C++(OOPs Concept)

MRS. D.SELVI

Procedure-Oriented Language

- Used in C,COBOL,FORTRAN(high level language)
- Emphasis on doing algorithms
- Larger program is divided into smaller program called functions
- Most of the function share global data
- Data moves openly around the system from function to function
- Employs top-down approach

Object-Oriented Programming

- C++,Smalltalk , ObjectPascal, Java use these features
- Emphasis on data rather than procedure
- Programs are divided into objects
- Data is hidden and cannot be accessed by external functions
- Objects communicate with each other through function
- New data and functions can be easily added whenever necessary
- Follows bottom-up approach

What is C++

- C++ is an Enhanced version of c Language which is developed by Bjarne Stroustrup in 1980 in AT & T's Bell Lab.
- C++ Inherits many features from C Language and it also Some More Features and This Makes C++ an OOP Language.
- C++ Provides Reusability of Code for Another user.

Characteristics of OOPs

- Object based features
- Inheritance
- Dynamic binding

Features of OOPs

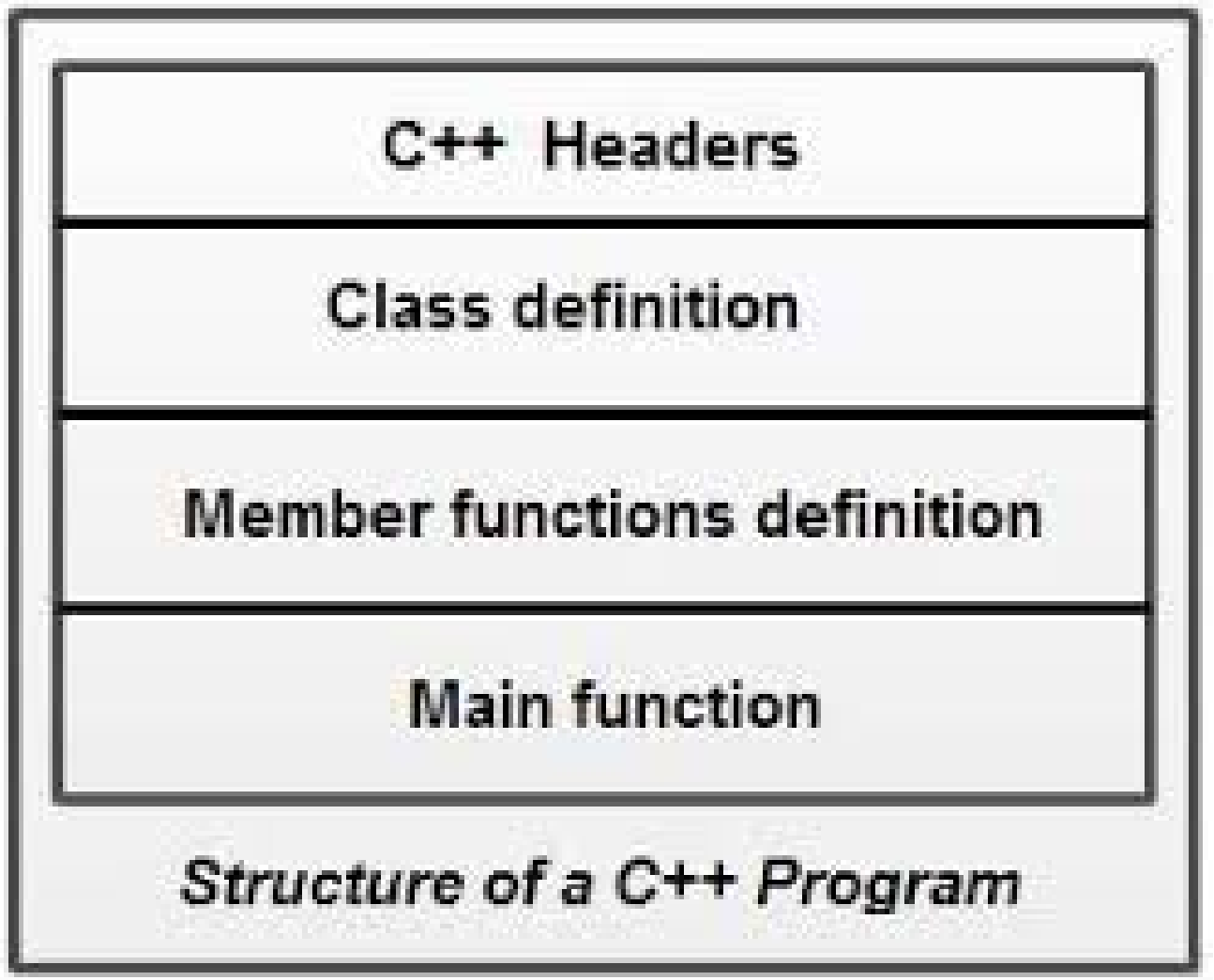
- Data encapsulation
- Data hiding and access mechanism
- Automatic initialization and clear up of objects
- Operator overloading

