C++ PROGRAMMING ........ 10
POD VARIABLES AND THEIR RELATED FEATURES ........ 10
DATA TYPES ........ 10
INTEGER TYPES ........ 10
FLOAT TYPES: ........ 11
CHARACTER TYPES: ........ 12
FUNCTION TYPE ........ 12
STORAGE CLASS SPECIFIER IN C++ ........ 13
STORAGE CLASS SPECIFIER PUZZLES ........ 14
OPERATOR IN C++ ........ 17
POINTERS: ........ 18
POINTER PUZZLES ........ 18
REFERENCE ........ 20
REFERENCE PUZZLES ........ 20
VALUE CATEGORY ........ 20
CASTING ........ 20
REFERENCE VARIABLE IN C++ ........ 21
ARRAY ........ 22
INITIALIZATION OF ARRAY ........ 22
FLAVOURS OF ARRAY SIZE ........ 24
ARRAY RELATION WITH POINTERS: ........ 24
INTERNAL REPRESENTATION OF ARRAY: ........ 25
ARRAY PUZZLES ........ 26
CONDITIONAL AND LOOPING STATEMENTS ........ 30
IF STATEMENT ........ 30
FAQ ........ 30
GOTO ........ 30
GOTO PUZZLES ........ 30
OPERATORS ........ 31
GENERAL OPERATORS ........ 31
COMMA OPERATOR ........ 31
BITWISE OPERATOR ........ 32
PLACEMENT NEW OPERATOR ........ 32
MISCELLANEOUS ........ 32
GENERAL OPERATORS PUZZLES ........ 32
FUNCTIONS ........ 37
TERMINOLOGY ........ 37
FUNCTION PUZZLES ........ 38
FUNCTION POINTERS ........ 38
FUNCTION POINTER PUZZLES ........ 39
EXCEPTION HANDLING ........ 40
STANDARD EXCEPTIONS ........ 40
VIRTUAL FUNCTION IN C++ ........ 41
VIRTUAL FUNCTION PUZZLES ........ 42
CLASS AND CLASS RELATED FEATURES ........ 44
CONSTRUCTOR & DESTRUCTOR ........ 44

CONSTRUCTOR ........ 44
COPY CONSTRUCTOR ........ 45
DESTRUCTOR ........ 45
CONSTRUCTOR & DESTRUCTOR PUZZLES ........ 45
MISCELLANEOUS ........ 52
DIFFERENCE BETWEEN CLASS AND STRUCT IN C++ ........ 52
CLASS MISCELLANEOUS PUZZLES ........ 54
INHERITANCE ........ 57
INHERITANCE PUZZLES ........ 57
MEMBER FUNCTIONS ........ 60
MEMBER FUNCTION PUZZLES ........ 60
MODIFIERS ........ 62
MISCELLANOUS MODIFIERS ........ 62
OBJECTS ........ 62
OBJECTS PUZZLES ........ 63
EXCEPTION IN C++ ........ 72
STANDARD LIBRARY ........ 74
STANDARD TEMPLATE LIBRARY(STL) ........ 74
CONTAINER CLASSIFICATION ........ 74
CONTAINER CORRECT USAGE ........ 74
TIME AND SPACE COMPLEXITY OF STL ........ 76
TERMINOLOGY ........ 77
GENERAL STL INTERVIEW QUESTIONS ........ 77
UTILITY FUNCTIONS ........ 78
MAP IN C++ ........ 79
VECTOR ........ 82
VECTOR PUZZLES ........ 82
ITERATOR IN C++ ........ 82
BIT FIELD ........ 82
BIT FIELD PUZZLES ........ 82
UTILITY FUNCTIONS ........ 84
FILL FUNCTION IN C++ ........ 84
STRING ........ 85
TEMPLATE ........ 86
UTILITY ........ 90
UTILITY PUZZLES ........ 91
YET TO CATEGORIZE ........ 92
C++ MISCELLANEOUS ........ 93
DIFFERENCE BETWEEN C AND C++: MORE THAN YOU KNOW ........ 93
DIFFERENCE BETWEEN C++11 AND BELOW VERSION ........ 94
CORE DUMPED (SEGMENTATION FAULT) ........ 94
THE CODE SECRETS YOU MAY NOT KNOW ........ 96
C AND CPP DIFFERENCES ........ 96

COMPARISON BETWEEN C++98 AND C++11 ... 96
GOOD PROGRAMMING PRACTICE ... 97
WEIRD FACTS AND INTERVIEW QUESTIONS ... 97
INCLUDE FILES ... 120
CPP APPLICATIONS ... 123
UNDEFINED BEHAVIOR IN PROGRAMMING ... 124
COMPILATION AND LINKING ... 127
C++11 ... 127
C++14 ... 127
HEADER FILES FOR COMMONLY USED FUNCTIONS ... 128
MATH OPERATIONS ... 128
STANDARDIZATION ... 128
RETURN VALUE  OPTIMIZATION (RVO) AND COPY ELLISION ... 129
INPUT AND OUTPUT OPERATIONS ... 129
FILE HANDLING IN C++ ... 130
C PROGRAMMING ... 133
POD AND RELATED CONCEPTS ... 133
INITIALIZATION CONCEPTS: ... 133
VARIABLE ARGUMENTS IN C ... 134
SIZEOF AND SIZEOF(): ... 134
POINTERS ... 136
CONFUSING POINTER DEFINITIONS ... 136
EXTENDED DATA TYPE ... 137
PREPROCESSING DIRECTIVES ... 137
DIRECTIVE ... 137
MACRO ... 138
MACRO PUZZLES ... 138
STRUCTURE ... 140
TAG NAME IN STRUCT ... 140
STRUCT PUZZLES ... 142
STANDARD LIBRARY FUNCTION ... 143
UTILITY FUNCTION ... 143
PERROR AND STRERR ... 143
TIME ... 144
STRING IN C ... 148
INPUT AND OUTPUT OPERATIONS ... 148
FILE HANDLING ... 148
READING THE INPUT FROM STREAM ... 149
C AND C++ GENERIC CONCEPTS ... 150
BUILDING THE APPLICATION ... 150
ERROR AND WARNINGS AT RUN TIME AND COMPILE TIME ... 150

DESIGN ........ 151
OBJECT ORIENTED CONCEPTS (OOPS) ........ 151
ABSTRACTION ........ 151
TOOLS AND FRAMEWORKS ........ 152
DEBUGGING & BUILD MANAGEMENT ........ 152
PROFILING ........ 152
GDB DEBUGGER ........ 153
THEORITICAL COMPUTER SCIENCE ........ 153
TERMINOLOGY ........ 153
PROGRAMMING LANGUAGES ........ 153
TERMINOLOGY ........ 153
DIFFERENCES ........ 154
SOFTWARE ENGINEERING ........ 154
TYPE THEORY ........ 155
GRAPH THEORY ........ 155
NUMBER SYSTEMS ........ 159
BANKERS ALOGORITHM - OS ........ 160
DATA AND INFORMATION ........ 160
DATABASE ........ 160
ORACLE PRO*C/C++ ........ 160
ORACLE SQL ........ 160
QUERY PUZZLES ........ 160
VIEW ........ 161
ORACLE-PL/SQL ........ 161
CONCEPTS ........ 161
STRING MANIPULATION ........ 162
CONCEPTS ........ 162
INDEX ........ 162
TRIGGER ........ 163
CURSOR ........ 163
ORACLE-SQL ........ 163
CONCEPTS ........ 163
STANDARD PL/SQL LIBRARY ........ 164
FILE READING ........ 164
COMMON THERORITICAL CONCEPTS ........ 164
TERMINOLOGY ........ 164
PERFORMANCE ........ 165
KEYS IN DATABASE ........ 165

DATABASE NORMALIZATION ....165
USING PARTITION CLAUSE FOR AGGREGATE FUNCTIONS .... 166
UNIX AND UNIX RELATED PROGRAMMING .... 167
MULTITHREADING-UNIX .... 167
PTHREAD LIBRARY ....167
MISCELLANEOUS CONCEPT ABOUT THREADS ....172
DIFFERENCE BETWEEN SYNCHRONIZATION AND SERIALIZATION: ....172
SPURIOUS WAKEUP IN THREADS: ....172
DIFFERENCE BETWEEN MUTEX, LOCKS, SEMAPHORE ....173
DIFFERENCE BETWEEN BINARY SEMAPHORE & MUTEX ....173
MAXIMUM THREAD ALLOWED PER PROCESS ....173
WHAT HAPPENS WHEN PARENT THREAD IS DIED BEFORE CHILD THREAD IS ALIVE? ....173
SOCKET PROGRAMMING .... 174
BSD SOCKET ....174
FUNCTIONS AND THEIR USAGE ....175
MISCELLANEOUS ....186
TCP IMPLEMENTATION ....187
SERVER PROGRAM ....187
CLIENT PROGRAM ....189
UDP PROGRAM ....191
SERVER PROGRAM ....191
CLIENT PROGRAM ....193
SIGNALS .... 195
HARD AND SOFT LIMIT IN UNIX .... 195
STANDARDS .... 197
SEMAPHORE .... 197
MISCELLENEOUS .... 202
SYSTEM RESOURCES ....202
SYSTEM FILES ....202
PRO *C PROGRAMMING .... 202
TERMINOLOGY .... 202
DYNAMIC SQL STATEMENT IN PRO *C/C++ .... 203
MORE ABOUT VARCHAR AND SQL_CONTEXT .... 203
BASICS OF PRO C .... 204
CONNECTING WITH DATABASE .... 205
DATATYPES IN PRO*C/C++ .... 207
SQL STATEMENTS IN PROC .... 209
DATABASE CONCEPTS USED IN PRO *C/C++ .... 210

CURSOR ..... 213
CONTEXT ..... 214
POINTERS & MEMORY MANAGEMENT ..... 216
NEW OPERATOR IN C++ ..... 216
MAKE FILE ..... 219
MISCELLANEOUS ..... 222
UNEXPLORED C AREAS ..... 222
HEADER FILES ..... 222
CONDITIONAL COMPILATION: ..... 222
UNIX CONCEPTS ..... 223
HOW TO CHECK THE PORT WAS LISTENING(UNIX)? ..... 223
LIBRARY FILES IN UNIX ..... 223
SET LD_LIBRARY_PATH='/PATH/TO/SHARED/LIBRARY/' ..... 225
UNIX COMMANDS ..... 226
NETWORK REALTED COMMANDS ..... 226
GREP COMMAND IN UNIX ..... 227
UNIX COMMAND LINE MEANING ..... 230
CRON TAB ..... 233
APTITUDE ..... 234
QUANTITATIVE APTITUDE ..... 234
TRAIN PROBLEM ..... 234
TERMINOLOGY ..... 234
FORMULA ..... 235
TELECOM AND NETWORKING ..... 235
SOCKETS - NETWORKING ..... 235
STANDARDS ..... 236
OSI LAYER ..... 236
LOCAL HOST ..... 239
BASICS ..... 240
TERMINOLOGY ..... 240
COMMANDS ..... 241
IP ADDRESSING ..... 241
WIRELESS LAN ..... 241
NETWORK PROTOCOLS ..... 241
TCP/IP ..... 241
MISC PROTOCOLS ..... 242
SOAP ..... 242

UNCATEGORIZED TOPIC ........ 242
PROJECT BASED QUESTIONS ........ 243
HR QUESTIONS ........ 244
GIT ........ 246
ONLINE PROGRAMMING TESTS ........ 246
FILE EXTENSION AND MEANING ........ 247
ABBREVATION OF COMPUTER TECHNOLOGY TERMS ........ 248
VIM EDITOR ........ 249
ABBREVIATION TERMS ........ 250
PRIME NUMBER PROGRAM FOR LARGE NUMBERS ........ 251
CREATING EFFICIENT PROGRAMS ........ 253
PUZZLES ........ 254
COMMON PROGRAMMING MISTAKES ........ 254
DATA STRUCTURES ........ 255
TERMINOLOGY USED IN DATA STRUCTURE ........ 255
LINKED LIST ........ 256
REVERSE THE LINKED LIST ........ 256
HASH ........ 257
HASH FUNCTIONS ........ 257
MESSAGE DIGEST ........ 257
SHA (SECURE HASH AL GORITHM) ........ 257
WHIRLPOOL ........ 257
MISCELLANEOUS ........ 257
GRAPH ........ 258
TREE ........ 260
REPRESENTATIONS ........ 266
REPRESENTING TREE IN ARRAY ........ 266
AVL TREE ........ 267
INSERTING NODE IN AVL TREE ........ 267
SNAPSHOT OF INSERTION AND DELETION ........ 268
RED BLACK TREE ........ 268
INSERT A ELEMENT IN RED BLACK TREE ........ 269
DELETE IN RED BLACK TREE ........ 269
SNAPSHOT FOR RED BLACK TREE INSERT AND DELETION ........ 270
MISC DATA STRUCTURES ........ 270
BINARY SEARCH TREE ........ 271
DATA STRUCTURES ........ 271
TERMINOLOGY ........ 271
DIFFERENCE AMONG SIMILAR DATA STRUCTURE ........ 272

QUEUE AND DEQUE DIFFERENCE ....272
BINARY TREE ....272
B+ TREE ....272
REAL WORLD PROBLEMS ....273
ALGORITHM ....273
ANALYSIS OF ALOGORITHM ....273
ASYMPTOTIC ANALYSIS ....273
ALOGORITHM METHODOLOGY ....273
DYNAMIC PROGRAMMING ....273
REAL WORLD PROGRAMS ....273
SELECTION OF ALOGORITHM ....274
DATA STRUTURE ORIENTED ....274
STACK BASED ALGORITHMS ....275
TERMINOLOGY ....275
INFIX TO POSTFIX ....275
INFIX TO PREFIX CONVERSION ....275
REVERSE STACK USING RECURSION ....276
CHECK FOR BALANCED EXPRESSION ....276
SORTING ALGORITHM ....277
INSERTION SORT ....277
HEAP SORT ....278
TRAVELLING SALES MAN PROBLEM ....278
PROGRAMMING CONCEPTS BASED ALGORITHM ....278
STACK RELATED ALGORITHMS ....278
ARRAY RELATED ALGORITHMS ....279
PRINT FROM KTH LARGEST NUMBER IN DECREASING ORDER ....279
TREE RELATED ALGORITHMS ....279
CONVERT BINARY TREE TO LINKED LIST ....279
LINKED LIST RELATED ALGORITHMS ....280
INSERT A NODE IN SORTED CIRCULAR LINKED LIST ....280
REVERSE LINKED LIST IN GROUP ....281
MERGE TWO SORTED LINKED LIST AS SORTED LINKED LIST ....281
SPLIT CIRCULAR LINKED LIST IN TO TWO ....281
BATCH SCRIPTING ....282
PROPERTIES OF BATCH ....282
GRAMMAR ....282
PYTHON ....283
INPUT AND OUTPUT ....283
SHELL SCRIPTING ....283

TERMINOLOGY ........ 283
BASICS OF SHELL SCRIPTING ........ 283
DESIGN PRINCIPLES ........ 286
SOLID ........ 286
DESIGN PATTERNS ........ 287
DESIGN PATTERN TERMINOLOGY ........ 287
VISITOR PATTERN ........ 288
EXCEL-VBA PROGRAMMING ........ 288
PROGRAMMING CONCEPTS ........ 288
MISC SUBJECTS ........ 289
BANKING ........ 289
TERMINOLOGY ........ 289
PAYMENT ........ 290
GROUPS AND CORPORATIONS ........ 290
CHEQUE ........ 292
CLASSIFCATIONS AND CATEGORY ........ 293
CLASSIFICATION OF MSME ENTERPRISES ........ 293
CLASSIFICATION OF FARMERS BASED ON LAND ........ 293
CLASSIFICATION OF BANKS BASED ON SIZE ........ 293
CLASSIFICATION OF ACCOUNTS IN BANK ........ 294
RESERVE BANK OF INDIA ........ 295
TYPES OF ACCOUNTS IN BANK ........ 295
HEADQUATERS ........ 295
BANK ACCOUNT ........ 296
TERMINOLOGY ........ 296
DEPOSIT ACCOUNTS ........ 296
MONETARY & MONETARY RELATED ........ 296
TRADING ........ 297
LOANS & INTEREST ........ 298
TERMINOLOGY ........ 298
TYPES OF INTEREST ........ 299
PSL(PRIORITY SECTOR LENDING) ........ 300
MISCELLANEOUS ........ 301
TERMINOLOGY ........ 302
MISCELLEANEOUS ........ 303
BASEL NORMS ........ 303
SECTORS IN INDIAN ECONOMY ........ 304
OPERATING SYSTEM ........ 305
TERMINOLOGY ........ 305
KERNEL ........ 305

MULTITHREADING-OS ..... 306
MATHEMATICS ..... 306
NUMERIC MATHEMATICS ..... 306
MISCELLANEOUS ..... 306
TERMINOLOGY ..... 306
SYMBOLS ..... 307
IPC – INTER PROCESS COMMUNICATION ..... 307
PIPE ..... 307
SHARED MEMORY ..... 307
MESSAGE QUEUE ..... 308
REMOTE PROCEDURE CALL ..... 308
MISCELLANEOUS ..... 308
MESSAGE QUEUE ..... 309
TRASH CAN ..... 310

# C++ PROGRAMMING

## POD VARIABLES AND THEIR RELATED FEATURES

### DATA TYPES

#### INTEGER TYPES

Unsigned int and unsigned are same

Unsigned long int and unsigned long are same

**Limits for integer types:**

All limits for integer types are defined under  the limits.h file

All the limits are configured in following macro

```
INT_MAX
FLT_MIN
UINT_MAX
ULINT_MIN
```

**Internal implementation:**

C & C++ Standard has no restriction on implementation of the storing signed number in the machine. It's depending on the compiler implementation.

Signed number can be stored in the machine by any one of the method below,

1. Signed magnitude

   MSB (Most significant bit) will be used as sign bit and rest of the bits will be used as value bit

2. 1's complement method
3. 2's complement method

   This is most widely used method by all machines.

### Why 2's complement is used widely?

This is used widely because addition of signed and unsigned number doesn't require a special logic. Whereas when you use other methods, adding of signed and unsigned number requires the special logic.

4. Excess-k Method

## FLOAT TYPES:

1. We cannot apply modulo operator on floating point numbers. We should use fmod(x,y) for finding the modulo for float.

### Internal implemenatation:

Standard hasn't  said anything about floating point representation on the machine. It is purely depends on the compiler implementation.

Most of the compilers are following IEEE 754 standard for representing floating point numbers.

Floating point numbers are represented by dividing the number in to following chunks,

```
Significant * base ^ exponent
```

### FAQ

### How to find the number is negative when it is represented using 2's complement?

It is identified based on the 1st bit. If 1st bit is zero, then it is negative number.

## CHARACTER TYPES:

1. Char can be unsigned 8 bit integer or signed 8 bit integer. It depends on the platform.
2. As per standard when you declare the variable by declaring simply as char, then variable can be signed char or unsigned char. This purely depends upon the implementation.

```
char a; // It can be signed or unsigned char
```

## FUNCTION TYPE

Function type is kind of data type which store the function address.

### Stdint header file:

### Need for the stdint:

The main objective of the stdint was to provide the portability for the code

int32_t is same as int datatype but int32_t guarantees that the size of the int datatypes is 32 bit at all platforms but normal int cannot guarantees that.

### Recursive (or) Inductive Data type

Recursive data type is the data type which contains the value and same type as member.

```
struct List
{
 int value;
 List *m_next_node; // Same Data type as member to refer next node
};
```

Mutual recursion data type is representing the similar kind of data type as member of data type.

```
t: =v f
v: [t[1],t[2]…t[n]]
Where t=tree and f=forest(List of trees)
```

## TYPE conversion in c++

### integer to unsigned integer

```
unsigned int i=-2;
```

While converting the signed integer to unsigned integer , compiler follows the following rule

```
unsigned int i =( INT_MAX + 1) - 2;
```

Technically, 2 power n is added to it where n is the number of bits used to represent the unsigned int

## STORAGE CLASS SPECIFIER IN C++

### Auto:

Auto is the storage class specifier, which indicates that variable will be automatically deallocated by the compiler when goes out of scope.

Auto keyword cannot be used for global variables.

Meaning of keyword auto has been changed since c++11 as it will deduce type automatically based on the return type.

### STATIC:

#### Static keyword for data members for class

### REGISTER:

Register is one of the storage class specifier which instructs the compiler to use a register instead of allocating memory in physical memory.

But most cases, compiler itself will allocate some variable as register as a part of optimization process.

Some compilers will not allow to get the address of the register variable but gcc allows it.

Register keyword cannot be used for the global variables.

Keyword register is deprecated until c++17. After c++17 this keyword is reserved and no use in this keyword.

### Extern

Extern keyword can be used for variables inside the block. But variables that is used for extern should be in global scope.

### Thread local

Thread local storage class specifier is used to create a thread specific variable.

Earlier, standard has to specification regarding thread specific variable. Compilers provided this functionality as an extension.

Thread local cannot be used with auto and register, only it should be combined with static and extern.

GNU Compiler:

__thread keyword is used to create an thread specific variable

## Mutable:

Usually data members cannot be modified when function is declared as const. If there is situation that even if function is declared data member need to be modified due to unavoidable situation. To achieve this declare the data member with mutable specifier.

```
#include<iostream>

class test
{
   public:
   int a;
   mutable int b;

   void updateConstFunc()const
   {
      // a=1; ==> Not possible since function is const
      b=2; //==> possible since mutable specified
   }
};

int main()
{
   test t;
   t.updateConstFunc();
   return 0;
}
```

## ACCESS SPECIFIER IN C++

1. Global variables need to be accessed in local scope by using unary scope resolution operator
2. When variable local and global scope has same name then, local variable will be accessed when using that variable. If global variable need to be accessed, then unary scope resolution operator should be used.

