class.

Syntax →

```
class className {
    ... ... ...
    friend returnType functionName (arguments);
    ... ... ...
}
```

Ex-1 : Working of friend function

C++ program to demonstrate the working of friend function.

```
#include <iostream>
using namespace std;
class Distance{
    private:
        int meter;
    //friend function
    friend int addFive (Distance);
    public:
        Distance() : meter(0) { }
};
```

int add.
Void main()
{
}
Syntax
void

```
//friend function definition
int addFive(Distance d) {
//accessing private members from the friend function
    d.meter += 5;
    return d.meter;
}
int main() {
    Distance D;
    cout << "Distance: " << addFive(D);
    return 0;
}
```

Output :-> Distance: 5

Here, addFive() is a friend Function that can access both private and public data members.

-> Though Ex-1 gives us an idea about the concepts of a friend function, it doesn't show any meaningful use.

-> A more meaningful use would be operating on objects of two different classes. That's when the friend function can be very helpful.

Notes

Ex-2 Add members of Two different Classes

08.00
```
// Add members of two different classes using
   friend functions
#include<iostream>
using namespace std;

// forward declaration
class ClassB;

class ClassA {
public:
    // constructor to initialize numA to 12
  ClassA() : numA(12) { }

private:
    int numA;

    // friend function declaration
    friend int add(ClassA, ClassB);
};

class ClassB {
public:
    // constructor to initialize numB to 1
    ClassB() : numB(1) { }

private:
    int numB;
```

09.00
10.00
11.00
12.00
01.00
02.00
03.00
04.00
05.00
06.00
07.00
08.00

Sunday 17

Notes

```
// friend function declaration
friend int add(ClassA, ClassB);
};

// access members of both classes
int add(ClassA objectA, ClassB objectB) {
    return (objectA.numA + objectB.numB);
}

int main() {
    ClassA objectA;
    ClassB objectB;
    cout << "Sum: " << add(objectA, objectB);
    return 0;
}
```

O/p : Sum 13.

In this program, [ClassA] and [ClassB] have declared add() as a friend function. Thus, this function can access private data of both classes.

One thing to notice here is the friend function inside [ClassA] is using the [ClassB]. However, we haven't defined [ClassB] at this point.

```
// inside classA
friend int add (ClassA, ClassB)
```

For this to work, we need a forward declaration of [ClassB] in our program.

```
// forward declaration
class ClassB;
```