Application of OOPs:

- Real time sytems
- Simulation and modelling
- Object oriented databases
- AI and expert system
- CIM/CAD system
- Neural networking
- Hypertext,Hypermedia

Structure of a C+ + Program

- Programs are a sequence of instructions or statements. These statements form the structure of a C++ program. C++ program structure is divided into various sections, namely, *headers, class definition, member functions definitions* and *main function.*
- C++ provides the flexibility of writing a program with or without a class and its member functions definitions.
- A simple C++ program (without a class) includes comments, headers, namespace, main() and input/output statements.



Structure of a C++ Program

Comments

- **Comments** are a vital element of a program that is used to increase the readability of a program and to describe its functioning. Comments are not executable statements and hence, do not increase the size of a file.

- Single line command:

  // An example to demonstrate

  // single line comment

- Multi line command

  /* An example to demonstrate

  multiline comment */

Input/Output Operator in C++

- The operator used for taking the input is known as the **extraction** or get **from operator** (>>)
- while the operator used for displaying the output is known as the **insertion** or **put to operator** (<<)

Cascading of Input/Output Operators

- The cascading of the input and output operators refers to the consecutive occurrence of input or output operators in a single statement.

```
#include<iostream>
using namespace std;
int main ()
{
int a, b;
cin>>a;
cin>>b;
cout<<"The value of a is
cout<<a;
cout<<"The value of b is
cout<<b;
return 0;
}
```

```
#include<iostream>
using namespace std;
int main ()
{
int a, b;
cin>>a>>b;
Cout<<"The value of b is : "<<b;
cout<<"The value of a is "<<a;
return 0;
}
```