## STORAGE CLASS SPECIFIER PUZZLES

1. **Whether data members of the class can be initialized at declaration time?**

```
#include<iostream>

class ArrayNotFound
{
 public:
       static const int val=10;
};

int main()
{
 std::cout<<ArrayNotFound::val<<std::endl;
 return 0;
}
```

OUTPUT:

```
10
```

EXPLANATION:

Const static data members of the class can be initialized at the time of declaration. Other data members cannot be initialized during declaration of the class.

2. **Whether const static data members of the class has to be redefined outside the class declaration as other static data members of the class?**

```
#include<iostream>

class ArrayNotFound
{
 public:
       static int a;
       static const int val=10;

};

int ArrayNotFound::a;
const int ArrayNotFound::val;

int main()
{
 std::cout<<ArrayNotFound::val<<std::endl;
 return 0;
}
```

OUTPUT:

```
10
```

EXPLANATION:

No. const static data members of the class no need to be declared again as other static data members of the class. Program will compile fine, without the following line,

```
const int ArrayNotFound::val;
```

But initialization of the const static data member will remain same as we declared in the class declaration.

3. **Whether static function can be called by using the objects of the class?**

```
#include<iostream>

class ArrayNotFound
{
 public:
	static void callme()
	{
		std::cout<<"A"<<std::endl;
	}
};

int main()
{
 ArrayNotFound obj;
 obj.callme();

 return 0;
}
```

OUTPUT:

```
A
```

EXPLANATION:

Yes, static function can be called by either by using a objects of the particular class or without objects, on the other hand non-static functions cannot be called without using objects.

4. **How to access the global variable when local variable has same name as global variable?**

```
#include<iostream>

double g=44;
int main()
{
 double g=52;
 {
	std::cout<<g<<":"<<::g<<std::endl;
 }

 return 0;
}
```

OUTPUT:

```
52:44
```

EXPLANATION:

When particular scope contains the variable with name same as global variable, then global variable can be accessed using the unary scope resolution operator.

::g

## OPERATOR IN C++

### SCOPE RESOLUTION OPERATOR:

1. There are two types of scope resolution operator
   a. Unary scope resolution operator(Accessing Global Variable)
   b. Binary scope resolution operator(Defining class member function outside class)

### TYPEDEF IN C++

TYEPDEF INT ARR[5]; IS EQUAL INTEGER ARRAY

### Difference between Data type, Data Structure and Abstract data type

- Abstract data type: Data type is represented in terms of mathematical model, such as possible values, allowed operation irrespective of the machine implementation. For instance, Integer can be represented as the abstract data type

```
// Integer Abstract Data type
Data:
 infininy<=X<=infinity
Operations:
 Add()
 Subtract()
 Multiply()
 Divide()
```

- Data type: Data type is represented in terms of mathematical model, such as possible values, allowed operation with respect to the machine implementation.

```
// Integer Data type in C++ 32 bit compiler
Data:
-32767<=X<=32768
Operations:
 Add()
 Subtract()
```

```
Multiply()
Divide()
```

- Data structures: This is how data is manipulated. Consider integer as a data structure, where as integer can be stored, retrieved in many forms from one machine to other machine. For instance, negative numbers can be represented in varies forms such as signed representation, 2's complement etc.

## POINTERS:

### TERMINOLOGY

#### STRICT ALIASING RULE:

While dereferencing the pointer to actual type, then pointer and the value pointed by the pointer should be of same type. This is called as strict aliasing rule.

### POINTER PUZZLES

1. **What happens when we access the single pointer with double array braces?**

```
#include<iostream>

int main()
{
 int i=10;
 int *sp=&i;
 int **a=&sp;
 std::cout<<a[0][0];
 std::cout<<sp[0][0];

 return 0;
}
```

OUTPUT:

```
Compilation error

9:20: error: invalid types 'int[int]' for array subscript

 std::cout<<sp[0][0];
```

EXPLANATION:

Pointer variable can be accessed using the array subscript operator, but compiler will validate this access. When it is single pointer only single array brace is allowed and for double pointer double array braces is allowed. Here following line works fine perfectly,

```
std::cout<<a[0][0];
```

But single pointer cannot be accessed using the double array braces.

## REFERENCE

### REFERENCE PUZZLES

1. **Whether same reference variable can be assigned with other variable after declaration?**

```
#include<iostream>
int main()
{
 int a=10,b=20;
 int &ref=a;
 std::cout<<ref<<":";
 ref=b;
 std::cout<<ref;
 return 0;
}
```

OUTPUT:

```
10:20
```

EXPLANATION:

Yes reference variable can be changed in such a way that to point other elements.

## VALUE CATEGORY

### Rvalue

1. Rvalue will bound only to const objects.

## CASTING

### Static cast:

When type of the variable is known then it is called static cast.

### Dynamic cast

When type of the variable is not known then it is called dynamic cast. Dynamic cast should be used only for polymorphic class.

#### Polymorphic class:

Class which contains at least one virtual function is called as polymorphic class.

What happens when invalid class is casted?

It will return NULL, so it is always safe to NULL check the dynamic class.

## REFERENCE VARIABLE IN C++

Reference is used in c++ to create a alias for the variable. It is just used like a nick name. Consider you have variable a, if you need to use the variable a as b, you can create a reference.

To create a reference append a ampersand symbol as prefix to variable at a time of creation. Don’t confuse with the ampersand symbol. It is not a symbol used for address reference here.

Consider the example

```
int a=5;
int a2=10;
int &b=a;
int &c; //Not valid need to initialize
int &d=NULL;  //not valid statement
int *p=NULL;
int e=p;  //not valid assigning NULL pointer
b=a2;  //not valid cannot initialize reference variable again
myclass &myref_obj=my_obj;
```

In above example, the alias name for a is created, it is b.

So here are some scenarios

```
cout<<a;  //prints 5
cout<<b;  //prints 5
b++;  //changes the value of a
cout<<a; //prints 6
cout<<b;  //prints 6
```

### Datatype conflict:

You cannot make a reference for the different data types  For Eg you cannot refer a integer variable to the double reference.

```
int a;
int &b=a;  /valid assignment
double &c=a;   //invalid assignment
```

### Difference between pointer and references:

The pointer and reference are almost same with some of the following differences.

You can assign a NULL value to pointer, but you cannot do that in reference.

Pointer can be reassigned many number of times, but in a reference we cannot do that. Reference can be initialized only once.

Reference must be initialized at the time of declaration. But pointers can be initialized any number of time.

If pointer is incremented it will point to next memory address, whereas if reference is incremented the variable that reference is pointing to will be incremented.

Although, reference seems to be different concept, Probably it is internally treated as a pointers by the compiler. Some other compilers will treat it as alias.

## REFERENCE VARIABLE IN C:

In c there is no reference variable. For call by reference use pointer.

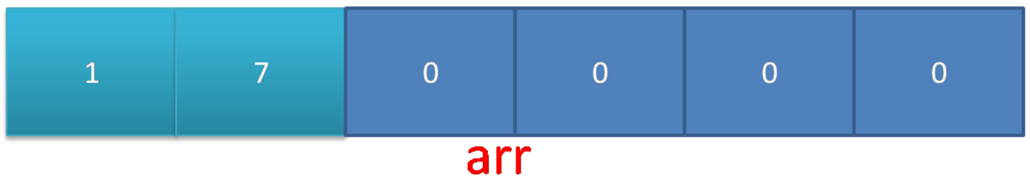
This is similar to pass by address.

```
int callme(int *a)
{
*a++; // g will be incremented.
}
int main()
{
int g;
callme(&g);
}
```

## ARRAY

### INITIALIZATION OF ARRAY

int arr[6]={1,7};



**All other blocksthat is uninitialized will be initialized to zero**

```
int  arr[10];
```

Garbage value will be initialized if we use this format.

```
static int arr[10];
```

In this case all the elements of the array will be initialized to zero because of the keyword static.

```
int arr[10]={0};
```

The above code will initialize the all elements of the array to zero.

```
int arr[10]={};
```

The above code will initialize the all elements of the array to zero.

```
int arr[10]={1,2}
```

The above code will initialize the first two elements of the array as initialized and remaining all elements will be initialized as zero.

```
int arr1[10]={1,2,3,4,5};
// int arr2[10]=arr1;  This kind of initialization is not possible in c program
```

The above kind of initialization is not possible in c program. You have to use either the for loop or memcpy for initialization.

```
char c[10]=”hello”;
// int i[10]=”12345”; Not possible for int,float
```

The char array can be initialized using the doble quotes. But it is not possible in the case of float and int.

```
int array[10]={[2]=2,[5]=1};
char c[10]={[2]=’a’,[3]=’d’};//strlen(c) is zero.
```

The array can be explicitly initialized using the above format. In case of int the remaining array will be initialized to zero. In case of char strlen returns zero if the first element of the array was not initialized.

```
char c[10]=” “;
```

The above will initialize the 1st element of array to empty space. In similar fashion if you leave two spaces then the first two index will be initialized to empty space.

IN

## FLAVOURS OF ARRAY SIZE

### VARIABLE LENGTH ARRAY

1. It is not part of standard.

### Tail padded array or Zero size array

Array with 0 size is called as tail padded array or zero size array. It is deprecated after c++11

### Flexible array member

Array with no size specified is called flexible array member. Same class can have many flexible array member.

```
int a[];
```

Only class or structure can have the flexible array member. Single class or structure can have more than one flexible array member.

## ARRAY RELATION WITH POINTERS:

1. Array returns address of the first element in the array  of type as datatype of array element  when used as rvaue without ampersand operator.

2. Array returns address of the first element in the array  of type  array  when used as rvaue  with ampersand operator.

```
int main()
{
   int a[3]={0};

   int *p1=a;

   int *p2= (int *)&a; // casting to int pointer
   int (*p3)[3]= &a; // &a returns array type

   // When p3++ is incremented, it will increment by 3*sizeof(int)

   return 0;
}
```

3. Exception when array used as rvalue is not a pointer was sizeof operator and unary ambersand operator.

4. When we increment the pointer to array, then it will increment with sizeofarray*sizeof(data type of array element)

5. When dereferencing a pointer to array, upon first dereference we get the array, and upon next dereference we get the actual element in the array

```
#include<iostream>
int main()
{
   int a[10]={1,2,3,4,5};

   int (*p)[10]=&a;

   printf("\n  :: %d ", *p[0] );
   printf("\n  :: %d ", **p );


   return 0;
}
```

## When array will be decompose in to pointers?

1. When some offset is added to the array.

```
int a[10];
*(a+1)=2;
```

## INTERNAL REPRESENTATION OF ARRAY:

Array is represented internally in two forms, they viz,

1. Row major representation

    a. All continuous row was represented in linear fashion

2. Column major representation

All continuous column was represented in linear fashion

## Array braces

In c and cpp array elements can be accessed in both ways

a[0]

0[a]

The above two formats are valid because compiler internally removes the braces and replaces by following

0+a or a+0

The following program compiles fine without error.

```
#include <cstdio>

 int main()
 {
    int a[]={4,47,75};
    printf("\n Value :: %d ", 0[a] );
    return 0;
 }
```

## ARRAY PUZZLES

1. **What is the size of the two dimensional array when some integer is added to it?**

```
/*
Assume the following size
int - 4 bytes
pointer 4 bytes
*/

#include<iostream>
int main()
{
 int b[10][10];

 std::cout<<sizeof(b)<<":"<<sizeof(b+1)<<std::endl;

 return 0;
}
```

OUTPUT:

```
400:4
```

EXPLANTION:

Array will be decomposing to pointer at many situations. This is one of the situation here. Since we have added one integer value with array, array was decomposed to pointer in the second print, so it is returning the size of pointer instead of size of array.

2. **Can we do an parenthesis based initialization for dynamic array?**

```
#include<iostream>
```

```
int main()
{
 char *s1=new char[10]('a');
 char *s2=new char[10]();
 std::cout<<s1<<":"<<s2<<std::endl;
 return 0;
}
```

OUTPUT:

Compilation error

parenthesized initializer in array new

EXPLANATION:

Dynamic array cannot be initialized with parenthesis. So following line will give a compilation error.

```
 char *s1=new char[10]('a');
```

Where as you can initialize with uniform initialization in c++11

```
 char *s1=new char[10]{'a'};
```

But empty parenthesis initialization is  allowed even for dynamic array.

```
 char *s2=new char[10]();
```

This is will just make all the char to 0(NULL) initially.

3. **what is size of the two dimensional array when dereference operator was used?**

```
//Assume size of int as 4 bytes,pointer size as 4 bytes

#include<iostream>
int main()
{
 int (*ptrToArray)[10][10];
 int *arrayOfPtr[10][10];
 std::cout<<sizeof(ptrToArray)<<":"<<sizeof(*ptrToArray)<<":"<<sizeof(**ptrToArray
)<<":"<<sizeof(arrayOfPtr)<<":"<<sizeof(*arrayOfPtr)<<":"<<sizeof(**arrayOfPtr);

 return 0;
}
```

OUTPUT:

```
4:400:40:400:40:4
```

EXPLANATION:

Remember ptrToArray is a pointer variable(Pointer variable for type array[10][10]) and arrayOfPtr is type(Pointer type- array of pointers)

1. sizeof(ptrToArray) : Since this is pointer it returns 4. This is similar to sizeof(int)

2. sizeof(*ptrToArray) – This is equal to dereferencing the pointer. Since our type is integer array of 10*10, it returns the 4*10*10=400 bytes. This is equal to sizeof(int[10][10])

3. sizeof(**ptrToArray) - This is equal to referencing the 1st row of the array. Since we have 10 elements in one row, size will be 10*4=40 bytes. This is similar to sizeof(int[10])

4. sizeof(arrayOfPtr) – Since this is type it returns the size of the type. Our type is int which holds about 10*10 elements. So size is 4*10*10=400 bytes. This is equal to sizeof(int[10][10])

5. sizeof(*arrayOfPtr) – This is equal to dereferencing a 1st row of array type. So it returns 4*10=40 bytes. It is equal to sizeof(int[10])

6. sizeof(**arrayOfPtr) – This is equal to dereferencing the one single element. So it returns 4 bytes. It is equal to sizeof(int*)

## 4. What happens when we use increment operator on a array parameter in function?

```
#include<iostream>

void PostInArrayNotFound(int p[][3])
{
 p++;
 std::cout<<p[0][2];
}


void singleArray(int q[3])
{
 q++;
 std::cout<<q[0]<<":";
}
int main()
{

 int a[3]={1,5,7};
 singleArray(a);

 int b[][3]={{25,50,100},{10,20,30}};
 PostInArrayNotFound(b);
 return 0;
}
```

OUTPUT:

```
5:30
```

EXPLANATION:

First Let us take this function,

```
void singleArray(int q[3])
```

Here the above statement will be changed internally by the compiler as

```
void singleArray(int *q)
```

This is because, array will decompose in to pointer, when we pass to the function. Now q is of type integer pointer. Now q contains the first element in the array. When incremented it will point to next element in array.

Let us take the another function now,

```
void PostInArrayNotFound(int p[][3])
```

Here, the above statement will decompose as,

```
void PostInArrayNotFound(int **p)
```

Now p is of type  pointer to int[3]. Note that array will decompose as pointer for single dimensional array. For multi dimensional array, array will decompose as pointer to array.

Now p contains the address of first element in array. But when we use increment operator, it will point to next row, rather than next element because pointer is pointing to type int[3], whereas in our previous case, it is pointing to int so it is pointing to next element.

## 5. Whether size can be given as null for array?

```
#include<stdio.h>

struct myStruct
{
 int a;
};

int main()
{
 struct myStruct a[]={};
 printf("%d",sizeof(a));
 return 0;
}
```

OUTPUT:

```
0
```

EXPLANATION:

Yes, sizeof the array can be null, but only when it is initialized with empty paranthesis. But size of the array cannot be null be null when it is simply declared and it is not initialized with parenthesis.

## ENUM

Enums can have the negative value. When negative value is specified for one list, then for next list 1 will be added. For eg: If value is -8 then for next list value will be -7(-8+1)

## CONDITIONAL AND LOOPING STATEMENTS

### IF STATEMENT

#### FAQ

1. **What happen when we assign variable inside the if statement?**

```
if(a=10)
```

The above statement is equal to

```
if(10)
```

### GOTO

#### GOTO PUZZLES

1. **Whether label which is declared in other functions can be accessed in the current function?**

```
#include<iostream>

void PostInArrayNotFound()
{
 goto myLabel;
}

int main()
{
 PostInArrayNotFound();
 myLabel:
        std::cout<<"A"<<std::endl;

 return 0;
}
```

OUTPUT:

Compilation error

error: label 'myLabel' used but not defined

EXPLANATION:

Labels can be accessed only to its local scope as similar to accessing the normal variables and objects.

2. **Whether labels which are declared within the inner block scope of function can be accessed by the same function?**

```
#include<iostream>

int main()
{

 goto myLabel;

 {
        myLabel:
        std::cout<<"A"<<std::endl;
 }

 return 0;
}
```

OUTPUT:

A

EXPLANATION:

Yes it can be accessed, whereas labels which are declared outside the function cannot be accessed from other functions.

## OPERATORS

### GENERAL OPERATORS

#### COMMA OPERATOR

## OPERATOR Vs SEPARATOR

Comma will act as both operator as well as separator and it depends on the context.

```
int a,b; //Separator
int a=(10,33); // Operator
```

## Associativity & Precedence:

It has left to right associativity.

It is lowest precedence of all operator.

## BITWISE OPERATOR

### Frequently asked questions:

1. **How to compare whether value is zero or 1 by using bit wise operator?**

```
var&1 == x
```

## PLACEMENT NEW OPERATOR

Initialization of object at specified address is called placement new operation.

## MISCELLANEOUS

### Whether semicolon is operator in C++?

No, semicolon is not a operator in C++.

## GENERAL OPERATORS PUZZLES

1. **Priority of comma operator between equal operator and brackets?**

```
#include <iostream>
int main()
{
   int k,m;
   k = 1, 2, 3;
   m = (1, 2, 3);
   std::cout<<k<<m<<std::endl;
   return 0;
}
```

OUTPUT:

```
13
```

EXPLANATION:

Comma operator executes from left to right and return the right most element.

Consider the below statement

```
  m = (1, 2, 3);
```

Since brackets takes higher precedence than equal operator, first (1,2,3) will get executed. Here, it returns the 3, so m is stored with value 3.

So, why it is not happened for the below statement?

```
  k = 1, 2, 3;
```

Here equal takes the higher operator precedence than comma operator.

Look here for operator precedence table.

http://en.cppreference.com/w/cpp/language/operator_precedence

2. **Can we place the array variable inside the array subscript operator?**

```
#include<iostream>
int main()
{
 int a[]={0,1,2,3,4,5};
 std::cout<<a[5]<<5[a]<<std::endl;
 return 0;
}
```

OUTPUT

```
55
```

EXPLANATION:

Array subscript operator can be used in any of the following way,

```
a[5] or 5[a]
```

Both forms are valid syntax. Commutative of array subscript operator is true. Hold on, it will not work for operator overloading. Only for POD types it will work.

How it works?

Array subscript operator internally decomposes in to following form.

Consider a[5] in statement. Internally a[5] will be converted to a+5. Since a is a pointer adding 5 will point to 6th (Remember array starts with zero) element in the array.

Similarly 5[a] will be converted in to 5+a.

## 3. Can we use the array subscript operator for char literal without assigning in variable?

```
#include<iostream>
int main()
{
 std::cout<<5["ArrayNotFound"]<<"ArrayNotFound"[5]<<std::endl;
 return 0;
}
```

OUTPUT:

```
NN
```

EXPLANATION:

Even though it seems to be very new think, concept behind it is very simple.

Here,

```
 std::cout<<5["ArrayNotFound"]<<"ArrayNotFound"[5]<<std::endl;
```

The above statement is equal to the following one,

```
char *s=”ArrayNotFound”
std::cout<<5[s]<<s[5]<<std::endl;
```

Oh. Hope you got it. Instead of using the char pointer variable, directly used the string literal in the statement.

## 4. What is operator precedence among the pre-increment, post-increment and indirection operator?

```
#include<iostream>
int main()
{
 int a[]={8,6,7};
 int *p=a;
 std::cout<<*p++<<":"<<*++p<<":"<<++*p;

 return 0;
}
```

OUTPUT:

```
6:6:9
```

EXPLANATION:

Post increment takes the first priority, precedence between pre-increment and indirection is same. So, associativity should be applied. For this precedence group, associativity is right to left.

5. **What is operator precedence among the pre-increment, post-increment and indirection operator during operator overloading?**

```
#include<iostream>

class ArrayNotFound
{
 public:
       ArrayNotFound& operator*()
       {
             std::cout<<"A";
       }

       ArrayNotFound& operator++(int a) //post increment  operator overload
       {
             std::cout<<"PO";
             return *this;
       }

       ArrayNotFound& operator++() //pre increment operator overload
       {
             std::cout<<"PR";
             return *this;
       }
};

int main()
{
 ArrayNotFound obj1,obj2;
 *obj1++;
 std::cout<<":";
 *++obj1;
 std::cout<<":";
 ++*obj1;

 return 0;
}
```

OUTPUT:

```
POA:PRA:APR
```

EXPLANATION:

Post increment takes the first priority, precedence between pre-increment and indirection is same. So, associativity should be applied. For this precedence group, associativity is right to left.