Variables

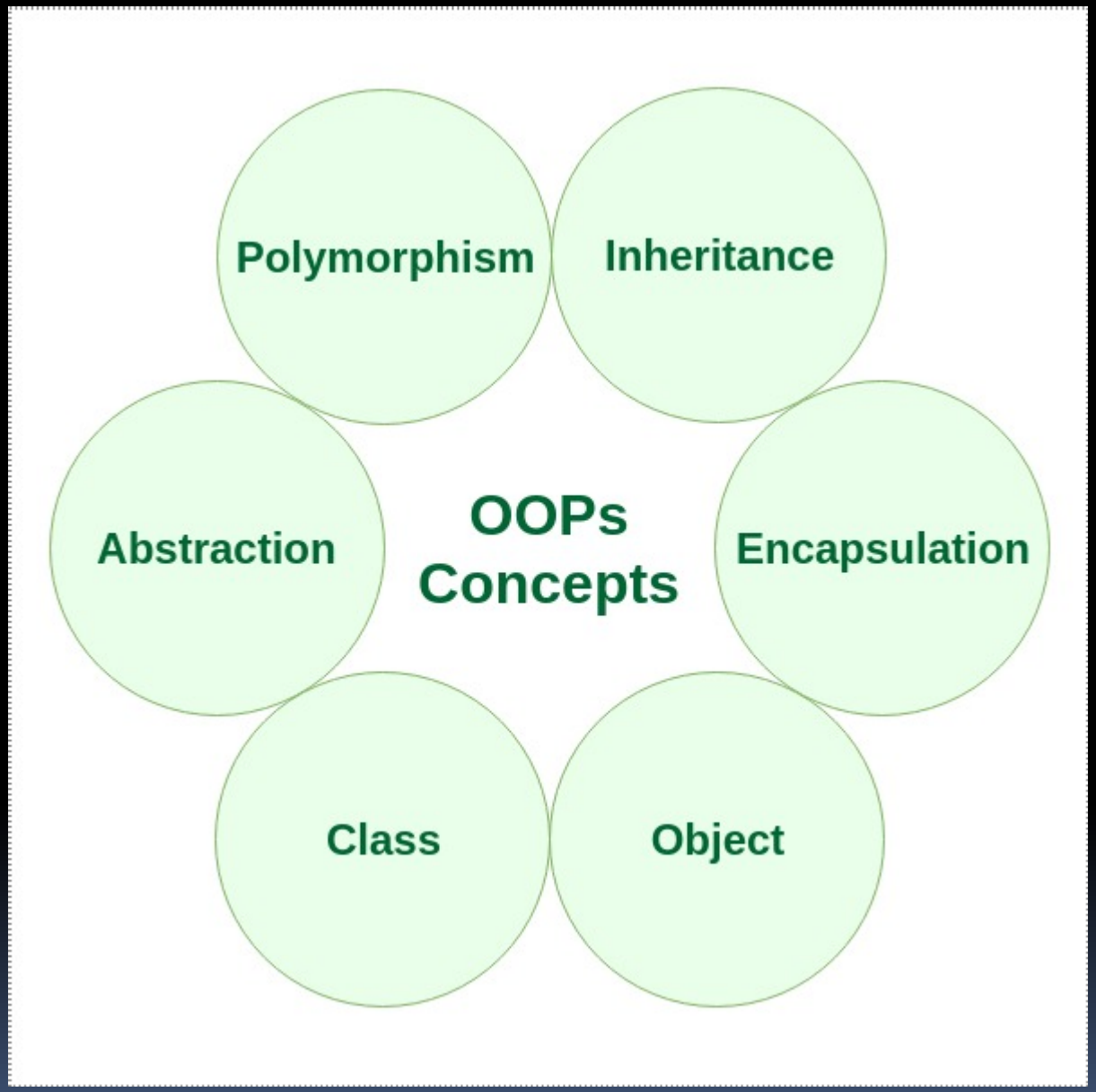
- **Definition:** "Variables are those quantities whose value can vary during the execution of the program"
- SYNTAX: **data_type variable_name;**
- **EG:**

```
int x, y, z;
```

Namespace

- One of the new features added to this language is namespace.
- A namespace permits grouping of various entities like classes, objects, functions and various C++ tokens, etc., under a single name.

all the modern C++ compilers support these statements.	old compilers may not support these statements
#include<iostream> **using namespace std;**	**#include<iostream.h>**


Polymorphism
Inheritance
Abstraction
OOPs Concepts
Encapsulation
Class
Object

Class

- It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.
- A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

keyword user-defined name

```
class ClassName

{  Access specifier:          //can be private,public or protected

   Data members;              // Variables to be used

   Member Functions() { }     //Methods to access data members

};                            // Class name ends with a semicolon
```

OBJECT

- An Object is an real time entity with some characteristics and behaviour.
- An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

- Syntax:

  ClassName ObjectName;

OBJECT:STUDENT

DATA:

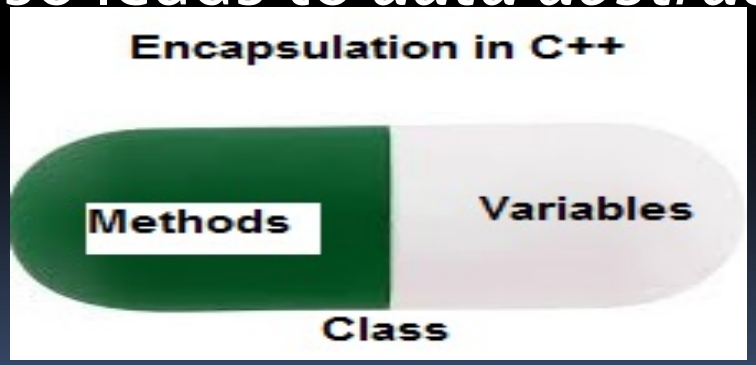
Name
D.O.B
Marks
.....

FUNCTIONS:

Total
Average
Display
.....

Encapsulation and Abstraction

- Encapsulation is defined as wrapping up of data and information under a single unit.
- Encapsulation is defined as binding together the data and the functions that manipulate them.
- Encapsulation also leads to *data abstraction or hiding*.



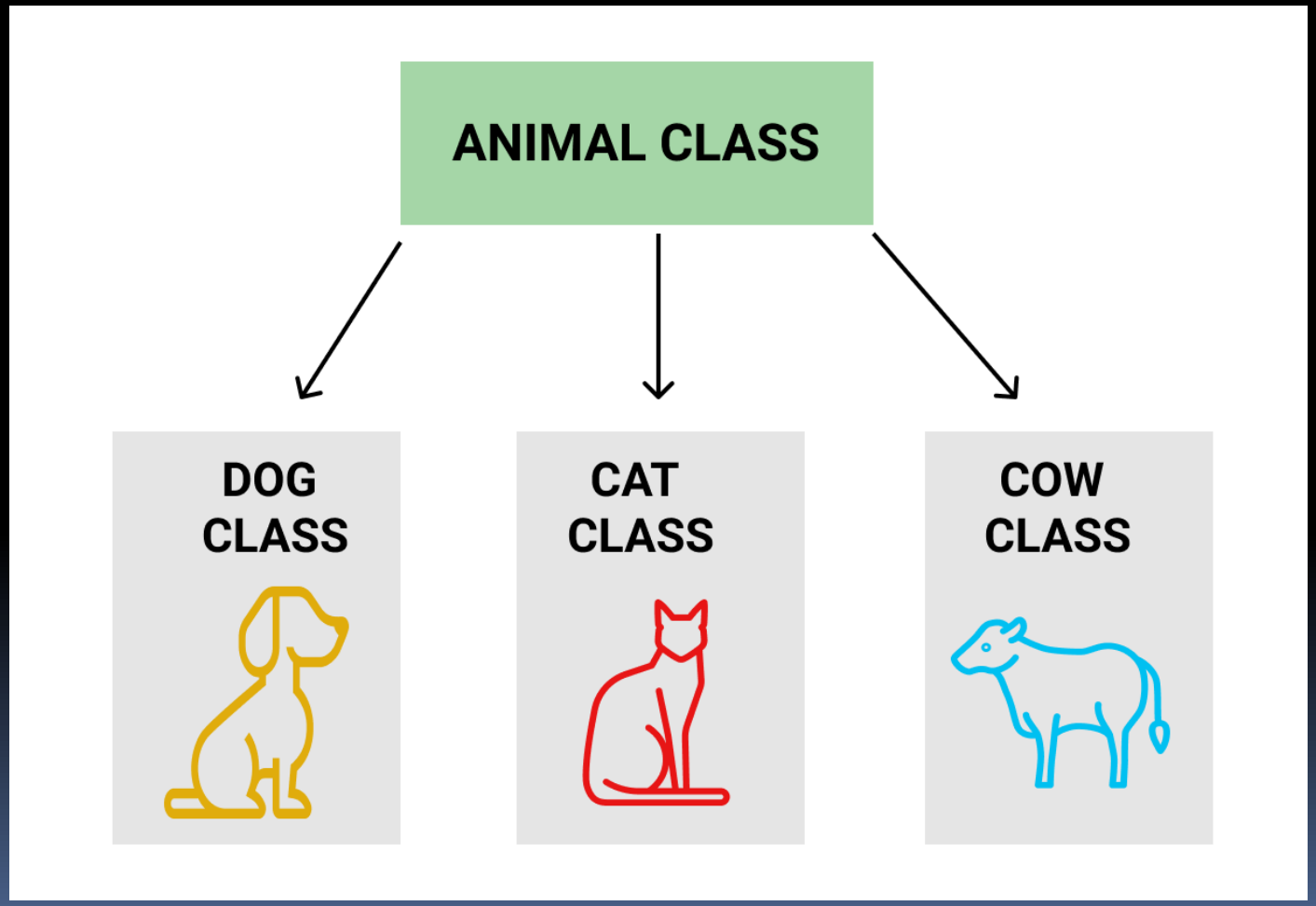
- Abstraction means displaying only essential information and hiding the details.
- Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
- *Abstraction using Classes*: We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.

INHERITANCE

- The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.