## 6. Consider post increment operator is overloaded, then when it is called when other overloaded operators are used in the expression?

```
#include<iostream>

class ArrayNotFound
{
 public:
       ArrayNotFound& operator++(int a)
       {
              std::cout<<"++"<<":";
              return *this;
       }

       ArrayNotFound& operator*()
       {
              std::cout<<"*"<<":";
              return *this;
       }
};

int main()
{
 ArrayNotFound obj;
 *obj++;

 int a[]={10,20,30};
 int *p=a;
 std::cout<<*p++;

 return 0;
}
```

OUTPUT:

```
++:*:10
```

EXPLANATION:

We all know that for POD types like int, post increment operator will return the current value for the expression and it will increment the location value.

How it works for operator overloading in C++?

Here there is no such thing like updating in the location. Since postfix operator is overloaded, programmer has to do such stuffs like following,

```
void operator++(int a)
{
 int l_int=this.m_int;
  this.m_int++;
  return l_int;
}
```

When you look here, old value is returned and at location value is incremented.

# FUNCTIONS

## TERMINOLOGY

### Helper function:

Function which is helps to do some activity for another function.

Mostly it is used in the recursive function. Say suppose when some part of the function has to be called recursively, then write some helper function with that part and call it recursively.

## GLOBAL FUNCTIONS:

### Variable Arguments:

Usually variable arguments can be achieved using the ellipsis operator.  To achieve in c++ style , left shift operator can be overloaded to all data types which you want to support. cout object of ostream generalization class is implemented in such an manner.

```
std::cout<<myInteger<<myString;
```

cout is an object of basic_ostream class. basic_ostream class overloads the << operator with integer type, string class.

### Mutually recursive function

Mutual recursive function is a function which call the other function and other function which calls the same function which is being called.

```
Func1()
{
 ...
 Func2()
 ...
}

Func2()
{
 ...
 Func1()
 ...
}
```

## FUNCTION PUZZLES

1. **What happens when variable is initialized with function which returns void?**

```
#include<iostream>

void callMe()
{
 std::cout<<"A"<<std::endl;
}
int main()
{
 int a=callMe();
 return 0;
}
```

OUTPUT:

Compilation error

error: void value not ignored as it ought to be

EXPLANATION:

When a variable is initialized with function which return void, then compiler will thrown an error.

## FUNCTION POINTERS

### RETURNING FUNCTION POINTER:

Syntax for returning function pointer is as follows

```
Return_type_of_returning_function
(*function_name(returning_function_arguments))(actual function arguments);
```

For Eg:

```
int (*myFunc(int))(char)
```

Here myFunc is a function which returns the function of type int (*)(char).

## FUNCTION POINTER PUZZLES

### 1. What happen when we dereference the function pointer?

```
#include<iostream>
int callme()
{
 std::cout<<"$"<<std::endl;
}
int main()
{
 int (*functionPtr)()=callme;
 (***********functionPtr)();
 return 0;
}
```

**OUTPUT:**

```
$
```

**EXPLANATION:**

Dereferencing the function pointer has no effect. It means it remains the same after dereferencing, unlike other pointers will get its value in the specified location.

### 2. How to return the function?

```
#include<iostream>

int* PostInArrayNotFound(int *p)
{
 return p;
}

int *(*broker(int*))(int* p)
{
 return PostInArrayNotFound;
}

int main()
{
 int i=40;
 int j=10;

 std::cout<< *((*broker(&j))(&i)) << ":" << *broker(&j)(&i) << std::endl;


 return 0;
}
```

OUTPUT:

```
40:40
```

EXPLANATION:

Program just returns the function  pointer. Further study here FUNCTION POINTERS

## EXCEPTION HANDLING

### No throw statement

No throw is an constant value which is passed as an argument to the new operator overloading function.

Actual definition of the new operator would be

```
void* operator new(size_t size, const std::nothrow_t& nothrow_value);
```

When calling it will be called  like this

```
MyClass *obj = new (std::nothrow) MyClass;
```

Here std::nothrow value will be passed, and function will not throw any exception on seeing that value.

## STANDARD EXCEPTIONS

### IMPLEMENATION DETAILS

- Exceptions are classified and separate class is created for classified exception and every class will inherit the base class Exception.
- Every exception class will overload the virtual function called 'what' to print the error message.

```
// Warning : This is possible implementation, actual implementation will vary

class Exception
{
 public:
 virtual const char * what()=0;
};

class bad_alloc : public Exception
{
 const char * what()
 {
       return "Running out of space";
```

```
    }
  };
```

## MEMORY RELATED EXCEPTION

### bad_alloc

When memory allocation is failed, then bad_alloc exception will be thrown.

## VIRTUAL FUNCTION IN C++

Virtual function is a function whose behavior in base class is overridden by the derived class. Derived class will also have the same signature as base class but the blocks of code will vary.

This is one of the implementation that comes under the category of polymorphism.

## Note

1. No need to mention the keywords as virtual for virtual functions in derived class.

### When we need to use the virtual function?

When we need to override the certain behavior of base class.

### How virtual function is called at run time?

1. Based on the maintenance of virtual table. Virtual table is also called as dispatch table.
2. Each inherited class maintains own virtual table. If 4 classes are inherited from the base class (Virtual), then 4 virtual tables will be maintained.

### Statically Typed and DynamicallyTyped languages

Language is statistically typed when its type of variable is known at compile time

Eg: C,C++,Java

Language is Dynamically typed languages when its type of variable is known at run time.

Eg: perl

### Early/compile-time binding and late/run-time binding:

Type of the variable is known at compile time is called Early binding

Eg: Intialzing variables, creating object for class

Type of the variable known at run time is called Late binding

Eg: Virtual methods, Function pointers.

## Pure Virtual function:

For pure virtual function the value will be NULL in virtual table

## VIRTUAL FUNCTION PUZZLES

### 1. What happens when virtual function is called from the base class member function?

```
#include<iostream>

class Language
{

 public:
       virtual void getName()
       {
             std::cout<<"my name is Language"<<std::endl;
       }

       void getProperties()
       {
             getName();
       }
};

class cpp : public Language
{

 public:
       virtual void getName()
       {
             std::cout<<"my name is cpp"<<std::endl;
       }
};

int main()
{
 Language *vptr=new cpp;
 vptr->getProperties();

 return 0;
}
```

OUTPUT:

my name is cpp

EXPLANATION:

Even though the virtual function is called within the other member function of any class, always virtual function will resolve based on vtable.

Possibly getName function in getProperties will resolve in following manner

```
*(this->vptr[0])()
//Here 0 refers the getName function.
```

2. **Whether virtual function will be allowed to call when it is placed under the private access specifier?**

```
#include<iostream>
#include<array>

class Fruit
{
 public:
       virtual void getColour()=0;
};

class Apple : public Fruit
{
 private:
       virtual void getColour()
       {
             std::cout<<"My Colour is red"<<std::endl;
       }
};

int main()
{
 Fruit *anyFruitPtr=new Apple;
 // NOTE getColour is under private section of Apple
 anyFruitPtr->getColour();
 return 0;
}
```

OUPUT:

My Colour is red

EXPLANATION:

Even though getColour is placed under private section of the Apple class it will be allowed to call when it is a virtual function.

So , why compiler can’t show the compiler error?

Checking the access specifier is part of compilation process and not an run time process.

Just take the statement below,

```
anyFruitPtr->getColour();
```

At run time, anyFruitPtr variable can hold any class object which is child of Fruit class. Since compiler was not aware of actual class in the anyFruitPtr variable, it cannot perform the access specifier validation for the virtual function.

# CLASS AND CLASS RELATED FEATURES

## CONSTRUCTOR & DESTRUCTOR

### CONSTRUCTOR

#### Whether default arguments can be used in constructor?

Yes, default argument can be used in the constructor.

```
MyClass(int a=10){}
```

#### Difference in usage of empty parenthesis

Consider the object is created with non-parameterized constructor,

**Stack memory allocation**

```
MyClass obj; // Valid, ‘obj’ is object of class ‘MyClass’
MyClass obj(); // Valid, but ‘obj’ is not object of class ‘MyClass’, rather obj is
function with return type as ‘MyClass’.
```

**Dynamic Memory Allocation**

```
MyClass *ptr=new MyClass(); // If class contains POD type, then it is default
initialized
MyClass *ptr=new MyClass;
```

#### Converting constructor

Converting constructor is single parameter constructor, which can be called just like assigning the value to the POD types during initialization.

```
MyClass obj=10; // Constructor of MyClass with int argument will be called
```

## COPY CONSTRUCTOR

**Note:**

1. Special member function constructor will not be present when copy constructor is written explicitly

## DESTRUCTOR

Can we call the destructor manually?

Yes, destructor can be called manually. The syntax for calling the destructor is

```
myClassobj.~MyClass();
```

**Virtual destructor**

In a polymorphic class, when derived class is stored in base class pointer and if delete is used on base class pointer, then virtual destructor need to be defined for base class.

Now whenever virtual destructor of base class is called, then it will in turn call the destructor of derived class.

```
~ virtual BaseClass
{
 // Don't worry I will call the derived class destructor
}
```

## CONSTRUCTOR & DESTRUCTOR PUZZLES

1. **Whether constructor will get called when we declare the object for non-parameterized constructor with empty paranthesis?**

```
#include<iostream>

class ArrayNotFound
{
 private:

 public:

        ArrayNotFound(int a)
        {
              std::cout<<"A"<<std::endl;
        }

        ArrayNotFound()
        {
              std::cout<<"B"<<std::endl;
        }
};
```

```
int main()
{
 ArrayNotFound a(10);
 ArrayNotFound b;
 ArrayNotFound c();

 return 0;
}
```

OUTPUT:

```
A

B
```

EXPLANATION:

Constructor will not be called for the following line

```
 ArrayNotFound c();
```

Actually this will not create an object c. C++ compiler will treat this like an function c which returns the ArrayNotFound as return value.

2. **Whether data member of the class can be accessed when object is declared with empty parenthesis?**

```
#include<iostream>

class ArrayNotFound
{
 private:


 public:
       bool learnSomethingNew;

       ArrayNotFound()
       {
             learnSomethingNew=true;
       }
};

int main()
{
 ArrayNotFound a();
 std::cout<<a.learnSomethingNew<<std::endl;

 return 0;
}
```

OUTPUT:

Compilation error

error: request for member 'learnSomethingNew' in 'a', which is of non-class type 'ArrayNotFound()'

EXPLANATION:

Below line will not create object a. This is equal to declaring the function a which returns ArrayNotFound as return value. So the member learnSomethingNew cannot be accessed since object is not created newly.

```
ArrayNotFound a();
```

3. **Whether conversion constructor will act as a conversion operator?**

```
#include<iostream>

class ArrayNotFound
{

 public:
       ArrayNotFound(int a)
       {
             std::cout<<"C";
       }
};


int main()
{
 ArrayNotFound obj=5;
 obj = 10;

 return 0;
}
```

OUTPUT:

CC

EXPLANATION:

It will not give any error even though assignment operator is not overloaded with the integer value.

Conversion constructor will act as a conversion operator.

4. **Whether conversion constructor will acts as conversion operator when assignment operator is overloaded to its specific type?**

```
#include<iostream>
```

```
class ArrayNotFound
{

 public:
       ArrayNotFound(int a)
       {
             std::cout<<"A";
       }

       void operator=(int a)
       {
             std::cout<<"B";
       }
};


int main()
{
 ArrayNotFound obj=5;
 obj = 10;

 return 0;
}
```

OUTPUT:

```
AB
```

EXPLANATION:

No, conversion constructor will not act as conversion operator when assignment operator is overloaded to its specific type

5. **Whether copy constructor will get called when object is returned in the local function?**

```
#include<iostream>


class ArrayNotFound
{

public:
 ArrayNotFound()
 {
       std::cout<<"C";
 }

 ArrayNotFound(const ArrayNotFound& obj)
 {
       std::cout<<"D";
 }

};



ArrayNotFound callme()
{
 return ArrayNotFound();
}


int main()
{
 ArrayNotFound obj = callme();
 return 0;
}
```

OUTPUT:

Output depends on the compiler

GNU compilers output:

C

EXPLANATION:

Here copy constructor is supposed to be called two times,

1. While returning the object in callme function
2. While copying the return value to obj in main function.

   But here compiler hasn't called the copy constructor for even single time. This is because of optimization done by the compiler. Explore more on copy elision technique.

6. **Whether default constructor will be created when copy constructor is written explicitly?**

```
#include<iostream>

class ArrayNotFound
{
 public:

       ArrayNotFound(const ArrayNotFound& obj)
       {
              std::cout<<"CC"<<std::endl;
       }
};
```

```
int main()
{
 ArrayNotFound obj1;
 ArrayNotFound obj2=obj1;

 return 0;
}
```

**OUTPUT:**

Compilation error

error: no matching function for call to 'ArrayNotFound::ArrayNotFound()'

**EXPLANATION:**

Default constructor will not be created when copy constructor is written explicitly.

7. **Whether copy constructor can be defined by taking parameter as value rather reference?**

```
#include<iostream>

class ArrayNotFound
{
 public:

       ArrayNotFound()
       {
       }

       ArrayNotFound(const ArrayNotFound obj)
       {
             std::cout<<"CC"<<std::endl;
       }
};

int main()
{
 ArrayNotFound obj1;
 ArrayNotFound obj2=obj1;


 return 0;
}
```

OUTPUT:

Compilation Error

error: invalid constructor; you probably meant 'ArrayNotFound (const ArrayNotFound&)'

ArrayNotFound(const ArrayNotFound obj)

EXPLANATION:

Copy constructor has to be defined by passing the parameter as reference.

Why it is not allowed as pass by value?

It is because when it is created without reference, then copy constructor will be called infinite number of times since for every call, it will internally call the copy constructor for copying one object to other object.

## 8. Can we access the private members of object which is received as parameter in public constructor?

```
#include<iostream>

class ArrayNotFound
{
 private:
       int iAmPrivate;
 public:

       ArrayNotFound():iAmPrivate(100)     {}

       ArrayNotFound(const ArrayNotFound &obj,int a)
       {
             iAmPrivate = obj.iAmPrivate; // Accessing private member of parameter
object
             std::cout<<iAmPrivate<<":"<<obj.iAmPrivate<<std::endl;
       }
};

int main()
{
 ArrayNotFound obj1;
 ArrayNotFound obj2(obj1,5);


 return 0;
}
```

OUTPUT:

```
100:100
```

EXPLANATION:

Yes we can access the private member of the parameter object if it is used in the constructor.

## 9. What is difference in calling the constructor with and without parenthesis?

```
#include<iostream>
```

```
class Interview
{
 public:
       int m_technical_marks;
       int m_aptidude_marks;
};

int main()
{
 Interview *ptr1=new Interview();
 Interview *ptr2=new Interview;

 std::cout<<ptr1->m_technical_marks<<":"<<ptr2->m_technical_marks;

 return 0;
}
```

OUTPUT:

```
0:GarbageValue
```

EXPLANATION:

POD types used in the class will be default initialized, when constructor is called with parenthesis. Remember, this will apply only to dynamic memory allocation.

## MISCELLANEOUS

### DIFFERENCE BETWEEN CLASS AND STRUCT IN C++

There are few differences between class and structure in C++.

1. When access specifier is not specified manually then all members of the class is private, where as in structure all members are public.

Consider the following example

```
class myClass
{
 int a; // a is private
};

class myStruct
{
 int a; // a is public
};
```

2. During inheritance use of class and struct will make difference in access specifier.

    a. When derived one is specified as class, then public member of the base class is private to the derived class.

```
class myClass
{
 public:
 int a;
};

class Derived : myClass
{
 int b;
};

int main()
{
 Derived d;
 d.a=10; // Compilation error, a is private because Derived is mentioned as class
}
```

    a. When derived one is specified as struct, then public member of the base class is public to the derived class.

```
class myClass
{
 public:
 int a;
};

struct Derived : myClass
{
 int b;
};

int main()
{
 Derived d;
 d.a=10; // a is public because Derived is mentioned as struct
}
```

3. Keyword ‘class’ can be used in place of typename for templates, where as struct cannot be used in place of typename.

```
template <class T>
class myClass
{
 T a;
};
```

## CLASS MISCELLANEOUS PUZZLES

1. **Whether public members of base class is accessible when derived one is struct and no access specifier is specified during inheritance?**

```
#include<iostream>
class BaseClass
{
 public:
 int a;
};

struct DerivedClass : BaseClass // Note here struct
{
 int b;
};

int main()
{
 DerivedClass d;
 d.a=10;
 std::cout<<d.a<<std::endl;
}
```

**OUTPUT:**

```
10
```

**EXPLANATION:**

When struct is specified while deriving the base class , then all public member of base class will be public to the derived class.

Refer Difference between class and struct in c++ for more difference between class and struct.

2. **Whether non-const function can be called by const object?**

```
#include<iostream>

class ArrayNotFound
{
 public:
       void callme()
       {
             std::cout<<"A"<<std::endl;
       }
};

int main()
```

```
{
 const ArrayNotFound obj;
 obj.callme();

 return 0;
}
```

OUTPUT:

Compilation error

accessing_variables_in_scope.cpp:15:13: error: passing 'const ArrayNotFound' as 'this' argument of 'void ArrayNotFound::callme()' discards qualifiers [-fpermissive]

EXPLANATION:

No. Const object cannot be used to call the non-const functions.

Why it is not allowed?

It is not allowed because, non-const functions will allow to modify the members of the object, but intention of const objects is members of the object should not be modified. So it will throw an compilation error.

3. **Can we modify the address of this pointer in member function of class?**

```
#include<iostream>

class ArrayNotFound
{
 public:
       char var='A';
       void callme()
       {
             const ArrayNotFound obj;
             this=&obj;
             this->var='B';
             std::cout<<var<<std::endl;
       }
};

int main()
{
  ArrayNotFound obj;
 obj.callme();

 return 0;
}
```

OUTPUT:

Compilation error

error: lvalue required as left operand of assignment

EXPLANATION:

this pointer cannot be modified in the member function.

Why this pointer is not allowed to modify?

Because this is an purely prvalue. It will be replaced as address of the object during compilation phase.

4. **Can we change the object which is referred by the this pointer?**

```
#include<iostream>

class ArrayNotFound
{
 public:
       char var='A';
       void callme()
       {
             const ArrayNotFound obj;
             *this=obj;
             this->var='B';
             std::cout<<var<<std::endl;
       }
};

int main()
{
  ArrayNotFound obj;
 obj.callme();

 return 0;
}
```

OUTPUT:

```
B
```

EXPLANATION:

Yes we can change the object which is referred by the this pointer.

How it is allowed to change. This pointer is nothing but an address of the object. As you think here object will not be modified.  Below statement calls the overloaded assignment operator for ArrayNotFound class.

```
*this=obj;
```

5. **Whether inner class member function can access the private members of outer class?**

```
#include<iostream>


class outer
{
 int m_outerClassPrivate=10;
public:
 class inner
 {
 public:
       void innerClassMethod(outer a)
       {
              a.m_outerClassPrivate=25;
              std::cout<<a.m_outerClassPrivate;
       }
 };
};

int main()
{
 outer::inner a;
 a.innerClassMethod(outer());

 return 0;
}
```

OUTPUT:

```
25
```

EXPLANATION:

Inner class member functions can access the outer class private members, where as normal class member functions cannot access the private members of the class.

## INHERITANCE

### INHERITANCE PUZZLES

1. **what is the size of the derived class when it inherits the classes which have common base class?**

```
#include<iostream>


/*
Assume the following size
int - 4 bytes
pointer 4 bytes
char 1 byte
```

```
*/


class Language
{
 char a;
};

class C : public Language
{

};

class CPP : public Language
{

};

class Program : C,CPP
{

};

class SpecialProgram : virtual C, CPP
{

};

int main()
{
 std::cout<<sizeof(Program)<<":"<<sizeof(SpecialProgram)<<std::endl;
 return 0;
}
```

OUTPUT:

```
2:8
```

EXPLANATION::

This is called as diamond problem in OOPS concept.

sizeof(Program) returns 2 because Program inherits C and CPP, which have common base class and it is not informed to the compiler.

How to solve this problem?

By using the virtual inheritance.

Even though we used virtual inheritance why it have returned the size as 8 for SpecialProgram?

Since we used the virtual inheritance it will add one hidden virtual pointer inside the SpecialProgram class. Eventhough pointer size is 4, it returned as 8 because we have one char variable in C's base class and size will become 5 now. Due to the structure padding size is returned as 8. One char variable in CPP's base class will be ignored since Language is inherited as part of C's inheritance.

## 2. What happens when we miss the virtual keyword for one class in diamond problem?

```
#include<iostream>


/*
Assume the following size
int - 4 bytes
pointer 4 bytes
char 1 byte
Structure padding 4 Bytes
*/

class m
{
 int b;
 int c;
 int d;
};

class n: virtual m{};

class o:  m{};

class x: n, o {};


int main()
{
 std::cout<<sizeof(x)<<std::endl;
 return 0;
}
```

OUTPUT:

```
28
```

EXPLANATION:

During diamond problem, when virtual inheritance is missed for one class then virtual inheritance will not be applied properly. So size is returned as 28.

For class n, because of class m size is 4+4+4+4=16. (3 int and 1 hidden virtual pointer)

For class o, since virtual keyword is missed, class m size is 4+4+4=12 (3 int)

Ultimately class x size is 16+12=28

When virtual keyword is used in class o then size of x will be 20.