- **Sub Class**: The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class**:The class whose properties are inherited by sub class is called Base Class or Super class.
- **Reusability**: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example: Dog, Cat, Cow can be Derived Class of Animal Base Class.



POLYMORPHISM

- The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.
- An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

- C++ supports operator overloading and function overloading.
- *Operator Overloading*: The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.
- *Function Overloading*: Function overloading is using a single function name to perform different types of tasks. Polymorphism is extensively used in implementing inheritance.

```
int main( )
{
    sum1 = sum(20,30);
    sum2 = sum(20,30,40);
}
```

```
int sum(int a,int b)
{
    return (a+b);
}
```

```
int sum(int a,int b,int c)
{
    return (a+b+c);
}
```

Message Passing

- **Message Passing:** Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Steps for message passing:

- Create classes that define objects and their behaviour
- Creating objects from class definition
- Establishing communication among objects

Benefits of OOPs

- Eliminate redundant code and extend the use of existing code using inheritance
- Building secure programs by the principle of data hiding
- Building secure programs from working modules that communicate with each other. Its saves development time and leads to higher productivity.
- Mapping objects in the problem domain to those in program
- Easy to partition work in projects based on objects
- Software complexity can be easily managed

- Can be upgraded from smaller to larger systems
- Captures more details of model in implementable form
- Message passing technique makes interface description with external systems much simpler
- Multiple instance of object can co-exist without interferance

Access Modifiers in C++

- Access modifiers are used to implement an important feature of Object-Oriented Programming known as **Data Hiding**.
- There are 3 types of access modifiers available in C++:

  Public

  Private

  Protected

- **Public**: All the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

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

// class definition
class Circle
{
   public:
      double radius;