3. **What is size of the empty class which has diamond problem?**

```
#include<iostream>


/*
Assume the following size
int - 4 bytes
pointer 4 bytes
char 1 byte
*/

class Language{};

class C : public Language{};

class CPP : public Language{};

class Program : C,CPP{};

int main()
{
 std::cout<<sizeof(Program)<<std::endl;
 return 0;
}
```

OUTPUT:

```
2
```

EXPLANATION:

Usually when class has no members, then size of the class will be 1. Here due to the diamond problem for Program class, size of the Program class is 2(1 byte for C class due to inheriting the Language class and 1 byte for CPP class due to inheriting the Language class).

## MEMBER FUNCTIONS

### MEMBER FUNCTION PUZZLES

1. **Whether member of base class can be accessed in the derived class when base class member functions are overloaded or overrided?**

```
#include<iostream>


class Language
{

 public:
       void callme(int a)
       {
             std::cout<<"A";
       }
};

class cpp : public Language
{

 public:
       void callme()
       {
             std::cout<<"B";
       }
};

int main()
{
 cpp l_c11;
 l_c11.callme(10);


 return 0;
}
```

OUTPUT:

```
test.cpp: In function 'int main()':

test.cpp:27:17: error: no matching function for call to 'cpp::callme(int)'

  l_c11.callme(10);

                 ^

test.cpp:27:17: note: candidate is:

test.cpp:18:8: note: void cpp::callme()

   void callme()
```

EXPLANATION:

When a member function in derived class is created with same name as base class, then all the base class member function will be hidden to the derived class object.

## MODIFIERS

### MISCELLANOUS MODIFIERS

#### Final

Final is not a keyword in c++. It is just an identifier.

##### Member functions

Class which inherit the class which contain final member functions cannot override the member functions which is marked as final.

#### Override

1. To indicate that current function is overrided member function of virtual function in the base class.
2. Override is not a keyword in c++. It is just an identifier.

##### Why override keyword is required?

1. To avoid errors made commonly by human

```
class Fruit
{
 public:
        virtual void getColour()const
        {
               std::cout<<"My Colour is Fruit"<<std::endl;
        }
};

class Apple : public Fruit
{
 private:
        virtual void getColour() // This is not override function, Note missing
const. Compiler will throw error when override is mentioned.
        {
               std::cout<<"My Colour is red"<<std::endl;
        }
};
```

## OBJECTS

## Intialization related features

### Intializer list

What is initializer list?

### OBJECTS PUZZLES

1. **What is the size of empty class?**

```
#include<iostream>

class ArrayNotFound
{
};

int main()
{
 ArrayNotFound b;
 std::cout<<sizeof(b)<<":"<<sizeof(ArrayNotFound)<<std::endl;
 return 0;
}
```

OUTPUT:

```
1:1
```

EXPLANATION:

Size of the empty object is 1 even though no variable is declared inside the class. This is because when programmer takes the address of the object it should not return the invalid address. To avoid these concern compilers will allocate dummy memory of implementation defined size to the class with no data members and member functions.

### Rule of five:

Whenever programmer defines any one of the following which involves dynamic memory allocation by himself then all of the remaining has to be defined by himself. For Eg, when programmer defines the constructor, then programmer has to define the copy constructor, destructor, e.t.c.

1. Constructor
2. Destructor
3. Copy constructor
4. Move constructor

When expression if of type rvalue then instead of copying compiler will transfer the ownership

5. Move assignment

## Object Slicing

Upcasting the derived class to base class is called object slicing

```
Derived d;
Base b=static_cast<Base>(d);
```

## Casting beween objects:

### upcast

Converting to base class from derived class is called upcast

```
Derived *d=new Derived;
Base *b=static_cast<Base*>(d);
```

### downcast

Converting to derived class to base class is called downcast. It is mainly used in the virtual functions.

```
base *b=new Derived;
Derived *d=dynamic_cast< Derived *>(b);
```

## Instantiating the inner class in c++

Instantiating the inner class is same as normal class instantiation. Follow the sample code for better understanding.

```
#include<iostream>

int a;

struct test
{
   int a;
   struct inner
   {
      void callme()
      {
         a=0;
         printf("\n  :: %d ", a );
         printf("\n  :: %d ", ::a );
      }
   };
};
```

```
int main()
{
   test::inner obj;
   obj.callme();
   return 0;
}
```

## Exception:

Stack unwinding will happen when exception occurs in program.

## RAAI

Resource Acquisition is initialization

## CADRe

Constructor allocates and destructor releases.

## Instantiation:

### Classification based on time of instantiation:

#### early instantiation:

Object is instantiated before the actual usage of the object in the program. For example object is created at some place and it is used only after some time.

#### LAZY instantiation:

Object is instantiated only when it is required.

## Wrapper Class

Class which provides an interface to some functionality is called wrapper class. Wrapper function is same as wrapper class.

Wrapper class is also called as adaptor class. This is similar to broker, which acts as intermediate between the caller and functionality.

## Casting

### Static cast

It checks whether conversion between specified types is valid.

**They what makes static cast different from other cast?**

Static cast ensures that whether conversion between the types is valid. But static cast does not perform run time check. It will perform during compile type statically. So it is called as static cast.

| TYPE1 | TYPE2 | IS VALID |
|---|---|---|
| Char* | int* | NOT VALID |
| Int* | Char* | NOT VALID |
| | | |

## Copy constructor in c++

**What is copy constructor?**

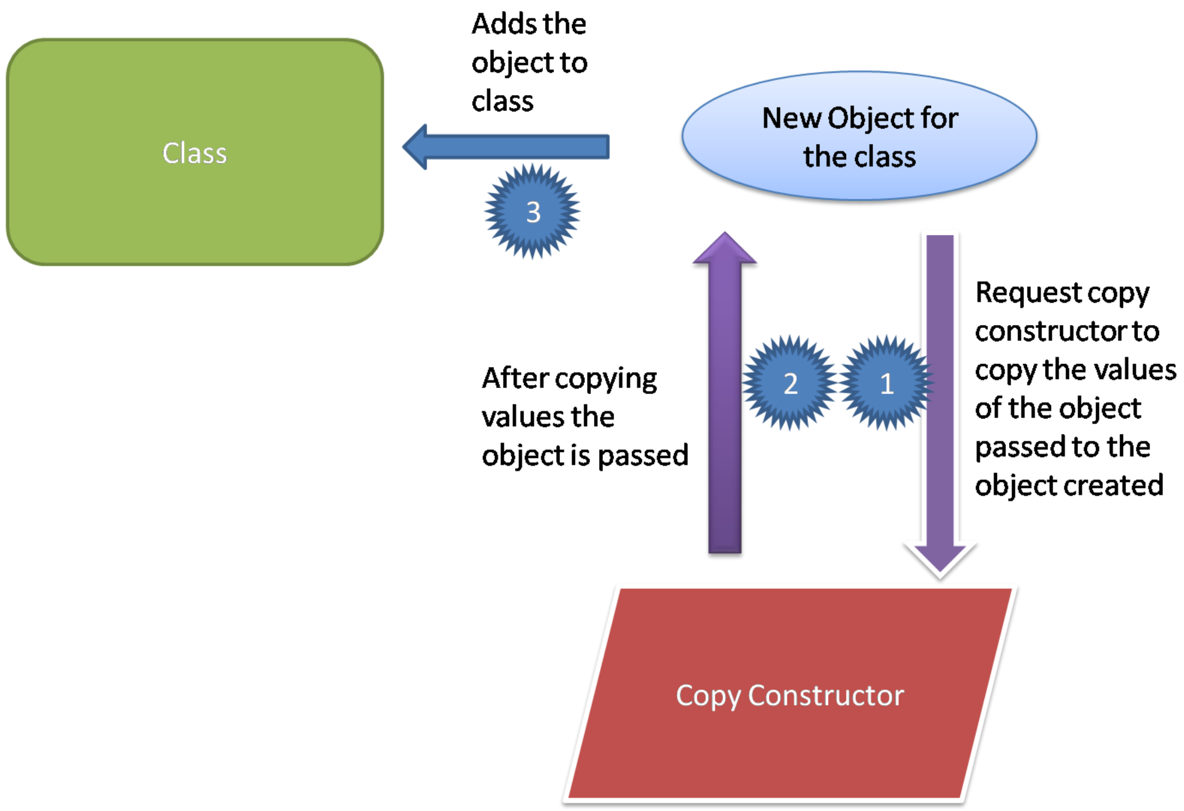
As the name suggests the constructor that used to copy the value of the one object to the other object while creating is called as copy constructor . So we are calling it as a copy constructor. Copy constructor, if not defined by the programmer then compiler will create its own.

In this case normal constructor will not be called and as normal constructor copy constructor will not have return type and have same name as class name.

In other words,

what is constructor? It is used to initialize the objects .

what is copy constructor? It is used to initialize the objects with other objects while creating it.


Adds the object to class
Class
3
New Object for the class
2
1
After copying values the object is passed
Request copy constructor to copy the values of the object passed to the object created
Copy Constructor

**When copy constructor is called ?**

when you pass an "**object as a value**" copy constructor is called. Don't just confuse with the object as value. It is same as call by value. Here instead of variable we were using the objects.

```
Fruits apple;  //Fruit is class, apple is object
callme(apple);  //object as value
```

The above code is similar to the following….

```
int a;
callme(a); //variable as value
```

I hope you have understand the difference between these two. so, coming to our topic copy constructor is called whenever you pass an object as a value.

And another important noting point is the copy constructor will not call again and again when we use object as a value. It will call only at  the time of object creation as above mentioned example.

when you **return a object in function**, the copy constructor will be called.

```
Fruits callme(Fruits apple)
{
 …..
       return apple;   // returning object of class Fruit
}
```

Next whenever you initialize the object with other object the copy constructor will be called.

```
Fruits apple, orange;
apple=orange; // Initializing orange object with apple
Fruits pinapple(apple); //Here too copy constructor called
```

**Syntax for copy constructor :**

```
class myclass
{
       myclass (const myclass &obj_passed)
               {
             //copy constructor code
             //create a new copy of object here
       }
 }
```

In above syntax :

obj_passed is the reference of the object from which we are going to copy the value. Note the const keyword in parameter.

**What happens if copy constructor is not created by user?**

So, are you wondering what happens if copy constructor is not created by the user. You just don't worry compiler will create it. Even if copy constructor is not defined the following code will work.

```
Fruit apple=orange;
```

where orange and apple are objects and fruit is class. Here we were assigning a orange object with the apple. In a compiler defined copy constructor, initialization of the object takes place "member wise". This process is called is called as **Shallow Copy**. Hope you understand member wise, Data members of the source object will be copied to destination object's Data members.

**Where the copy constructor is must?**

When we fail to create a copy constructor the compiler will create it by own. In such cases if member of the object is pointer then address will be copied in the object created by the copy constructor. So object created by class and a copy constructor will share a same memory location. In such case when destructor is called, no problem for the 1st object, memory will be deallocated but in the second case the application will crash. So in case of pointers the copy constructor is must.

## CONSTRUCTOR PUZZLES

1. **How order of object construction is decided when more than one class object is kept as member of the class?**

```
#include<iostream>
class Comment
{

public:
 Comment(int a)
 {
       std::cout<<"Comment constructor called"<<std::endl;
 }
};

class Bugs
{

public:
 Bugs(double a)
 {
       std::cout<<"Bugs constructor called"<<std::endl;
 }
};


class Post
{

public:
 Post(): c(35), b(25.25)
 {

       std::cout<<"Post constructor called"<<std::endl;
 }

private:
 Bugs b;
 Comment c;
};

int main()
{
 Post p;

 return 0;
}
```

OUTPUT:

```
Bugs constructor called

Comment constructor called

Post constructor called
```

EXPLANATION:

The order of construction of objects is decided based on the order of declaration in the class. It is not decided based on the way we call the objects constructor in the initialization list.

Here b is declared first in the class, so bugs constructor is called even though comment is first initialized in the post constructor.

2. **Which constructor is called when object is declared using empty initialize list?**

```
#include<iostream>
#include<initializer_list>
```

```
class ArrayNotFound
{
 public:
       ArrayNotFound()
       {
             std::cout<<"DC"<<std::endl;
       }

       ArrayNotFound(std::initializer_list<int> p_obj)
       {
             std::cout<<"IC"<<std::endl;
       }
};

int main()
{
 ArrayNotFound obj{};
 return 0;
}
```

OUTPUT

```
DC
```

EXPLANATION

**CLASS PUZZLE**

1. **What happens, when an initializer variable is present in more than one scope and if class member is initialized with such kind of intializer?**

```
#include<iostream>

class ArrayNotFound
{
public:
 static int a;
 static int b;
};

int a=10;
int ArrayNotFound::a=25;
int ArrayNotFound::b=a;

int main()
```

```
{
 std::cout<<"ArrayNotFound::b : "<<ArrayNotFound::b;
 return 0;
}
```

OUTPUT:

```
ArrayNotFound::b : 25
```

EXPLANATION:

When a class member is defined outside the class and if the intializer variable which is used for initializing the class member is present in more than one scope, then the class member will be initialized with the variable which belongs to the own class member scope.

I think it is bit confusing, Let me explain with the following example,

Consider the following statement,

```
int ArrayNotFound::b=a;
```

When you look in to this, class member 'b' is defined outside the class and it is initialized with variable 'a'. Here a is present in two scopes,

a. Global scope

```
int a=10;
```

b. Class scope

```
static int a;
```

So compiler will get confuse now, and it will follow as per the standard rule.

It means it will pick up the class scope a instead of global variable a, because static variable 'b' belongs to the ArrayNotFound scope.

## ENUM PUZZLE:

1. **What happens when we use the negative value in the enum value?**

```
#include<iostream>

enum NegativeEnum
{
 MinValue=-5,
 MaxValue,
};
```

```
int main()
{
 std::cout<<"MaxValue: " <<NegativeEnum::MaxValue<<std::endl;
 return 0;
}
```

OUTPUT:

```
-4
```

EXPLANATION:

Enum in C++ is allowed to initialize with negative values.

Now, what happens to next enum value when we initialize the enum with negative value?

Simple, it will be incremented by one.

Here MinValue value is -5, now next value MaxValue is (-5+1)=-4

## EXCEPTION IN C++

Exception is one which is used to handle the unexpected situations.

### need for exception:

While coding small programs exception case may find not useful. At the same time while developing the huge applications exception is a good friend to the programmers.

Consider the scenario, that you are using the database in your program, you cannot be sure that database connectivity will always be there. At sometimes the database connectivity may go down. As a programmer we need to handle this situation. Here exception is useful.

So write a exception here, whenever the exception occurs display the alert message.

If exception is not handled properly then our program will crash and stops running.

### try throw catch blocks:

### try block:

Whenever you feel the exception occur place the code in try block.

```
try
{
 // Code for which exception occurs
}
```

### catch block:

Whenever error occurs, catch block will be called

```
catch(int a)
{

}
```

### throw block:

Whenever you feel the exception will occur, use a throw statement. Whenever the throw statement is encountered then the catch block will be called.

### throw after the funciton signature:

The throw after the method signature is used for unhandled exceptions. If you feel that some exceptions are no need to be handled by the function you can use the throw after the method signature.

There will be no catch block will be called for that throw statement.

The exception that is mentioned in the type list only will be thrown. other exceptions will not be handled if exception occurs then the program will crash.

```
void myfunction(int a)throws (int a, std::logic_error)
{
---
}
```

In above code, only int error and logic error that was caught by program will be thrown. It means you no need to throw anystatment for int and logic error. It is automatic. when other error occurs the program will crash.

One sad news is that this one was depreciated in c++ 11.

http://stackoverflow.com/questions/21012430/what-is-the-purpose-of-using-throw-after-fuction-signature-in-c/21012715?noredirect=1#comment31605920_21012715

### ellipis operator in catch block:

Triple dot "…" is the ellipsis operator. This operator denotes that any number of argument and various kinds of datatye.

```
catch(…)
{
}
```

This catch block will accept the more than one type of argument here. I.e., this catch block will accept integer throw, logical error throw and so on. You you want to define the generic catch statement you can use this one.

## STANDARD LIBRARY

### STANDARD TEMPLATE LIBRARY(STL)

#### CONTAINER CLASSIFICATION

| SEQUENCE CONTAINER | ASSOCIATIVE CONTAINER | ADAPTER CONTAINER |
|---|---|---|
| Array | Map | Stack |
| Vector | Multi-map | Queue |
| De-queue | Unordered map | Priority_queue |
| List | Unordered multi-map | |
| Forward_list | Set | |
| | Multi-set | |
| | Unordered set | |
| | Unordered multi-set | |
| | | |
| | | |

#### CONTAINER CORRECT USAGE

**Factors influencing in choosing correct STL**

1. Order is important
2. Last in first out or first in last out
3. Finding element by key
4. Need to merge collection
5. Storing key to separate one
6. Allowing duplicates

7. Insert erase at middle
8. Insert or erase at front
9. Size will vary widely

**KEY**

- NR-NOT RECOMMENDED
- R-Recommended
- M – Maintained
- NM-Not maintained
- NA- Not applicable

| CONTAINER | FREQUENT INSERTION | FREQUENT DELETE | SEARCH | INSERTION ORDER OF ITEMS | POSSBILE INTERNAL REPRESENTATION | Random access | ADAPTION |
|---|---|---|---|---|---|---|---|
| VECTOR | NR | NR | NR | M | Static Array | Possible | OWN |
| MAP | R | R | R | NM | RB Tree | Not Possible | OWN |
| LIST | R | R | NR | M | Double linked list | Not Possible | OWN |
| SET | R | R | R | NM | RB Tree | Not Possible | OWN |
| DEQUEUE | R | NR | NR | M | Array to Array. Master array maintains the list of array. | Not Possible | OWN |
| QUEUE | NR | NR | NR | M | Depends on the Adaptor | Not Possible | Container adaptor |
| ARRAY | NA | NA | NR | M | Array | Possible | OWN |
| FORWARD LIST | R | R | NR | M | Singly linked list | Not possible | OWN |
| STACK | R | NA | NA | M | Depends on the Adaptor | Not possible | Container adaptor |
| UNORDERED MAP | R | R | R | NM | Hashing | Possible | OWN |
| UNORDERED SET | R | R | R | NM | Hashing | Possible | OWN |
| PRIORITY QUEUE | R | NA | NR | NM | Depends on the Adaptor | Not possible | Container adaptor |
| MULTIMAP | R | R | R | NM | Red black tree | Not possible | OWN |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| MULTISET | R | R | R | NM | Red black tree | Not possible | OWN |
| UNORDERED MULTI MAP | R | R | R | NM | Hashing | Possible | OWN |
| UNORDERED MULTISET | R | R | R | NM | Hashing | Possible | OWN |

| CONTAINER | Deleting any element in container | Position based Access | | | |
|---|---|---|---|---|---|
| VECTOR | Possible | Possible | | | |
| MAP | Possible | Not possible | | | |
| LIST | Possible | Not possible | | | |
| SET | Possible | Not possible | | | |
| DEQUEUE | Possible | Possible | | | |
| QUEUE | Not possible | Not possible | | | |
| ARRAY | Not possible | Possible | | | |
| FORWARD LIST | Possible | Not possible | | | |
| STACK | Not possible | Not possible | | | |
| UNORDERED MAP | Possible | Possible | | | |
| UNORDERED SET | Possible | Possible | | | |
| PRIORITY QUEUE | Not possible | Not possible | | | |
| MULTIMAP | Possible | Not possible | | | |
| MULTISET | Possible | Not possible | | | |
| UNORDERED MULTI MAP | Possible | Possible | | | |
| UNORDERED MULTISET | Possible | Possible | | | |

## TIME AND SPACE COMPLEXITY OF STL

| CONTAINER | INSERT | DELETE | SEARCH |
|---|---|---|---|
| Array | O(1) | NA | O(N) |
| VECTOR | O(1) for back and O(N) for others | O(1) for remove O(n) for erase | O(N) |
| MAP | O(log N) | O(log N) | O(log N) |
| LIST | O(N) | O(N) | O(N) |
| SET | O(log N) | O(log N) | O(log N) |

| | | | |
|---|---|---|---|
| DEQUEUE | O(1) | O(1) | O(N) |
| QUEUE | O(1) | O(1) | O(N) |
| FORWARD LIST | O(1) | O(N) | O(N) |
| STACK | O(1) | O(1) | NA |
| UNORDERED MAP | O(1) | O(1) | O(1) |
| UNORDERED SET | O(1) | O(1) | O(1) |
| PRIORITY QUEUE | O( log N) | O(log N) | O(N) |
| MULTIMAP | O(log N) | O(log N) | O(log N) |
| MULTISET | O(log N) | O(log N) | O(log N) |
| UNORDERED MULTI MAP | O(1) | O(1) | O(1) |
| UNORDERED MULTISET | O(1) | O(1) | O(1) |

## TERMINOLOGY

| Terminology | Description |
|---|---|
| Insert | Sorts the element while inserting and it rearranges the order of the element in container |
| Push_back | Add the element without sorting and rearranging the order of element in container |
| Associative container | Look based on key rather than absolute position |

## GENERAL STL INTERVIEW QUESTIONS

1. **How capacity of the standard template library is incremented?**

```
#include<iostream>
#include<vector>

int main()
{
 std::vector<int> a(10);
 std::cout<<a.size()<<":"<<a.capacity()<<std::endl;
 return 0;
}
```

**OUTPUT:**

```
10:<Implementation defined value>
```

**EXPLANATION:**

Size of the vector will grow based on the implementation defined value.

Standard haven't said about how much vector size should be incremented when it reaches the maximum capacity. It depends on the implementation, how size has to be incremented.

## UTILITY FUNCTIONS

### Functor

Functor is the function object.