      double  compute_area()
      {
         return 3.14*radius*radius;
      }

};
```

- **Private**: The class members declared as *private* can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.

- **Protected**: Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass(derived class) of that class.

C++ Tokens

- A token is the smallest individual unit of a program that is meaningful to the compiler. Tokens can be classified as follows:
    - Keywords
    - Identifiers
    - Constants
    - Strings
    - Special Symbols
    - Operators

Keyword

- Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program.

C language supports **32** keywords which are given below:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

While in **C++** there are **31** additional keywords other than **C** Keywords they are:

- asm bool catch class
- const_cast delete dynamic_cast explicit
- export false friend inline
- mutable namespace new operator
- private protected public reinterpret_cast
- static_cast template this throw
- true try typeid typename
- using virtual wchar_t

Identifiers

- These are user defined names consisting of arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character.
- Name cannot start with a digit
- A declared keyword cannot be used as a variable name
- Uppercase and lowercase letters are distict
- They must consist of only letters, digits, or underscore

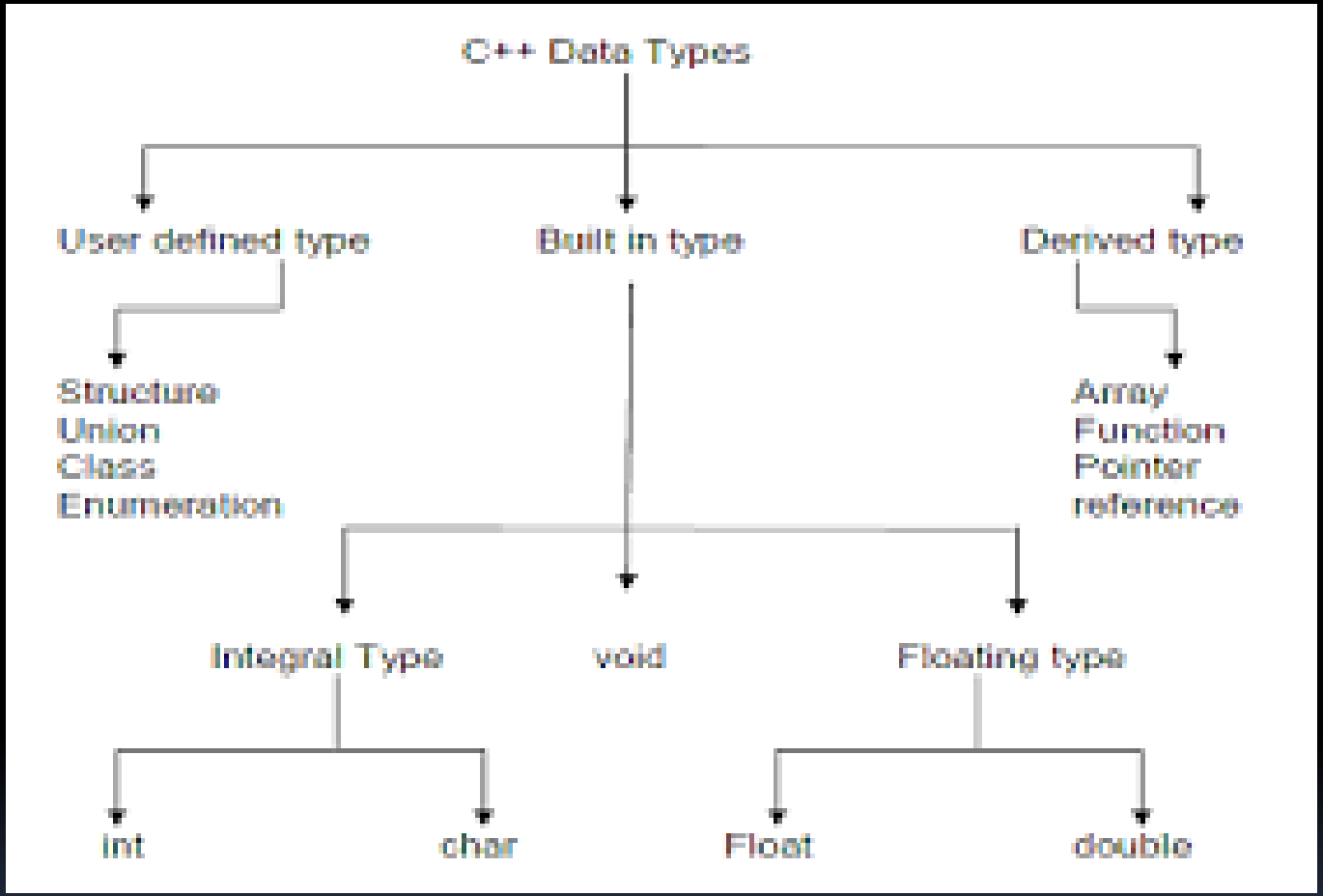
Constants

- Constants refer to fixed values. They are also called as literals.
- Their value do not change during the execution of program
- **Syntax:**

const data_type variable_name; (or) **const data_type *variable_name;**

Types of Constants:

- Integer constants
- Real or Floating point constants
- Octal & Hexadecimal constants
- Character constants
- String constants


C++ Data Types
User defined type
Built in type
Derived type
Structure
Union
Class
Enumeration
Array
Function
Pointer
reference
Integral Type
void
Floating type
int
char
Float
double

- **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Valueless or Void
- Wide Character

- **Abstract or User-Defined Data Types**: These data types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:Class
- Structure
- Union
- Enumeration
- Typedef defined DataType

Structures and Classes

- User defined data types such as struct and union in c
- C++ permits to define another user defined data type - class
- Class variables are known as objects

Enumerated Data Types

Attaching names to number

enum

enum shape{circle, square, triangle};

enum colour{red, blue, green, yellow};

enum position{on,off};

new type names

IN C

enums to be ints

IN C++

Does not permit an int value to be automatically converted to an enum value

EX:

```
colour background = blue;
colour background = 7;
colour backround = (colour) 7;
```

An enumerated value can be used in place of an int value

```
int c = red; // color type promoted to int
```

By default, the enumerators are assigned integer values stating with 0 for the first enumerator, 1 for the second ,.....

```
enum colour{red, blue, green}
enum colour{red, blue = 7, green = 9}
enum colour{red = 5, blue, green}
```

Derived Data Types

- The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:Function
- Array
- Pointer
- Reference

Arrays

Array size is the exact length of the string constant

ex: char string [3] = “zxc”;

the size should be one larger than the number of characters in the string.

Ex : char string [3] = “zxc”;

char string [4] = “zxc”;

Symbolic constants

Two ways:

- Using the qualifier constant
- Defining a set of integer constants using enum keyword.

In C and C++

const – constant expression

```
const int size = 10;  ( const size =10;)
 char name [ size];
```

The named constants are just like variables except that their values cannot be changed.

Type Compatibility

sizeof ('x') = sizeof (int)

Declaration of variables

```
int main()
{
    float x;     //declaration
     float sum = 0;
     for{int l =1;i<5;i++}     // declaration
      {
          ----
         ------
      }
    float average;     //declaration
      {
         -----
         ----
      }
```

Reference Variable & Call by reference

New kind of variable – reference

Data –type & reference – name = variable – name

float sum = total;

```
void   f(int & x)    // reference
 {
   X = x+10;
 }
 int main ()
{
 int m =10;
f(m);          //function call
```

Operators in C++

operators	name	Functions
::	Scope resolution operator	Same variable name can be used to have different meanings in different blocks
::*	Pointer – to – member declarator	To declare a pointer to a member of a class
->	Pointer – to – member operator	To access a member

What is Expressions in C++?

- A combination of variables, constants and operators that represents a computation forms an expression.
- These categories of an expression are discussed here.

  Constant expressions

  Integral expressions

  Float expressions

  Relational or Boolean expressions

  Logical expressions

  Bitwise expressions

  Pointer expressions

Constant expressions

- The expressions that comprise only constant values are called constant expressions.
- 20
- ‘a’
- 20+9/2.0

Integral expressions

- The expressions that produce an integer value as output after performing all types of conversions are called **integral expressions.**
- For example

  x

  6*x-y

  10 +int (5.0) are integral expressions

Here, x and y are variables of type into

Float expressions

- The expressions that produce floating-point value as output after performing all types of conversions are called **float expressions.**
- For example

9.25

x-y

9+ float (7) are float expressions

Here, x 'and yare variables of type float.

Relational or Boolean expressions

- The expressions that produce a bool type value, that is, either true or false are called **relational or Boolean expressions.**
- For example

  x + y<100

  m + n==a-b

  a>=b + c are relational expressions.

Logical expressions

- The expressions that produce a bool type value after combining two or more relational expressions are called **logical expressions**
- For example,

  x==5 &&m==5

  y>x || m<=n are logical expressions.

Pointer expressions

- The expressions that give address values as output are called **pointer expressions.**
- For example

  &x

  ptr

  ptr+1 are pointer expressions.

Here, x is a variable of any type and ptr is a pointer.

Bitwise expressions

- The expressions which manipulate data at bit level are called **bitwise expressions.**
- For example

  a >> 4

  b<< 2 are bitwise expressions.

Special assignment expressions

- Chained assignment
- Embedded assignment

Chained assignment

- **Chained assignment** is an assignment expression in which the same value is assigned to more than one variable, using a single statement. For example, consider these statements.

  a = (b=20); or

  a=b=20;

- For example, consider these statements.

  int a=b=30; // illegal

  int a=30, int b=30; //valid

Embedded assignment

- **Embedded assignment** is an assignment expression, which is enclosed within another assignment expression. For example, consider this statement

```
a=20+(b=30);   //equivalent to b=30; a=20+30;
```

Compound Assignment

- **Compound Assignment** is an assignment expression, which uses a compound assignment operator that is a combination of the assignment operator with a binary arithmetic operator. For example, consider this statement.

```
a + =20; //equivalent to a=a+20;
```

The general form is:

```
var1 op=var2; //equivalent to var1=var1 op var2
```

Where op=binary arithmetic operator

Implicit Conversions

- Implicit conversion, also known as automatic type conversion refers to the type conversion that is automatically performed by the compiler.
- For example,

in expression 5 + 4.25, the compiler converts the int into float as float is larger than int and then performs the addition.

Order of Data Types

Data Types
char short int
int
unsigned
long int
unsigned int
float
double
long double

smallest

largest

Typecasting

- Typecasting refers to the type conversion that is performed explicitly using type cast operator. In C++, typecasting can be performed by using two different forms which are given here.

```
(data_type)expression //data type in parentheses
```

where,

data_type = data type (also known as *cast operator)* to which the expression is to be converted.

- Eg:

```
avg=sum/float(i);
```

New cast operators:

reinterpret_cast

dynamic_cast

static_cast

const_cast

C++ Operators with Precedence and Associativity

- Operator precedence determines the grouping of terms in an expression. The associativity of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right