### Lambda function

Lambda function is an unnamed temporary function defined as inline in code without name.

Syntax:

```
[localvariable that need to be accessed](return_type)
{
// Operation to be performed
}
```

### Accessing all local variable inside lamba function

ACCESSING ALL LOCAL VARIABLES AS REFERENCE

```
int n;
[&]() {n=10};
```

ACCESSING ALL LOCAL VARIABLES AS VALUE

```
int n;
[=]() {n=10;}
```

All Local variables are accessed as const by default. To remove the constness use keyword as mutable

```
int n;
[=]() {n=10}(); // This statement will give error since n is const by default

[=]()mutable {n=10}
```

calling the lambda function:

To call the lamda function use the function parenthesis at the end

```
[=]()mutable { printf("Hello Lambda"); }();
```

### for_each function

It will iterate through each element in the container and corresponding functor need to be written to perform some operation on the value of the container.

For Eg, If you are storing dynamically created objects in the map container and you can release the memory of all elements stored in the container using the for_each function.

```
for_each(beginiterator,endIterator, functor)
```

Lambda function for for each:

```
for_each(beginiterator,endIterator, [](int a){printf("%d",a) }
```

Here a is an value type of the class pointed by the iterator.

## MAP IN C++

Maps are associative containers.

wondering what is associative container?

Associative container is group of templates that implement the ordered associative arrays.

Now wondering what is associate arrays?

Associative arrays are nothing but the name value pairs. i.e., There will be lots of keys and for each key there will be one value.

There are several other things related to this such as map,multimap,set, multiset etc.,

Maps are implemented using a binary search trees.

### INTERNAL PROCESSING OF MAP:

Internally map sorted it elements by its key using a strict week ordering. strict week ordering is nothing but an a < b == true and b < a == true, but b != a

Map interally uses an RB(Red-Black tree) for manipulating its key and pair.

### PROPERTIES OF MAP:

- ✓ In maps we have to access the value using the keys. We cannot access the value using the containers.
- ✓ Keys in map should be unique. Contents of the keys can be updated at any time if needed
- ✓ Nested maps are possible

### SYNTAX:

```
#include<map>
map<key_datatype,value_datatype>map_name;
```

In above prototype the key_datatype and value_datatype may be anything. It can be primitive data type such as int, char or classes or struct.

Eg:

```
map<int,double>average;
```

In above one, the key is of type int and value of type double some other possible declaration are

```
map<int,my_struct>my_map;
map<int,my_class>my_map;
map<char,double>my_map;
```

## ACCESSING CONTENT OF MAP:

### ARRAY LIKE ACCESS:

This method access the content of the map using the key. Here pass the key inbetween the square bracket where the content of the key will be retrieved.

```
printf("\n Average of rollno 1 is %f ",average[1]);
```

## INSERTING CONTENT TO MAP:

There are two ways to insert in map,

- Array like insertion
- using insert method

ARRAY LIKE INSERTION (overloading of [] operator):

The map content can be accessed as array method

```
average[141]=50.545;
```

Here the roll no 141 is assigned with average 50.545; This implementation in done in map using the operator overloading.

This insertion method will search for the key, if key exists then the corresponding value will be updated. If key does not exists then it will create a  new key and insert the value.

Internally, it will search for the key if key present then it will return the reference to the key, for inserting the new value.

Array like insertion for non-primitive types:

Insertion using primitive data type will work fine.

What happens to this case

```
map <Node1,Node2> my_map; // Node 1 and 2 are structs
my_map[Node1]=Node2;
```

Here error will be thrown. Since the map uses the strict week ordering it needs to compare before it insert in to the map. How it will compare the Node1 and Node2. Both are of struct type. So all we need to do is, write the operator overloading function for < operator.

```
bool operator<(const Node &Node1,const Node &Node2)
{
// Your stuff
}
```

Here two parameters should be passed. It need not be of type const. Since we are using the reference variable we have using the const here. Same will be applied for using class as key. Always bare in mind that both the parameter for the overloading the < operator is an key, Value will not come to picture for this case. Consider the above code we have used two parameter Node1 and Node2 as struct which is of type key and should not be of type value. And then the operator overloading function here is called when there was two elements in the map.

Disadvantages of array like insertion:

When a key that you were inserting exists already, then key will be overwritten.

USING INSERT METHOD:

```
average.insert(pair<int,double>(2,45.24));
```

Here the pair is template class. Which calls the pair constructor. Here the insert method always takes only one argument. Here the pair template class create a reference by combining the key and value.

DIFF BETWEEN ARRAY LIKE INSERTION & USING INSERT METHOD:

The basic difference between the two types of insertion is array like insertion overwrites the existing key where as the insert method does not overwrite the key already used.

## UTILITY FUNCTIONS IN MAP:

### FIND SIZE OF MAP:

You may ask why there is a need for map? Can the above can be implemented in struct. This is because, the index of struct can be only int. But here the map gives you flexibility. You can have any datatype. Also searching content in maps is little bit powerful.

### Insert_or_assign()

Used to insert or assign the value to the existing key

## VECTOR

Vector is one of the sequence container in c++. The sequence container is nothing but the container which is used to store the data in a sequential fashion.

Need for Vector:

1. Array does not support flexible addition and deletion of members
2. storage size need to be handled by the programmer in case of array. whereas the vector handles it automatically.

   Disadvantages of vector:

   1. Size of the vector is allocated more than the required size to handle the member addition and deletion. So memory taken by the vector will be comparatively more when compared to the array

### VECTOR PUZZLES

.

## ITERATOR IN C++

Iterator in c++ was used to point the some data in the container class. Most of standard Template library(STL) supports the iterator class.

### Arithemetic operations on iterator:

Itertor supports

Iterators mostly declared as member of the class which is it implemented

### SET

1. set interally uses an RB(Red-Black tree) for manipulating its key and pair.

2. Cannot modify the element in the set

## BIT FIELD

### BIT FIELD PUZZLES

1. **What is the size of class when bitfield is used in the class?**

```
#include<iostream>
/*
Assume the following size
int - 4 bytes
pointer 4 bytes
char 1 byte
*/

class ArrayNotFound
{
 public:
       int a:8;
       int b:16;
};

int main()
{
 ArrayNotFound obj;
 std::cout<<sizeof(int)<<":"<<sizeof(ArrayNotFound);
 return 0;
}
```

OUTPUT:

```
4:4
```

EXPLANATION:

Usually int will take 4 bytes(Assumption) when it is declared normally in class. Since this class contains 2 int, then size of class should be 8 bytes. It is not 8 bytes because we have used bitfield in class.

```
int a:8;
int b:16;
```

Here, a takes 1 byte(8 bits) and b takes 2 bytes (16 bits) and sum is 3 bytes. But we have got the result as 4 bytes for class. This is because padding rule is applied in struct.

2. **What happens when arithemetic overflow occurs for bit field?**

```
#include<iostream>
/*
Assume the following size
int - 4 bytes
pointer 4 bytes
char 1 byte
*/

class ArrayNotFound
{
 public:
       int a:3; //3 bits
};

int main()
{
 ArrayNotFound obj;
 obj.a=2;
 std::cout<<obj.a<<":";
 obj.a=3;
 std::cout<<obj.a;
 return 0;
}
```

OUTPUT:

When arithmetic overflow occurs in the bitfield, then it will start again from the first value.

Here a is signed integer data type with 3 bits. 1 bit for sign value and 2 bits for data value. So range will be (-4 to 3). Here assigning 2 will be stored properly, but since 3 is out of range it will give -4 as result.

## UTILITY FUNCTIONS

### FILL FUNCTION IN C++

Fill is a template function used to initialize array or vectors using the some value. You should specify the starting pointer , number of elements to be reseted and value to be reseted.

Fill is the standard c++ function comes under the namespace std.

You can use fill to fill the array from intermediate.

**SYNTAX:**

```
void fill(type *start,type *end,const int reset_value);
```

**SAMPLE PROGRAM:**

```
#include<iostream>

int main()
{
   using namespace std;

   char arr[]="123456";
   printf("\n     arr    :: %s ",    arr       );
   fill(arr,arr+5,'a');
   printf("\n     arr    :: %s ",    arr       );

   return 0;
}
```

### LIMITATIONS ON FILL FUNCTION:

you can reset the value in sequence only

This is similar to for loop internally

use of  memset is efficient when compare to fill but memset works on bytes level.

### PROGRAMMING USAGE:

Used to initialize the array

used for filling some range of value in the array

## STRING

String is implemented as class in c++.

### Size_type

std::string::size_type  is alias of size_t in string class

size_t is unsigned integral type.

### npos

std::string::npos is the maximum value of the size_t.

The definition of npos is

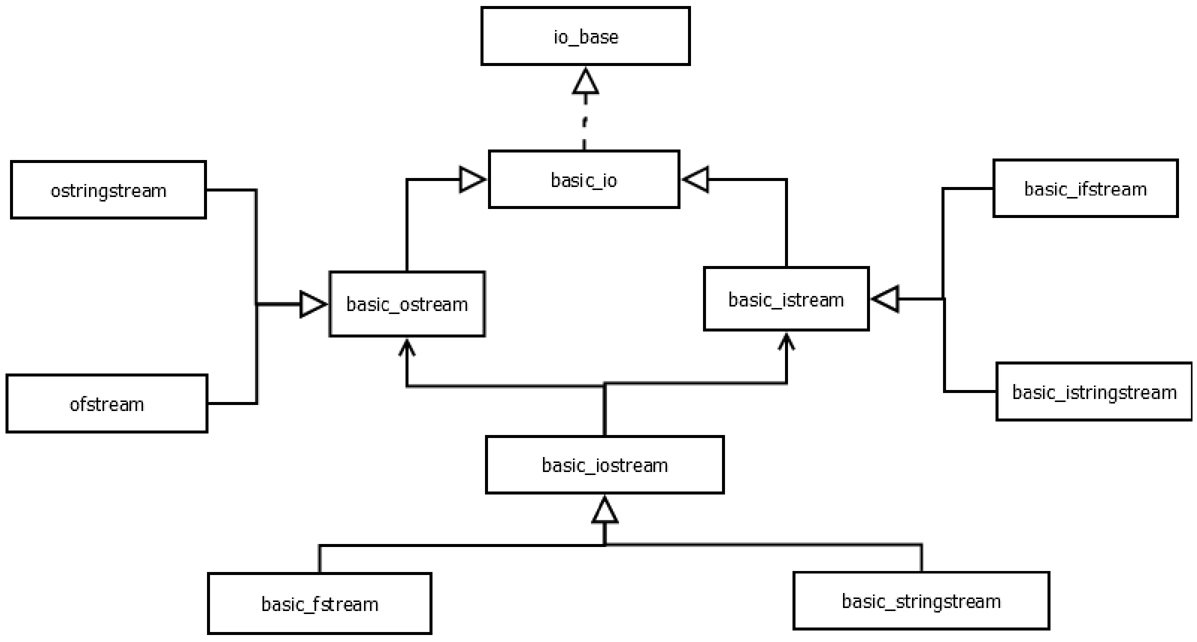
```
static const size_type  npos = -1;
```

Actually the npos is unsigned integer because its datatype was size_type(alias of size_t).

Then why it is initialized to -1. Our aim is to initialize the npos to maximum value. Here assigning -1 to unsigned integer will

### Input output class

### Class Structure



## Nested class

### Why nested class?

1. To hide the implementation details to others

## TEMPLATE

Apart from programming, template is one, which just defines the outline. Consider the leave letter template, which does not contain much information. As a template it contains only from address column but it does not have a from address. similarly it has a body section but it does not have for what purpose the leave is taken.

In c++, templates are much more similar to that. Here, in class template only the class name will be given while declaring. But which datatype the class uses will not be mentioned. This is more generic way of programming.

Then you may ask why there is need for template class?

Here the class does a same functionality for all datatypes. If template class was not there then you have to define the multiple class for each datatype that the class uses. But using templates we are creating only one class.

Real time example:

Consider the Arithemetic operation class. The datatype for the Arithemetic operation can be int, float, double. Inorder to define the multiple class we can just create the single template class and we can make specific whenever we are needed.

Syntax:

```
template <class type1> class template_class_name;
```

Eg:

```
template <class type1> class arithmetic_oper;
```

In above syntax, the template is keyword just to indicate the class is of type template. Then another new thing is <> brackets. while instantiation, the datatypes that are going to be used are mentioned there. In above we can allowed to instantiate using one type but we can give many as <class type1,class type2>.

## DEFINING THE TEMPLATE CLASS:

This is just as normal class definition. In normal class definition you will be using the datatype as int. float etc for declaring the variable. But here you will be using the typename defined inside the angle brackets.

here type1 is identifier. This type1 can be int,float,double etc., This is decided while creation of object for the class.

```
template <class type1>class arithmetic_oper
{
   public:
      type1 variable_name;
       int fixed_type_int;
};
```

## DIFFERENCE BETWEEN CLASS AND TYPENAME IN TEMPLATE:

There is no major technical difference between the class and typename. we can use them interchangeably. But at one rare case it is different. Need to explore that.

Template parameter can be anything. It can be type(int,float), non-type(values), or template

## CREATING OBJECT FOR TEMPLATE CLASS:

```
arithmetic_oper <int> int_operation;
```

Here the arithmetic_oper is template class name. Here I want to make the class to use the int variable. so I have used int inside the angle brackets. I cannot give more than one type here. If I want to give more than one type then I need to declare the template class as <class type1,class type2>

## CREATING TEMPLATE FUNCTION INSIDE THE  TEMPLATE CLASS:

Declartion of function template with new type inside the class template goes like this

```
template <class F_Type1> F_Type1  add(F_Type1,F_Type1);
```

Here the new type F_Type1 is created .

And the definition of template function goes like this

```
template <class c_Type1>
template <class type1>
type1 arithmetic_oper<c_Type1> ::  add(type1 a,type1 b)
{
   return a+b;
}
```

Here consider this, We have to specify the both templates such as class template c_Type1 and function template type1.

## TYPE CASTING OF FUNCTION TEMPLATE:

consider the following snippet,

```
addition_value=(int)int_operation.add<int>(5.10,10);
```

Here the <int> represent the type casting to parameters. Here both parameter were different. If it was same, then compiler will auto deduct it. we are confusing the compiler in deducting the parameter type. So we explicitly indicate the compiler regarding parameter types. Then (int) is normal type casting. It stores the resultant value in to addition_value variable by typecasting to integer.

## SAMPLE PROGRAM:

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

template <class type1>
class arithmetic_oper; //declaring template

template <class type1>
class arithmetic_oper //defining template
{
   public:
      type1 variable_name; //Here we can change the datatype
      int fixed_type_int;  //cannot change the datatype

      // Using different type for function template
      template <class F_Type1> F_Type1  add(F_Type1,F_Type1);

      //Using same type as class for function template
      type1 sub(type1,type1);

};

template <class c_Type1>
template <class type1>
type1 arithmetic_oper<c_Type1> ::  add(type1 a,type1 b)
{
   return a+b;
}


template <class c_Type1>
c_Type1 arithmetic_oper<c_Type1> :: sub(c_Type1 a,c_Type1 b)
{
   return a-b;
}


int main()
{

   //Creating object as int using template
   arithmetic_oper <int> int_operation;
   int_operation.variable_name=10;
   printf("\nUsing a class as a integer :: %d ",int_operation.variable_name);
   int addition_value=int_operation.add(5,10);
   printf("\nADDITION OF GIVEN VALUE IS %d",addition_value);
   addition_value=(int)int_operation.add<int>(5.10,10); // Type casting as int.
   printf("\nADDITION OF GIVEN VALUE IS with type casting %d",addition_value);


   //Creating object as double using template
   arithmetic_oper <double> double_operation;
   double sub_value=double_operation.sub(50.52,25.02);
   printf("\nSUBTRACTION OF GIVEN VALUE IS %f",sub_value);

   //Creating object as char using template
   arithmetic_oper <char> char_operation;
   char_operation.variable_name='A';
   printf("\nUsing a class as char type :: %c ",char_operation.variable_name);

   return 0;
```

```
}
```

**OUTPUT:**

```
Using a class as a integer :: 10
ADDITION OF GIVEN VALUE IS 15
ADDITION OF GIVEN VALUE IS with type casting 15
SUBTRACTION OF GIVEN VALUE IS 25.500000
Using a class as char type :: A
```

**How template function  or class is works internally?**

Consider the following template function in your program

```
template<class T>
void printme(T a)
{
 std::cout<<a<<std::endl;
}
```

Now assume that you have called this function 3 times in your program as follows,

```
printme<int>(5);
printme<int>(48);
printme<double>(4.54);
```

Here template function printme is used for types int and double in the program. So compiler will internally create an two functions printme as follows

```
void printme(int a)
{
 std::cout<<a<<std::endl;
}

void printme(double a)
{
 std::cout<<a<<std::endl;
}
```

## UTILITY

**is  same template class:**

It is used to find whether two types are equal. It is mainly used to check whether received types are same.

```
#include<iostream>
#include<utility>

template<typename T1,typename T2>
class myClass
{
 public:

 static void callme()
 {
       std::cout<<std::is_same<T1,T2>::value<<std::endl;
 }
};

int main()
{
 myClass<int,int>::callme();

 return 0;
}
```

## UTILITY PUZZLES

### 1. Whether objects can be used to compare the two things in is_same template class?

```
#include<iostream>
#include<utility>

template<typename T1,typename T2>
class ArrayNotFound
{
 public:

 static void callme()
 {
       int a;
       float b;

 std::cout<<std::is_same<T1,T2>::value<<":"<<std::is_same<a,b>::value<<std::endl;
 }
};

int main()
{
 ArrayNotFound<int,int>::callme();

 return 0;
}
```

OUTPUT:

Compilation Error

error: type/value mismatch at argument 1 in template parameter list for 'template<class, class> struct std::is_same'

EXPLANATION:

Only data type should be used to compare two things in is_same utility. So, why objects should not be used in the is_same utility template class. Possible implementation of is_class can be

```
template<typename T1,typename T2>
class is_same
{
};
```

Since objects cannot be passed as template parameter, it will give compilation error.

## YET TO CATEGORIZE

using the template feature in c++, you can define the functions and classes in a generic type. The generic type means, you can use a any datatype. By defining a function or class for generic type you can use it for a multiple datatype. It provides the code reusability.

There are two kinds of template.

Function template

class template

### Impact on function overloading:

If template concept is not introduced, then we have to use the function overloading concept because we need to define the function for each data type. Because of template we are using a generic type.

### Class template:

Class template is specifically designed for the container clas

/*** need knowledge of container class to proceed

### function template:

Function template is used for defining the generic parameters.

template<typename T>

void callme(T a,T b)

```
{
        T c=a+b;
}
```

Here the function contains the two parameter a and b, whenever the function is called it implicitly takes as datatype of the  arguments passed. For eg if you call function callme as callme(5,6). Then T will be of type int. If it is called as a callme(1.2,5.5). Then it will take the datype as float. It is not possible to give a mixed datatype. For eg callme(5,2.2).

| sample call | Descrption |
|---|---|
| **callme(1,2)** | Possible. It will take T as int at runtime |
| **callme(5.2,6.2)** | Possible. It will take T as float at runtime |
| **callme(2,5.6)** | Not possible. Returns a error. |

# C++ MISCELLANEOUS

## DIFFERENCE BETWEEN C AND C++: MORE THAN YOU KNOW

### USE OF elaborated type specifier:

In C, it is compulsory to use a elaborated type specifier for using the struct, union, and enums.

So are you wonder what is elaborated type specifier am I correct?

ELABORATED TYPE SPECIFIER:

While creating the object for the struct , you need to use a struct keyword along with the type name. (strcut s obj;) here struct is keyword s is typename obj is object. This process of creating the object using type name and their corresponding keyword is called as elaborated type specifier.

i.e, we cannot use like following,

```
struct s
{
---;
}
```

```
s obj;// Invalid in c, but valid in c++
```

Finally, In C we have to use ( struct s {} as struct s obj; ) and in c++ (s obj; )But using elaborated type specifier is also valid in c++.(struct s obj)

In similar to the above discussion, in c++ we are using class c{} as c obj;

### CALL BY REFERNCE:

There is no reference variable in c. For call by reference we need to pass the pointer in c, where as in c++ we can use the reference variable.

## DIFFERENCE BETWEEN C++11 AND BELOW VERSION

C++ 11 allows variadic template . But lower version will allow only the variadic functions.

## CORE DUMPED (SEGMENTATION FAULT)

Illegal access of memory or uninitialized pointers. i.e, usually occurs because of dynamic memory allocation

Any attempt that made to change the read only memory will cause a segmentation fault

### 1.changing and accessing the value of the null pointer

If you change or access the value of the null pointer then the core dump will occur. consider the scenario

```
int *p=NULL;
*p=10; // core dump
cout<<*p; //core dump
```

In above case you are changing the value of the null pointer. This invalid access of memory. so core dump will occur at both scenario. But in the following scenario

```
int *p;
*p=10; // core dump
cout<<*p; //core dump
```

The code may or maynot throw core dump error. This is because the pointer p will point to some garbage address. But this code is absolutely useless.

### using throw statement for undefined catch block

When you use a throw statement, which does not have an matching catch statement , then it will throw an core dump error.

For instance

```
try
{
 throw 20.54;
}
catch(int a)
{
 -----
}
```

In above code, I have catch only for a integer. But I have thrown a double value. So here Core dumb occurs.

Associated error:

terminate called after throwing an instance of ‘d’

Don’t confuse with d here. It is double and i stands for int.

## Using %s for int in pointer:

```
struct
{
int i;
}s;
printf(“%s”,s->i);
```

## Changing the content of char pointer

When you change the content of the char pointer then the segmentation fault will occur.

```
char *p="HELLO";
 p[0]='s';
```

## Unlimited recursive calls:

When the unlimited recursive function calls was made then the stack overflow will occur then the seqmentation fault will occur.

## Dividing 1 by Zero

## THE CODE SECRETS YOU MAY NOT KNOW

### intializing a variable

```
int a=(4,5,6);
```

Here a will be initialized with 6. Because evaluating from right to left.

### INITIALIZING VARIABLE IN PARAMTER

```
int callme(int a=10,int b=5)
```

## C AND CPP DIFFERENCES

1. **INTIALIZING POINTER WITH VALUE WITHOUT TYPECASTING:**

int *p=100; // type conversion error in c++

int *p=100; // valid in c, But gives the warning

2. **Operator overloading and function overloading is supported in c++ and not in c**
3. **Multiple declaration of global variables is allowed in c**

```
int a;
int a;
int a;
```

But when initialized, then even in c multiple declaration is not allowed.

```
int a;
int a=10;
int a; // Invalid already initialized
```

4. **Using a symbol(function) without declaring**

In c we can use the symbol without declaring when their return type is int. But for other types it need to be declared explicitly. For c++ all types it need to be declared explicitly.

5. **void * is implicitly converted any type of pointer in c but not in c++**

```
int main()
{
    int i;
    void *v=&i;
    int *a=v;
}
```

## COMPARISON BETWEEN C++98 AND C++11

### STRUCT TM :

tm_min range is modified

In 98 it is 0-61

in 11 it it 0-60

## GOOD PROGRAMMING PRACTICE

1. use const keyword if the value should not be changed
2. use dynamic memory allocation where possible
3. initialize the freed pointer to null dangling pointer
4. always initialize or memset the variable before use
5. always place macro inside the do while (1). if statement issue will come
6. choose carefully the scope of variable
7. use some naming convention for variable
8. Validate the parameter before you use for operation

## WEIRD FACTS AND INTERVIEW QUESTIONS

### 1. Why c++ does not support the unsigned floating point:

Because unsigned float does not have an equivalent machine code to be executed by the CPU. I.e., Most of the hardware implementation does not support this feature

No standard defines that floating point can be negative.

### 2. Can we declaring variables at any places in the block:

Did you remember in school days that variables will be allowed to be declare at the beginning of the block only. If so then you have used a c90 standard or earlier. After c99 standard variables were allowed to be declared at anywhere in the block.

### 3. What is dual in “select * from dual” ?

Dual is one dummy table in oracle database. It contain only one field and one column with column name “Dummy” and value in the table is ‘x’.

To know about these just execute “select * from dual;”

What is the need for dual table?

This table is never used to store data. The main purpose of this table is to select the pseudo column or to perform arithemetic operations.

For Eg select SYSDATE from dual;

## 4. Different types of NULL usage and their meaning in program

NULL is an pre-processor macro which may contain the following definition.

```
#define NULL 0
```

'\0' is an ascii character and equivalent ascii value is 0

"\0" is an string which is null terminated.

```
#include<iostream>

int main()
{
   char *null_1=NULL;

   char *null_2='\0';

   char *null_3="\0";

   return 0;
}
```

## 5. Write the program to print the executable name

Use the argv[0] parameter. It will print the executable name.

```
#include<iostream>

int main(int argc, char *argv[])
{
   using namespace std;

   printf("\n ");
   printf("\n  :: %s ", argv[0] );


   return 0;
}
```

## 6. what function will return when function with no return statement and return value is specified for function?

As per the standard result is undefined. On x86 architecture it will return the value of the EAX register.

EAX register will contain the result of the last executed instruction. Answer depends upon the platform which you were using.

But mostly it returns the result of the last executed instruction.

```
#include<iostream>

int express()
{
   int a=5+5;
}

int donothing()
{

}

int declarevariable()
{
   int a=9;
}


int main()
{
   using namespace std;

   cout << '\n' << express();
   cout << '\n' << donothing();
   cout << '\n' << declarevariable();

   return 0;
}

OUTPUT:
10
UNPREDICTABLE
9
```

7. **whether sizeof of two different pointers will vary in same machine?**

The answers is yes.  But we may studied that the size of pointer is common for all int pointer,char pointer,double poiter.These are data pointers.

Sizeof data pointer will be different from function pointer

int main()

```
{
        int *a;// data pointer
        int (*my_function)(); // Function pointer
        printf("%u is not equal to %u", sizeof a,sizeof my_function);
        printf("Disclaimer: This is purely Architecture dependent");
}
```

Reason behind this info:

System will store the code and data in different memory. Eg : Havard Architecture

## 8. Opaque pointer and opaque data type

### Opaque data type:

When concrete implementation is not exposed to the user then it is called as opaque data type.

**What is concrete implementation?**

Consider you were writing one library and exposing to the user. Let the library be "Calculator". So when user wants to use your library one has to access from their source code.

Consider the following sample code,

```
int main()
{
 CALCULATOR *l_calculator_object; // Opaque Data type

 openCalculator(l_calculator_object); // Library function call

 closeCalculator(l_calculator_object); // Library function call

}
```

Here library haven't exposed the implementation of Calculator class. Implementation of Calculator class is present in the library. Here Calculator is the opaque data type.

At library, they may define the CALCULATOR as follows

```
typedef  calculator CALCULATOR;
```

So what is opaque pointer then?

Pointer to the opaque data type is called as an opaque pointer

Here l_calculator_object is an opaque pointer

```
EG: (Void *) is also one of the opaque pointer.
```

(or)

Pointer variable defined with opaque data type is called opaque pointers.

Consider another example,

```
typedef struct s s_t;
```

Here Implentation details of struct s is hidden from the end users.

The end user can just declare and use the struct for further communitcation to the functionality module

FILE pointer in c is also one of the opaque pointer.

Eg: FILE *fp;

fclose(fp); // Use of opaque pointer at client side

**Objective of the concept**

To hide the implementation details of the data structure.

Advantages:

Changes in the type will not affect the client code (For Eg if we add one more member in the structure it will not affect the client code. so client code no need to be compiled again. Only the library has to be compiled.

## Information hiding

Information hiding is nothing but hiding the implementation details to the end user. Here opaque pointer or opaque data type is one of the information hiding.

Why opaque data type is one of the information hiding technique?

Consider the FILE pointer in c file library. Here we know FILE is an structure or type definition of some structure.

```
FILE *l_file_pointer;
```

Here we don’t know the members which is defined inside that structure, it means they hidden information inside that structure. So it is called as information hiding.

## Transparent data type

Transparent data type is opposite to the opaque data type. Here information about the structure like member of the structure, data type of the member will be open to the programmer.

Consider the following code.

```
struct myStruct
{
 int a;
}

int main()
{
 myStruct l_obj;
}
```

Here myStruct is an transparent data type.

## Declaring local object for the opaque data type

You cannot declare the object for the opaque data type. This is because, when local object is created for some structure, then during compilation compiler will expect for the concrete implementation of the structure. Since concrete implementation for the structure is not available, you cannot create an object for the opaque data type.

## Calculating the size of opaque data type

No, you cannot calculate the size of the opaque data type.

**Why you cannot calculate the size of the opaque data type?**

Since you don’t have the concrete implementation of the structure, you cannot create the object to that structure. So obviously you cannot calculate the size of the opaque data type.

## Advantages of opaque pointer

1. We can hide the implementation of the class to the user of the class
2. Any change the structure of the class will not affect the user of the class because client is using only an opaque pointer
3. Whenever class structure is changed, user of the class no needs to change the library.

## Disadvantages of opaque pointer

Class cannot be accessed directly. Everything needs to be accessed as member function.

## Application of opaque pointer:

Opaque data type is mainly used in the operating system related activity like writing content in file, reading a content of file etc.,

### 9. How to achieve the data encapsulation in c?

By using opaque pointers.

### 10. Why C++ standard not recommending to initialize the variable to zero ?

```
 int a;
 printf("%d",a); // We will get garbage value
```

This is because memory for the auto variables is allocated at runtime. Assigning the variables with zero may incur a runtime cost. Consider the following statement with huge array size

```
 int a[2000];
 initialize_array(a); // calling function and assign "-1"
```

Here the actual programmers intention is to initialize the array a with "-1". If compiler assigns zero in the first line while creating the variable then application will waste the time in assigning the array with zero because actual users intention is to assign "-1".

### 11. Why following program allows to compile when function prototype or implementation of function is not present before calling that function?

In c following code is valid because c does not expect a prototype while compiling, but in c++ it is not valid because c++ expect a prototype while compiling.

```
int main()
{
 printf("\n WILL YOU ALL ME TO CALL");
 callme();
}

int callme()
{

}
```

### 12. Program to print a string without main function:

This is simple program hack. It can be achieved through function macro. Here main function is renamed as begin function. During pre-processing phase, begin is replaced as main function.

```
#include<stdio.h>
 #define decode(s,t,u,m,p,e,d) m##s##u##t
 #define begin decode(a,n,i,m,a,t,e)
 int begin()
 {
       printf("hello");
 }
```

## 13. Multiple declaration of global variable with same variable name

Guess the following output?

```
int a;
int a;

int main()
{
   return 0;
}
```

Solution: It will not cause any compilation errors in c, but not in C++. This is because multiple declaration of global variables without initialization are allowed in c, since C standard doesn’t restrict on this. But if Global variable is initialized then declaration should not repeat again.

```
int a; // Valid
int a=10; // Valid
//int a; - Invalid, variable already initialized

int main()
{
   return 0;
}
```

## 14. How to allocate the dynamic char array without using char pointer in struct?

Solution: By the struct hack technique

```
struct s
{
 char a[0];
};
```

When allocating memory for struct allocate a memory for char array too.

```
struct s *ptr=(s*)malloc(sizeof(s)*10); //char array with 10 bytes
```

1. Reason for using struct hack is memory will be allocated in linear way along with structure memory.

## 15. What is the size of the class when char array is declared with size 0?

```
#include<iostream>

struct structhack
{
   char a[0];
};

struct no_member_struct
{
};

int main()
{
   printf("\n  :: %u ", sizeof( structhack ) );
   printf("\n  :: %d ", sizeof( no_member_struct ) );
   return 0;
}
Output:
:: 0
:: 1
```

## 16. Can we declare a array with empty size is class or struct?

Is below program will compile?

```
#include<iostream>

class test
{
   public:
   int a[];
   char b[];
};

int main()
{
   test t1;
   return 0;
}
```

OUTPUT: No compilation error. Array with empty size is allowable syntax in c and c++.

## 17. Whether Calling a function without  function declaration  in C or C++ is allowed?

Whenever you call a c function or C++ function, compiler will expect the function definition or function prototype as declaration.

Whether C or C++ programming will allow calling the function without function declaration?

**Example program:**

```
int main()
{
    myFunction(); //calling without pre declaration,

   return 0;
}

int myFunction()
{

   return 0;
}
```

Answer is yes in case of C programming(Terms and condition will apply) and answer is no for C++ programming.

**Calling function in C programming function declaration:**

Function with return type as integral value no need to be declared or defined explicitly in the C programming.

So then what is that terms and conditions?

When your return type is  not a integral value, then you have to explicitly define  or  declare the function . If it is not defined then compiler will throw an error like, I can't found "function declaration" or "function definition" for the function you called.

**Calling function in C++ programming function declaration:**

For c++ it will not compile for all return types irrespective of the return type of the function declaration.

## 18. Why array index in C and C++ was starting with zero(0) instead of one(1)?

Because arrays are implemented similar to pointers in c and c++.

Array is not completely a pointer, but arrays  behaves like a pointer which will depend upon the situation where you were using in code.

Now consider one int array example:

```
int a[10];
```

In the above int array example, I have declared int array a with size as 10.

Here compiler will allocate memory continuously with first element of the array pointing to the first element.

Consider element of the first address as 1000.

Just hold on up to here, let me teach you about subscript operator.

**What is array subscript operator in C or C++?**

Operator which is used to access the array element is called array subscript operator or index operator.

Array index operator [] example:

```
int a[10];
```

How this index operator works in c or c to access array element?

Consider you were accessing third element in array like a[2]

Here how compiler will access the third element is like following

Compiler will change the a[2] in to  (base address of a + 2).

Hope you noticed. So, whatever number you were giving accessing array element, compiler will add it to array base address and it will access the element.

So how to access the 1st element in the array.

You have to add 0 to the base address. The base address of the array contains the 1st value of the array .i.e., arr[0]. Compiler uses the a+0.  To This is the reason array index is starting with zero instead of one.

**Another technical reason for array index starting with zero:**

Not only for offset purpose the array index was starting with zero.

There is also another reason which is considered by the compiler designers and computer scientists to start the index with 0.

We need to consider how much memory array index value takes to store in memory.

To represent the b power n values we require an n bits to store array index in memory if index started with 1.

Seems to be confusing, Look at the example below

For Eg, Consider you want to represent the 8 array elements in memory.

If you start with 0 (0-7) we will required for 3 bits to represent in terms of binary.

If you start with 1(1-8) we will be required 4 bits. Because 8 cannot be represented using the 3 bits it requires an additional 4th bit.

This is the reason why addressing the memory in computer system also starts with zero instead of 1.

### 19. Whether any performanace impacts will occur when we declare variable inside and outside of for loop in c or c++ programming?

I think your answer will be yes.

But my answer is no(Terms and conditions apply).

There will not be any performance impacts when declare the variable inside and outside the loop. But this depends upon your type is POD or non-POD Type.

#### Why there is no performance impacts for POD Types:

Compiler will do some magic in the name optimization for POD-types alone.

When you declare a variable inside the loop, compiler will optimize in such a way that it will not allocate memory for that variable for each iteration rather it will keep the variable as register. So no memory allocation is required at each time of iteration.

**Why terms and conditions apply?**

Remember this purely depends on your compiler. Standard is not giving guarantee for this.

If case your compiler is showing the performance impacts, then better you should throw your compiler.

Consider this for loop example:

```
for (int i=0;i<10;i++)
{
int a=10; // There is no performance impact because compiler will optimize it
MyClass obj; // There is performance impact

}
```

### What about non-POD types like class?

Yes there will be performance impact for the non-POD types this is because constructor need to be called at every iteration.

## 20. Whether 0(zero) is decimal literal or octal literal in C or C++ programming?

You may studied literal starting with 0 is octal. For Eg 07 will be treated as octal number, on other hand 7 will treated as integer by compiler.

So, now question is whether 0 will be treated as integer or octal by compiler?

It will treated as octal by the compiler  not as an integer since rule of thumb is every literal which starts with zero will be considered as octal by the compiler.

## 21. What is difference between user and schema in oracle database?

As you think, both user and schema in oracle database was same thing but there are few differences between them.

User is just a key to access the database resources, where as schema in oracle is collection of objects which contains the information about access to the resources.

Whenever you create user, schema is created by default with same name in oracle.

## 22. What is difference between const char *ptr and char const *ptr in C programming?

They are equivalent in syntax and semantics.

```
const char *ptr is same as char const *ptr;
```

Both form of declaration represents that they are pointer to const char type. I.e., value pointed by the address cannot be changed.

Even in declaration of normal variables const qualifier can be represented before or after the datatype.

For eg:

```
int const a; is same as const int a;
```

## 23. Can we call the non-volatile member function using the volatile object?

Guess the output of the following program.

```
#include<iostream>

class VolatileClass
{
   public:
      int a;
      void callfunc()
      {
         std::cout<<"Called";
      }
};

int main()
{

   VolatileClass volatile a;
   a.callfunc();

   return 0;
}
```

**Answer:**

The above program will not compile because ,

### Simple Answer:

Volatile objects should call only volatile member functions.

### Complex Answer:

If object is declared as volatile, then all member function which is called with that object should be declared as volatile

```
void callme() volatile
{
  std::cout<<"Called";
}
```

So why it has to be volatile?

Note: Answer is based purely on my point of view.

In, volatile is keyword which instructs the compiler like "Dear genius, don't perform any optimization on that object". Here for all member function this pointer is passed as hidden variable. Since object is declared as volatile, compiler should not perform any optimization on this pointer.

Consider one member function which is declared without volatile

```
void callme()
{
  this->a=1;
       // Perform some other activity.
  this->a=1;
}
```

Since compiler is genius, compile will make two statements “this->a=1;” as one statement because throughout the function value of a is 1. So second statement “this->a=1;” will be removed by the compiler.

Compiler will perform this kind of genius activity when volatile keyword is used in the member function.

## 24. Whether const and volatile keyword is considered in the function overloading?

Guess the following output?

```
#include<iostream>

class FunctionOverloading
{

   public:
      int a;
      void callFunctionOverloading()
      {
         std::cout<<"Called Normal Function";
      }

   void callFunctionOverloading()const
      {
         std::cout<<"Called const Function Overloading";
      }

   void callFunctionOverloading()volatile
      {
         std::cout<<"Called Volatile Function Overloading";
      }
};

int main()
{
   FunctionOverloading b;
   b.callFunctionOverloading();

   return 0;
}
```

O/P:

```
Called Normal Function
```

Explanation:

Not only based on parameters of the function, you can overload the functions based on the const and volatile keyword.

Following functions have different meaning

```
void callFunctionOverloading()
void callFunctionOverloading()const
void callFunctionOverloading()volatile
void callFunctionOverloading() const volatile
```

1st function will be called when this function is called with normal objects. 2nd function will be called when this object is declared with const object and finally 3rd one will be called when the object is declared with volatile keyword.

## 25. When destructor of the local objects will be called when go to is used within the function block?

Guess the output?

```
#include<iostream>

class ArrayNotFound
{
 private:
       int m_id;

 public:
       ArrayNotFound(int id)
       {
             std::cout<<"ArrayNotFound constructor called for ID:
"<<id<<std::endl;
             m_id=id;
       }

       ~ArrayNotFound()
       {
             std::cout<<"ArrayNotFound destructor called for ID:
"<<m_id<<std::endl;
       }
};

int main()
{
 int c=1;
 ArrayNotFound obj(1);

 MyLabel:

 ArrayNotFound obj2(2);
 if(c == 1)
 {
       c=0;
       goto MyLabel;

 }

 return 0;
}
```

Answer:

```
ArrayNotFound constructor called for ID: 1
ArrayNotFound constructor called for ID: 2
ArrayNotFound destructor called for ID: 2
ArrayNotFound constructor called for ID: 2
ArrayNotFound destructor called for ID: 2
ArrayNotFound destructor called for ID: 1
```

Explantion:

Whenever label is used in the program, and if control of the program is returned due to goto statements then destructor will be called for the objects declared under the label.

Since obj2 is declared under the MyLabel destructor is called for that object. One should notice that destructor is not called for obj because it is above the label statement. But destructor for obj will be called once at the end of the main function block

Consider the following statement,

```
MyLabel:
 ArrayNotFound obj2(2);


 if(c == 1)
 {
       c=0;
       goto MyLabel; // Desctructor will be called for obj2
 }
```

### 26. Can we initialize the variable with same variable while declaring?

Guess the output?

```
#include<iostream>
int main()
{
 double *ptr=ptr;
 int a=a,b=10,c=b;
 std::cout<<static_cast<bool>(ptr)<<std::endl;
 return 0;
}
```

OUTPUT:

Program compiles without any compilation error. And the output of the program is undefined.

EXPLANATION:

Standard allows to declare the variable with same name. It is perfectly valid as per standard.

So what happens when we initialize like that?

Nothing will happen, only garbage value will be stored.

And the output of the program is undefined because garbage value can be anything.

## 27. Whether non-type template parameter can be assigned with some value?

Guess the output?

```
#include<iostream>

template<int a>
void PostInArrayNotFound(int b)
{
 a=10;
 std::cout<<a<<std::endl;
 std::cout<<b<<std::endl;

}

int main()
{
 PostInArrayNotFound<2>(25);
 return 0;
}
```

Output:

Compilation error.

Explanation:

Non-type template parameter is an read value not an lvalue. For lvalue you cannot assign anything.

```
a=10; is similar to 2=10; after compilation process.
```

During compilation time, compiler will replace the a as 2.

## 28. What happens when extern "C" is used without block in the program?

Guess the output?

```
#include<iostream>

extern "C" int myGlobalVar;

extern "C" int myGlobalVar2;
int myGlobalVar2;


extern "C" { int myGlobalVar3; }


int main()
{
 myGlobalVar=25;
 myGlobalVar2=50;
 myGlobalVar3=100;
 std::cout<< myGlobalVar << " " << myGlobalVar2 << " " << myGlobalVar3 <<
std::endl;
 return 0;
}
```

Output:

Linker error.

Explanation:

All statement in program, except the following lines is fine in the program.

```
myGlobalVar=25;
std::cout<<  myGlobalVar << " " << myGlobalVar2 << " " << myGlobalVar3 <<
std::endl;
```

So, what is the problem in above lines?

myGlobalVar is not defined rather only it is only declared.

Following statement only declares the variable.

```
extern "C" int myGlobalVar;
```

To declare and define the variable following statement has to be used or declare the variable again like I did for myGlobalVar2.

```
extern "C" { int myGlobalVar3; }
```

### 29. What happens when float argument is passed to overloaded function with int and char as parameters(Type Conversion rules)?

Guess the output?

```
#include<iostream>

void PostInArrayNotFound(int a)
{
 std::cout<<"Int is called"<<std::endl;
}

void PostInArrayNotFound(char a)
{
 std::cout<<"Unsigned int is called"<<std::endl;
}

int main()
{
 PostInArrayNotFound(5.22);

 return 0;
}
```

Output:

It will give an compilation error.

EXPLANATION

There is no function with name “PostInArrayNotFound” and double as an argument.

I think, now compiler is in big tension, to call which “PostInArrayNotFound” function now?

Because float can be implicitly converted to any one of the integral type. Note carefully integral type can be char, unsigned char, int, unsigned int.

So because of this confusion, compiler will throw error on you like, call to the “PostInArrayNotFound” function is ambiguous.

### 30. Whether user defined objects is allowed as non-type template parameter?

Guess the output?

```
#include<iostream>
#include<string>

template<std::string s>
void PostInArrayNotFound()
```

```
{
 std::cout<<s<<std::endl;
}

int main()
{
 std::string s="Hello";

 PostInArrayNotFound<s>();
 return 0;
}
```

Output:

It will give a compilation error, saying constant expression is required for the non-type template parameter.

EXPLANATION:

First of all, non type template parameter is a compile time constant expression.

What is constant expression?

If a result of the expression is known at compile time, then it is called as constant expression.

Consider the following statement,

```
6*9/10;
```

Here value is known at compile time, so it is constant expression. In turn in following statement,

```
myVar *125;
```

we are now aware of myVar value. So it is not compile time constant.

Coming to our program, since object s is not a constant expression, compiler will throw an error.

## 31. Priority of  implicit conversion of  user defined type and system defined type

Guess the output?

```
#include<iostream>
#include<string>
class ArrayNotFound
{
 public:
       bool learnSomethingNew;

       ArrayNotFound(const char *p_char_pointer)
       {
              std::cout<<"ArrayNotFound constructor"<<std::endl;
```

```
        }
};

void PostInArrayNotFound(const void *)
{
 std::cout<<"void pointer parameter"<<std::endl;
}

void PostInArrayNotFound(ArrayNotFound p_obj)
{
 std::cout<<"ArrayNotFound class parameter"<<std::endl;
}

int main()
{
  char *a="HAI";
 PostInArrayNotFound("DD");
 return 0;
}
```

Output:

```
void pointer parameter
```

EXPLANATION:

Here PostInArrayNotFound is an overloaded function with parameter as follows

1. Void pointer
2. ArrayNotFound object(Since parameter of ArrayNotFound constructor is char pointer call to PostInArrayNotFound("DD") will inturn call an ArrayNotFound constructor when "void PostInArrayNotFound(const void *)" is not present in the program)

Here compiler has rights to call any one of the function. But since compiler can't behave as it likes, standard said to follow the following rule.

```
System defined implicit conversion(void pointer) will take the high priority when
compared to the user defined implicit conversion.
```

## 32. When constructor and destructor will be called for various object declaration methods in C++ classes?

Guess the output?

```
#include<iostream>

class ArrayNotFound
```

```
{
 private:
       int id;

 public:
       ArrayNotFound(int a)
       {
             std::cout<<"ArrayNotFound constructor called: "<<a<<std::endl;
             id=a;
       }

       ~ArrayNotFound()
       {
             std::cout<<"ArrayNotFound destructor called: "<<id<<std::endl;
       }
};

int main()
{
 /* C++11 - Assume move constructor is supported */
 ArrayNotFound l_obj = ArrayNotFound(1);
 const ArrayNotFound &l_ref_obj = ArrayNotFound(2);
 ArrayNotFound &&rvalue_ref_obj = ArrayNotFound(3);
 ArrayNotFound(4);
 ArrayNotFound l_obj3 = ArrayNotFound(5);
 ArrayNotFound l_obj2(6);

 return 0;
}
```

Output:

```
ArrayNotFound constructor called: 1
ArrayNotFound constructor called: 2
ArrayNotFound constructor called: 3
ArrayNotFound constructor called: 4
ArrayNotFound destructor called: 4
ArrayNotFound constructor called: 5
ArrayNotFound constructor called: 6
ArrayNotFound destructor called: 6
ArrayNotFound destructor called: 5
ArrayNotFound destructor called: 3
ArrayNotFound destructor called: 2
ArrayNotFound destructor called: 1
```

EXPLANATION:

Usually destructor will be called immediately for temporary objects soon after the construction.

As said above, destructor for following declaration method will be called immediately

```
ArrayNotFound(4);
```

But In the following objects creation method, even though temporary objects is used to create objects, destructor will not be called immediately. This is because of move semantics concept in c++. Temporary objects created will be moved instead of copying and destroying the temporary object.

```
ArrayNotFound l_obj = ArrayNotFound(1);
const ArrayNotFound &l_ref_obj = ArrayNotFound(2);
ArrayNotFound &&rvalue_ref_obj = ArrayNotFound(3);
ArrayNotFound l_obj3 = ArrayNotFound(5);
```

## INCLUDE FILES

1. Some of the paths by the compiler are included by default. For eg /usr/include, /usr/local/include. If this option need to be disabled then pass the flag "-nostdinc"
2. GCC provides some environmental variables to include the path for the header files. If path is specified in the environmental variable then compiler will search for the header file in the specified path
   a. C_INCLUDE
   b. CPLUS_INCLUDE

### #include"" and #include<>

It is purely compiler dependent.  Different compilers have a different meaning for that one. This article is for GNU Compiler collection 3.4.6

### #include"myheader.h"

The above directive statement will look at the Current directory (directory where the code is compiled) first. If file was found in the current directory then it will pick up the file. If not found it will proceed the search towards the standard library. If file is found there it will pickup there.

### #include<stdlib.h>

The above statement will look at the predefined C standard library path. The path for standard library will be defined using the environmental variable.

### Similar confusing definitions:

some programmers will define angle bracket is used for the calling the system headers and double quotes was used for calling the programmer defined headers. Its acceptable truth but that is not a proper definition for that one.

we can also call the standard library file “stdio.h” as #include”stdio.h” instead of #include<stdio.h>. Both directive calls will work fine here. But we cannot call the programmer defined headers such as “myheader.h” using the angle bracket.

| Stament | Path of given header file | Valid/Invalid | Reason |
|---|---|---|---|
| **#include<stdio.h>** | Path defined in environment variable for accessing C standard library | valid | It will pick up the stdio.h file because stdio.h will be present at that path |
| **#include”stdio.h”** | Path defined in environment variable for accessing C standard library | valid | After searching the current directory, It looks for the standard library path. |
| **#include<myheader.h>** | Current directory | Invalid | It will not search at current directory. It will search at Library path |
| **#include<fullpath/to/myheader.h>** | Current directory | Valid | Since full path is given, It will search in that path also. |
| **#include”fullpath/to/myheader.h”** | Current directory | Valid | Since full path is given, It will search in that path |

| | | | |
|---|---|---|---|
| | | | also. |
| **#include"myheader.h"** | Current directory | Valid | It looks at current directory. File will be present there |
| **#include"mydir/myheader.h"** | currentdir/mydir/myheader.h | valid | It will search for directory named mydir in current directory. |
| **#include<mydir/myheader.h>** | currentdir/mydir/myheader.h | invalid | It will search for directory named mydir in standard library path. |

## other noting points:

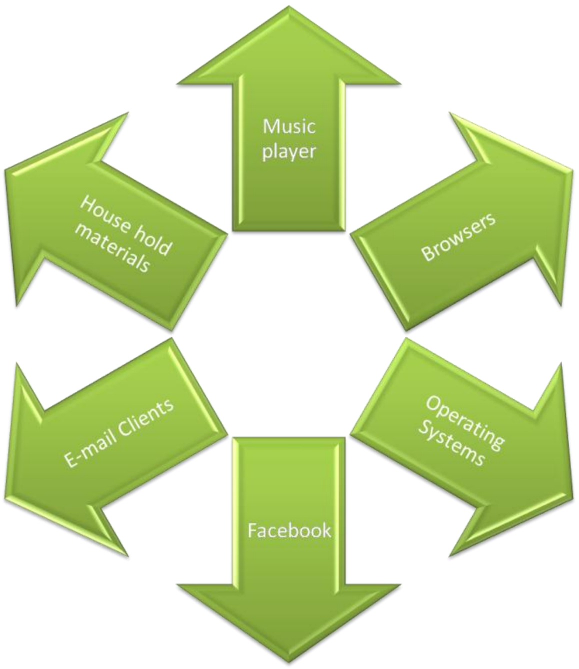
If you include a header file, then compiler will compile the included header file too.

Make sure that you knows the difference between the .c and .h file. The c file will contain the implementations where as the .h file will contain the prototype. This is cultural one. But there is no technical difference between these two.

If you create a file named stdio.h in current directory and if you use a statement #include"stdio.h". Then the compiler will not pick up the stdio.h at the standard library path. Since double quotes is given it will pick up at the current directory path. So if you use a printf statement which is part of stdio.h definion then compiler will throw error that printf is not present there. Because it picked up the stdio.h at current directory it is the dummy file created by the programmer. In that same scenario if you use <stdio.h>, it will work fine because it pickups the stdio.h at standard path.

## CPP APPLICATIONS

Although new programming languages are evolving every day, still C++ plays a vital role in every part of our life. Each and every application you were using is made of C and CPP.



Even though other program are easy to use people depend mostly on cpp because of the following reasons,

Performance

Execution speed

No language compromised the above two parameters with c and C++.

Here are very few areas where c++ is used.

| Area | Application |
|---|---|
| **Email Client** | Mozilla Thunderbird |
| **Operating System** | Most Mac OS, Symbian OS and even windows. |
| **Music** | Winamp media player |
| **Database** | Mysql |
| **Browser** | Mozilla firefox , Chromium browser |
| **Search engine** | Google(few parts) |
| **Kernel** | Unix kernel completely. |
| **Emmbedded softwares** | Washing machine, Microwave oven |
| **Social Network** | Facebook |
| **Application software** | Word, Excel |

## UNDEFINED BEHAVIOR IN PROGRAMMING

Undefined behavior is nothing but we cannot deduct how a application will react to some situation. Code will work fine without any semantic errors. But the output of the code was unpredictable.

C and C++ are most appropriate examples of such behavior.

For instance. If you declare a variable in c or c++, then you can use that variable without initializing it. Code will work fine for that situation. But the output of the code was unpredictable. It vary time to time. This behavior is called undefined behavior

### WHY UNDEFINED BEHAVIOR:

The undefined behavior is allowed mainly to increase the efficiency and performance.

For eg consider the order of evaluation of function parameters. some architecture will be fast when reading from left to right and vice versa.

### DIFFERENCE BETWEEN UNSPECIFIED AND UNDEFINED BEHAVIOUR:

The undefined behavior and unspecified behavior both are different.

unspecified behavior is something where the implementation is undefined but it should conforms to standard. For eg evaluation of function parameters. Here the order of evaluation of function parameter is unspecified. It is up to the implementation.

Undefined behavior is something where we cannot predict the results. Here although it conforms to standards, here the result are unpredictable.

## HOW TO OVERCOME UNDEFINED BEHAVIOUR IN PROGRAMMING:

It is in the hands of the programmer to overcome the undefined behavior in programming.

There are also some tools available like valgrind,gdb  for pointing out these issues. Then always compile with –Wall (if your compiler is gcc) to overcome some of this issues.

## UNDEFINED BEHAVIOUR IN C AND C++:

### USING THE VARIABLE WITHOUT INITIALIZATION:

when you use a variable without initializing then it leads to a undefined behavior. Because it use the garbage value. so its result is unpredictable at all time.

```
int a,b;
b=a+10; // garbage value to b
```

### FUNCTION WITHOUT A  RETURN STATEMENT:

As per semantics, the compiler will allow without return statement even if the return type of the function is other than void say int,float,double etc., But this will result in unpredictable results always.

```
int myfunction()
{
// No return statement
}
int main()
{
int a=myfunction();
}
```

### VARIABLE SIZE OVERFLOW

When the variable size overflows  then it will result in unpredictable results. For eg size of MAX_INT  is 65535. Then the following code will result in overflow. Here the if value of N is greater than the MAX_INT, the value of i never fails in the condition i<N because after variable i reaching the limit, it starts to count from 0 (INT_MIN). So the loop never ends.

```
for(i=0;i<N;i++)
{
//Loop runs infinite times if N is greater than MAX_INT
}
```

## DEREFERENCING A DANGLING POINTER

When you free a memory after malloc. The memory will get freed. But the pointer will point to the same location even after the freeing process. Since the pointer refers to the freed memory. the pointer is called as dangling pointer. When you dereference a dangling pointer you will get the unpredictable results always.

```
char *p=(char *)malloc(sizeof(char)*5);
free(p); // Here memory deallocated but p points to same location. So p is a
dangling pointer
*p=*p+1; //Unpredictable results
```

## DEREFERENCING THE WILD POINTER:

The wild pointer is nothing but the uninitialized pointer. When you use a uninitialized pointer then it will return some garbage value

```
int *p;
*p=*p+1; // undefined behaviour
```

## RETURNING A ADDRESS OF LOCAL VARIABLE

When you return the address of the local variable, then it is also an undefined behavior.

```
int callme()
{
int a;
return &a; // Scope of dies here. No use of passing a’s address
}
```

## CHANGING THE CONST CHAR POINTER

When you attempt to change the const char pointer then the result will be unpredictable.

```
char *p=”HELLO”;
p[1]=’S’; // Lead to segmentation fault
```

in above snippet, the HELLO is string literal stored in read only memory. Here we are attempting to change the read only memory. Code works without any compilation error. But the segmentation fault error will occur.

## COMPILATION AND LINKING

### Compiler for c++

clang

gcc

### Header files

#### Precompiler header files:

Header files are compiled in to intermediate form, so that it is not required to compiled once again when they are included in the source code. This reduces the compilation time

File extenstion

| Extension | Meaning |
|---|---|
| .pch | Pre compiled header file |
| .gch | GCC pre compiled header file |
| .h | C header file |
| .hpp | C++ header file |

## C++11

### New features of C++11

1. Lamda expression
2. Automatic type deduction
3. Rvalue reference
4. Delegating constructor
5. Unordered concepts in STL

## C++14

### New features of C++14

1. Generic function arguments for lamda functions- Using auto for function arugments
2. Return type of function can be specified as auto
3. Constexpr

## HEADER FILES FOR COMMONLY USED FUNCTIONS

| Functions | Header File |
|---|---|
| **Sleep** | Unistd.h |
| **Strcpy** | Cstring |
| **Read,write,close** | Unistd.h |
| **Sstream** | Iomanip |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

## MATH OPERATIONS

1. abs() gives the absolute value.

```
abs(-10) return 10
```

## STANDARDIZATION

### Terminolgy:

#### Defacto standard

Things which is currently in practice and not in law is called as defacto standard.

### ISO STANDARD

#### ISO Standard releases

| VERSION NAME | NICKNAME |
|---|---|
| C++17 | 1Z |
| C++14 | 1Y |
| C++11 | 0X |
| C++98 | |

## RETURN VALUE  OPTIMIZATION (RVO) AND COPY ELLISION

Before learning the Return value optimization. Just  understand the following concepts

### As-If rule:

The process of meeting the requirements or c++ standards, by following any of the implementations by the compiler is called as as-if rule.

### SIDE EFFECTS:

Any operation that makes changes to the area outside the scope is called side effects. For example the return value of the function should return the value to calling function. In addition to that if it changes the value of the global variable then it is said to have the side effects.

## INPUT AND OUTPUT OPERATIONS

1. Istream and ostream was the instantiation of template class basic_istream and basic_ostream placed in the header iostream.
2. Iosbase is base class of all input and output streams in c++. It is non-template clas.
3. Basic_ios is derviced class of iosbase. It is an template class.

### Manipulator function:

#### std::endl

endl stands for end of the line.

It is manipulator function with no arguments of basic_ostream class used to flush the output to the standard output stream.

It is used in conjunction with << operator for std::cout object.

Under ostream, function pointer is implemented to accept the endl function

### Static function

Static function is an function whose scope is restricted to the file. Normally function can be accessed outside the file, where as static function can be accessed only within the file.

Static function cannot be const because there is no meaning in it because there is no hidden this pointer for static function.

Static function cannot be volatile.

## FILE HANDLING IN C++

### Reading file in c++

Reading a file in c and c++ are different. In c we will use the file pointer and in c++ we use the streams.

#### Initializing object for file:

The file object can be intialzed by the following way

```
ofstream fileobj;
ofstream fileobj(“myfile.txt”,ios::in);
```

The first form just creates the object where as the second form creates the object as well as open the file with input operation mode.

#### Opening a file manually:

You can also open the file manually by calling the member function open.

```
fileobj.open(filename);
fileobj.open(filename,mode);
```

C++ supports both the  formats.  If mode is not passed then the default mode is applied

The list of default mode for open member function  is

| class | default mode |
|---|---|
| ofstream | ios::out |
| ifstream | ios::in |
| fstream | ios::in \| ios::out |

More than one mode can be  passed by using the pipe symbol.

```
fileobj.open(filename,ios::in | ios ::app );
```

The list of modes available are

| Mode | Description |
|---|---|
| ios::in | Performs input operation |
| ios::out | Performs output operation |
| ios::app | All output operation performed at the end of the file. This indicates the rest of file is remain untouched, |
| ios::binary | Binary mode |
| ios:trunk | Deletes the content of the old file and replace with the new one. |
| ios:ate | Set the cursor at the end of the file |

You can also check the file is opened or not using the is_open() method.

## Read a file using a overloaded operator <<

The << operator is overloaded to read a file.

```
file_obj >> buffer;
```

This system reads the file line by line. You can use eof() function to check whether the system read the entire file.

Here the buffer can be a char array and also a string class object.

## getline() function

Beaware of getline function here. There are two getline functions. Function which is defined under the string class and another function is defined under the istream class. Both takes the different set of parameter.

getline() under string class

```
istream& getline (istream& is, string& str, char delim);
istream& getline (istream& is, string& str);
```

getline() under the istream class

```
istream& getline (char* s, streamsize n );
istream& getline (char* s, streamsize n, char delim );
```

## Read() method of istream:

The read method reads the n number of character in the stream.

```
istream& read (char* s, streamsize n);
```

Its stores the result in the char array

```
file_obj.read(buffer_char,10);
```

### Manipulation on file position:

use the seekg function to move beween the various positions of file. seekg is the member function of istream.

```
istream& seekg (streampos pos);
istream& seekg (streamoff offset, ios_base::seekdir way);
```

The first function moves the file pointer to the particular file position. whereas the second function goes to the certain position based on the offset set. The second parameter decides from where it should start.  The various value can be set to way is ios_base::beg, ios_base::cur, ios_base::end.

Caution:

seekg doesnot work when the end of file flag is set. At such cased you need to call the clear method to reset the flags.

### clear() method of istream:

The clear reset the eofflag. Calling clear function doesnot make file pointer to move to the beginning of file.

#### File pointer postion:

```
streampos tellg();
```

The tellg returns the file pointer position. Return type is streampos but you type cast it to integer. tellg will work even if eofbit flag is set. tellg is the member function of istream class.

Here the 'g'  in tellg represents the get and 'p' in tellp represents the put.

### Number of characters read gcount:

The gcount returns the number of characters read at last time.

```
streamsize gcount() const;
```

The gcount is the member function of istream class.

### Ignore the content up to some delimiter:

```
istream& ignore (streamsize n=1, int delimiter=EOF);
```

This will ignore the content of file up to the delimiter and the n represent the maximum no of limit to go. By default both the values are set to 1 and EOF.

### Write a content to file:

This will write the content to file poited by ostream object.

```
ostream& write (const char* s, streamsize n);
```

# C PROGRAMMING

## POD AND RELATED CONCEPTS

### INITIALIZATION CONCEPTS:

#### intialization of external variables:

If variable is declared as a extern, then compiler will search for the definition of the in the program.

For eg:

```
extern int ex;
int main()
{
 printf("external initialization=%d",ex);
}
ex=20;
```

the above program will print "external initialization=20" because the compiler will initialize the external variable before the entering the main function.

#### intialization of variables using ocatal number:

```
int x=011;// 011 is the octal number which is equal to decimal number 9;
```

The octal number can also be used for initialization. If number is initialized with zero before then it is octal number. The compiler will convert it to decimal by default.

#### intialization of two  variables at a time:

```
int i,j;
i=j=4,3;
```

The above initialization is valid one. Compiler will not show error. But the both i and j will be initialized with 4. (i=4,j=4).

#### intialization of array using fill :

You can initialize the array using  fill as follows

```
fill(arr+2,arr+sizeof(arr),'0');
```

## Intializing the non-static class members in declaration

Since c++11, initialization of class members is allowed in class declaration.

```
class MyClass
{
 int a=10;
 int a{10};
};
```

But it is not allowed before c++11, because actual memory is not created during declaration. But wondering how it is allowed now? It is by simple hack. Actually this variables are initialized only during the constructor call. Compiler will put an hidden code in the constructor to perform initialization.

## VARIABLE ARGUMENTS IN C

```
#include<iostream>
#include "exploreme.h"


int myFunction(int n,...)
{
   va_list l_va_list;
   va_start(l_va_list,n);

   while(n > 0)
   {
      printf("\n Values are  :: %d ", va_arg(l_va_list,int) );
      n--;
   }

   va_end(l_va_list);

   return 0;
}

int main()
{
   myFunction(2,5,10);

   return 0;
}
```

## SIZEOF AND SIZEOF():

Size of is an unary operator. unary operator is a operator that acts upon the single operand at a time Eg: postincrement, logical not and so on

size of can be used as following formats

```
sizeof(int), sizeof int, sizeof i,sizeof(arr),
```

It is used to calculate the size of the data type as well as the size of the given variable or object.

Size is returned in the bytes which is of the form “size_t”. size_t is an unsigned type which is always positive.

Sizeof as return type

You can also return the sizeof array using the size_t as datatype.

```
size_t  my_function()
{
size_t mysize=sizeof(int);
return mysize;
}
```

size of can be applied to any datatypes such as basic, derived, Enumerated and void

when you want to know about the compiler that how many bytes it takes for the datatype then you can easily find using the sizeof operator

Need for sizeof

Even though the size of the datatypes are predefined by the c standard already Not every implementation of the compiler follows the correct standard. So size of is useful at a time of allocating the dynamic memory. so whenever we need to allocate the memory dynamically we should use the sizeof function to find the size of the datatype.

We cannot use the size of operator in preprocessor expressions. Because the preprocessor does not know about the sizeof

Difference between sizeof and sizeof()

we can use sizeof as both keyword “sizeof int”and function type sizeof(int). Although we use the sizeof as function type it doesnot mean that it is an function. Because according to the C standard function should only take arguments as value not as datatype.

When sizeof is applied to the static array or structure, it will  return the size of  the entire array and entire structure.

Using Sizeof to find the number of elements in a array:

```
sizeof(arr)/sizeof(arr[0]);
```

You can also write a macro to that defines the size of

```
#define arraysize(array) sizeof(array)/sizeof(*array)
```

In one of the above statement I have mentioned that c preprocessor does not know about the sizeof. But here #define just expands the macro at runtime. Here the sizeof will work.

Cannot apply sizeof to structure without members:

Applying sizeof is invalid in the following cases.

```
struct x;
int arr[];
```

Although the above declaration is leagal in C, Applying sizeof is illegal here. Because without knowing the member of the array and structure it cannot find the size.

C++ 11 standard allows to find the sizeof to the structure member too.

For eg:

```
struct s
{
 int a;
 int b;
}
int main()
{
 cout<<sizeof(s::a);
}
```

## POINTERS

### CONFUSING POINTER DEFINITIONS

#### 1.null pointer:

The pointer which is assigned as NULL.

Eg: int *p=NULL; //Null pointer

### 2.void null pointer:

The void pointer which is assigned as null

Eg: void *p=NULL;

### 3const pointer and char constant:

This means that the pointer as well the char is constant

Eg : const * const str=”hello”;

### 4.Dangling pointer

The memory pointed by the pointed variable is freed.

### 5.raw pointer

## EXTENDED DATA TYPE

int, float, char are the fundamental data type. whereas the sizeof the int,float and char will vary to systems to systems and architecture to architecture. To write a implementation  independent code we need to use the datatypes without varying its sizes at different implementation.

To solve this issue standard provides the extended datatypes like int8_t, uint8_t which makes the developers to write the implementation specific code.

# PREPROCESSING DIRECTIVES

## DIRECTIVE

### DIRECTIVE:

1. It is the instruction to the compiler for performing some tasks.
2. It is not a part of language construct or grammar in language. It varies from compiler to compiler
3. Some examples of directive are #define, #line, #include e.t.c.,

### #line directive

1. It modifies the current line number and file name to given line number and file name

2. Application: Compiler will do an preprocessing before compiling a source code. One of the work in preprocessing is replacing the header files with actual code. At the time of preprocessing compiler will change the line number

## MACRO

### STRINGNIFY OPERATION IN MACRO

# will convert the argument of the macro function in to string literal.

For example,

```
#define foo(arg) #arg
std::cout<<foo(helloWorld);
```

During preprocessing foo(helloWorld) will be expanded as “helloWorld” due to the foo macro function.

**Properties:**

1. Macro which is used after # will not expand further.

## is used to merge two literal in macro function. It will not convert to string literal, where as single # will convert to string literal.

For example,

```
#define foo(arg1,arg2) arg1##arg2
std::cout<<foo(hello,ArrayNotFound);
```

During preprocessing foo(hello,ArrayNotFound)  will be expanded as “helloArrayNotFound” due to the foo macro function.

### MACRO PUZZLES

1. **Difference between # and ## in macro expansion?**

```
#include<iostream>
#define expand(a,b) a##b
```

```
#define singles(bs) (#bs)

int main()
{
 int expand(hello,world)=10;
 int object =11;
 std::cout<<helloworld<<":"<<singles(object)<<std::endl;
 return 0;
}
```

OUTPUT:

```
10:object
```

EXPLANATION:

# will get expanded as string literal. Here singles(object) will expand as “object”.

## is used to merge two literals in macro.

Refer MACRO for more information.

2. **Whether stringnification operation will be converted to string literal?**

```
#include<iostream>
#define myFunc(a,b) a##b

int main()
{
 std::cout<<myFunc(hello,vinoth);
 return 0;
}
```

OUTPUT:

Compilation error

```
macro_expansions.cpp:6:20: error: 'hellovinoth' was not declared in this scope
```

EXPLANATION:

## will merge two literal and it will convert to string. Since it is not converted to string literal it is throwing error like 'hellovinoth' was not declared in scope.

Where as, use of single # will be convert to string literals.

3. **Whether macros used after # will expand further?**

```
#include<iostream>

#define outer(a) #a
#define inner(a,b)
```

```
int main()
{
 std::cout<<outer(inner(a,b))<<std::endl;
 return 0;
}
```

OUTPUT:

inner(a,b)

EXPLANATION:

Macro which is used after # will not expand further. Unlike functions, outer macro will expand first and inner macro will expand next.

4. **Whether macros can be defined within local scope?**

```
#include<iostream>

int main()
{
 #define MAX 20

 {
       #define MAX 30
 }

 std::cout<<MAX<<std::endl;

 return 0;
}
```

OUTPUT:

30

EXPLANATION:

#define can be declared in the local scope or global scope. But it is not similar to variable in accessing the macros.

When macro is re-defined with same name, compiler will not throw error, it will just throw a compiler warning.

## STRUCTURE

### TAG NAME IN STRUCT

Tag name in struct is one which is the name of the struct we are giving while creating the struct. For instance, in the following declaration

```
struct s
{
---
}var;
```

s is the tag name. Some will misunderstood s as type name. Here s is not a type name they are actually tag names. They are also like a normal identifiers in c.

The following usage is wrong in c code,

s var1; // Wrong Declaration

struct s var1; //Correct Declaration

Although they are name of the structures we cannot create the variable by just calling them. You need to use the struct keyword while using that identifier.

The process of calling the struct s var1 is called as **elaborated type specifier.**

The enums, union also fall under same category. The name of the enum or union cannot be used separately, they have called with their keywords such as struct, enum;

## WHY TYPEDEF IS GIVEN OFTEN IN STRUCT:

You will wonder sometimes, that why programmers always use a typedef in the struct keyword. This is because, writing

```
struct s
{
--
};
struct s member1;
```

is equal to

```
typedef struct  s
{
--
}s_ref;
s_ref member1;
```

If we fail to use the typedef and in order to use the struct in future, we need to specify struct keyword often. So we are using the typedef in struct programs.

## NORMAL IDENTIFIERS AND TAG NAMES:

The normal identifiers cannot be used like this

```
int flag;
void flag();
```

Compiler throws error at second declaration. But in family that uses the tag names  can have the same name as identifier.

```
struct s {};   //valid
enum s{};     //valid
union s{};     //valid
```

This is because they are referred using the keyword. C stores this identifier in a different table.

And even the following cases also is valid

```
int s;
struct s{};
```

## STRUCT PUZZLES

### 1. Whether object name can be same as class name?

```
#include<stdio.h>

struct Interview
{
 int technical_marks;
 int mcq_marks;
};

int main()
{
 struct Interview Interview;
 Interview.technical_marks=10;

 printf("%d",Interview.technical_marks);

 return 0;
}
```

OUTPUT:

```
10
```

EXPLANATION:

Yes, object can have same name as structure.

## STANDARD LIBRARY FUNCTION

### UTILITY FUNCTION

### PERROR AND STRERR

**perror:**

This function is used to display user message along with the system predefined message.

Upon the call to the perror as follows

```
perror("Error in creation of socket : ");
```

Here the Programmer message is "Error in creation of socket :" In addition to that it also prints the System defined message. Consider the error occurred while creation of socket. Here the Errno value in errno.h will be updated. Then based on the errno value along with predefined Message system message will be printed on the console.

**strerr:**

Here the errno have to be passed manually to get the err description.

Here we cannot pass the user message and it is usually given inside the printf statement.

```
printf("strerr:  %s\n",strerror(errno));
```

**SAMPLE PROGRAM:**

```
#include<iostream>
 #include<errno.h> // if you include this no need to declare the errno variable
 using namespace std;


 /********************************************
  *The below code is just simulation
  *errno variable will be updated with some value automatically by the program
  *when error occurs
  * *****************************************/


 int main()
 {
   //  int errno; This is wronng because it is already defined in the errno.h
    int i=-1,j=0;
    printf("\n Printing up to err no 5");

    for(i=1;i<5;i++)
    {
       errno=i;
       perror("\nMy error description: ");   // No parameter is passed here. It
takes the errno by default
       printf("strerr:  %s\n",strerror(errno)); // Here the errno is passed

    }

    cout<<"\n";
    return 0;

 }
```

OUTPUT:

```
Printing up to err no 5
My error description: : Not owner
strerr:  Not owner

My error description: : No such file or directory
strerr:  No such file or directory

My error description: : No such process
strerr:  No such process

My error description: : Interrupted system call
strerr:  Interrupted system call
```

## TIME

C provides the standard library to calculate the system time. In c you need to include the header file time.h and in c++ you need to include <ctype>

### DATE_TYPE:

```
time_t is the data type name used for storing the epoch time.  The typedef of
time_t is varies depend upon the platform. It is usually a long int or float type.
```

### Note on epoch time:

The epoch time is nothing but the number of seconds since 1 JAN 1970. The 1 Jan 1970 is UTC time. This time is also called unix time or posix time.

## struct tm:

```
struct tm {
               int tm_sec;         /* seconds */
               int tm_min;         /* minutes */
               int tm_hour;        /* hours */
               int tm_mday;        /* day of the month */
               int tm_mon;         /* month start with 0  */
               int tm_year;        /* year since 1900*/
               int tm_wday;        /* day of the week */
               int tm_yday;        /* day in the year */
               int tm_isdst;       /* daylight saving time */
           };
```

## FIND THE TIME DIFFERENCE:

```
double difftime(time_t time1, time_t time2);
```

This returns the difference between the two times in double format.

## time function:

```
time_t time(time_t *epoch_time)
```

This function actually returns the number of seconds till JAN 1 1970. Note that the return value of the time function is also an time_t. If argument is not a null pointer then the value stored in the timer and return value will be same. If time function fails to fetch the calendar time then it returns null.

```
timer=time(NULL);
```

The argument can be passed as NULL. Even If passed as NULL, the return value will be time_t.

## localtime function:

This localtime converts the epoch time to readable format. The local time returns the struct tm.

```
time_struct = localtime(&current_time);
```

## mktime function:

```
time_t mktime (struct tm * time_struct);
```

This does the reverse process of the localtime. It returns the epoch time based on time struct. But make sure that the year should be subtracted from 1900 and month by 1 before converting  to epoch time when tm struct was filled manually.

```
printf("\n mk_time to get epoch time again  :: %d ",    mktime(time_struct));
```

## asctime function:

It converts the time struct to human readable format.

```
char* asctime (const struct tm * time_struct);
```

The snippet is

```
printf("\n asctime :: %s ", asctime(time_struct) );
```

## strftime

This function is used to format the date according to our needs and stores in a buffer

```
 size_t strftime (char* formatted_string, size_t maxsize, const char* format,const
struct tm* time_struct );
```

Choose the list of format specifers from here

http://www.cplusplus.com/reference/ctime/strftime/

some basic format specifiers are

| Format | Meaning |
|---|---|
| %b | Month name in abbreviation (Aug) |
| %d | day of month (01-31) Zero padded for single digit |
| %e | Day of month( 1-31) Space padded for single digit |
| %m | month as decimal number(01-12) |
| %M | minutes (00-59) |
| %y | Year last two digits (92) |
| %Y | Year last four digits (1992) |

| | |
|---|---|
| %H | Hour in 24 Hour format (00-23) |
| %S | Seconds(0-61) |

Simple formatted time is

```
printf("\n  formatted_time :: %s ",  formatted_time   );
```

## gmtime:

This function is used to compute the Greenwich median time

```
struct tm * gmtime (const time_t * timer);
```

## ctime:

This is similar to the asctime

```
char* ctime (const time_t * timer);
```

## Clock function:

This is used to calculate the cpu time

```
clock_t clock ();
```

The return type of the clock function is the clock_t. which is the basic arithmetic type typedefed to clock_t.

The return value is the number of clock ticks that elapsed till the epoch time.

So to calculate the clock time of program execution take the difference of two clock time executed at start and end of the program

It also has the inbuilt macro CLOCKS_PER_SEC to calculate the cpu time in terms of seconds.

## Sample  Program:

```
#include<iostream>
#include<cstdio>
#include<ctime> // In c it is <time.h>

int main()
{
   using namespace std;

   time_t last_connect_time,current_time;
   struct tm *time_struct;
   char formatted_time[25];

   time(&last_connect_time);
   sleep(1);
   current_time=time(0); // This is similar to time(&current_time)
   printf("\n DIFFERENCE TIME IN SECONDS  :: %f ",
difftime(current_time,last_connect_time) );

   printf("\n Epoch time or unix time     :: %ld ",         current_time );

   time_struct = localtime(&current_time);
   printf("\n CURRENT YEAR :: %d ",  (time_struct->tm_year+1900)     );
   printf("\n CURRENT MON  :: %d ",  (time_struct->tm_mon+1)     );
   printf("\n CURRENT DAY  :: %d ",  time_struct->tm_mday     );
   printf("\n CURRENT HOUR :: %d ",  time_struct->tm_hour     );
   printf("\n CURRENT MIN  :: %d ",  time_struct->tm_min     );
   printf("\n CURRENT SEC  :: %d ",  time_struct->tm_sec     );

   printf("\n mk_time to get epoch time again  :: %ld ",    mktime(time_struct) );

   printf("\n asctime       :: %s ",     asctime(time_struct)     );
   strftime(formatted_time,sizeof(formatted_time),"%d-%b-%y",time_struct);
   printf("\n formatted_time:: %s ", formatted_time );


   return 0;
}
```

## Miscellaneous information

1. UTC – Coordinated universal time: It is the time standard.

2. GMT – Greenwich mean time: It is time zone

## Mathematical Related Utility

### strtol

converts the string to long with respect to the base provided.

```
strtol(char *startptr,char *nextstrPtr,int base);
```

## STRING IN C

1. Vsprintf is used to print the string to char pointer based on the format string provided. This is similar to printf concept

# INPUT AND OUTPUT OPERATIONS

## FILE HANDLING

1. Stdin and stdout are pointer variables of  Opaque structure FILE in c

2. Freopen is used to change the stream pointed by the opaque pointer FILE. Using this we can make the stdout to write in file instead of console.

3. If stdout and stdin is closed in one process, then it will not affect the other process.

4. Fopen,fread are C standard functions where as open,close are UNIX Specific functions.

## System calls involved in file

| Function Name | System call |
|---|---|
| fopen | Yes |
| fread | Yes |
| fwrite | Yes |
| fclose | Yes |

## READING THE INPUT FROM STREAM

### gets:

This will read the string from the stdin. Null character will be appended to the end of the string

```
gets(char *str);
```

### puts:

This will write a string in to the stdout stream.

```
puts(char *str);
```

### fgets:

```
char* fgets(char *str,int num,FILE *stream);
```

fgets automatically appends the null character at the end. This is safe when compared to scanf because it asks for the number of character to be read so that it cannot leads to invalid memory access.

For Eg

```
char str[10];
fgets(str,10,stdin);
```

### fputs:

fputs write the string in to the specified buffer.

```
fputs(char *s,FILE *stream);
```

To write in to the stdin buffer

```
fputs(str,stdout);
```

### Scanf :

#### Excluding of characters

We cannot exclude the numbers or floating points. Only characters can be excluded.  To Specify only this characters

Use the [] to exclude the character.

# C AND C++ GENERIC CONCEPTS

## BUILDING THE APPLICATION

### TERMINOLOGY:

#### TRANSLATION UNIT

Source code which is converted in to intermediate form is called translation unit.

##### What is intermediate form?

Intermediate form is nothing but source code is converted in to machine instruction. But it is not completely converted to machine instruction. Remaining job is done by the linker.

This is also called as compilation unit.

#### PREPROCESSING:

Preprocessor does the following process,

1. Replaces the header files with actual content
2. Expands the macro
3. Conditional compilation

Simply we can say it handles all the statements which starts with “#” sign.

## ERROR AND WARNINGS AT RUN TIME AND COMPILE TIME

### 1. BUS ERROR

Wrong address is referred during run time. This occurs due to improper compilation.

# DESIGN

## OBJECT ORIENTED CONCEPTS (OOPS)

### List some oops concepts:

1. Encapsulation
2. Polymorphism
3. Inheritance/Composition
4. Object and classes
5. Message passing/Dynamic dispatch

### Terminology

### object:

Object is a data structure to combine data and related operations. Object is a realization of class.

### Class

class is the data type used to achieve OOP in C++

### Object Oriented Programming (OOP)

Programming using Objects is called Object Oriented Programming

### Difference between object oriented and object based language?

#### Object oriented:

Supports all concepts which is related to objects.

Eg: C++, Java

#### Object Based language:

It supports only some of the object concepts.

## ABSTRACTION

## Interface

### When to use interface?

When underlying information has to be hidden while delivering the class, then we should use interface.

## Abstract class

Abstract class is one of the specific use cases of Interface. Class which contains at least one pure virtual function is called abstract class.

### Need for abstract class

1. Defining a common protocol for set of concrete subclasses. Any further addition of concrete class which belongs to the same common protocol will not impacts the code which we have done for common protocol. For eg: Seller is interface and concrete class can be online seller, book seller. Any further addition of seller will not affect the selling functionality given to clients.

### When to use abstract class?

1. First, we have to think requirement in abstract way. Abstract in the sense, thinking the requirement by separating its associations and attribute. For Example, if requirement is for online sale, then we have to remove that association online and we have to think as sale as abstract one. If there is chance for different type of sale in future, then we have use abstract class there.

2. When any changes to the implementation that should not affect my usage of the code

3. When there need to be default implementation, then we can go for abstract class.

## Difference between abstract class and interface?

| Abstract class | Interface |
|---|---|
| Contains at least one pure virtual function | Contains only pure virtual function |
| It can be definition of any function that can be used by derived class. | It should not have definition of any function that can be used by derived class. |

# TOOLS AND FRAMEWORKS

## DEBUGGING & BUILD MANAGEMENT

### PROFILING

### List of profiling tools

#### Gprof

Profiling tool developed by the GNU.

## GDB DEBUGGER

#### Compilation of program to use debugger:

The program need to be compiled with –g flag to inform the compiler that the program will be used for debugger purpose.

-g informs the compiler to enable the debugging symbols

# THEORITICAL COMPUTER SCIENCE

## TERMINOLOGY

### FORMAL SYSTEM

Formal system is one in which tokens are manipulated according to the pre-defined rules. In simple it is well defined rules.

For Eg, Any programming language, chess

### Web service:

Machine to machine interaction is called web service through a language which can be understood by both machine.

For Eg: XML is one language which can be used as communication medium between two machine

### Interoperable:

Ability of exchanging information between two entities and make use of that information

## PROGRAMMING LANGUAGES

## TERMINOLOGY

### Programming paradigm:

Style in which programming languages are built in is called programming paradigm.

## Strongly typed and weakly typed

It is classified on how strictly types are differentiated.

### Strongly typed language

1. Compiler will throw error when there is type mismatch

### weakly typed language

1. Compiler will perform implicit conversion when there is type mismatch.

## Context switching

Process of storing and restoring something to begin at where we left is called context switching.

## Static and dynamic language

It is classified based on where the type information is available

### Static language

When type information is associated with variable, then it is called as static language.

Eg: C,C++,Java

### Dynamic Language:

When type information is associated with value, then it is called dynamic language.

Eg: Python

## DIFFERENCES

## Difference between script and programming language

During execution of source code, when it is interpreted, then it is called as script and when it is compiled, it is called as programming language.

# SOFTWARE ENGINEERING

## TERMINOLOGY:

### DEPRECATED:

Feature is present in the application, but it will be removed soon in the application in one of the consecutive releases.

### OBSELETE

Not up to date.

## TYPE THEORY

## GRAPH THEORY

### Definition:

Graph theory is nothing but study of graph. Graph is nothing but representation of objects where pair objects are connected by the link.

In terms of mathematics, graph is defined as follows

```
Ordered pair G=(V,E)
Where G is Graph, V is set of Vertices, E is set of Edges
```

### Classification of graph:

a. Based on relationship between nodes

   a. Directed graph(one-way relationship)

      i. Directed graph will traverse only in Specified direction

   b. Undirected graph(Dual-way relationship)

      i. Undirected graph will traverse  in both direction

### Terminologies

| TERMINOLGY | DESCRIPTION |
|---|---|
| Acyclic graph | Graph with no cycles is called as acyclic graph |
| Mixed graph | Graph that contains directed and undirected edges is called as mixed graph |
| Multiedges | When same two vertices is connected by multiple edges then it is called as multi edges |
| Multigraph | It is an undirected graph which contains the multiedges |
| Loop | When edge refer to same vertex, it is called as loop |
| Simple graph | Undirected graph with no loop and multiple edges |
| Quiver (or) Directed Multigraph | As name suggests, it is directed multigraph |
| Weighted graph | Weight or cost is attached to each edge in the graph |

| | |
|---|---|
| Half edges, no ends, loose edge | edges of the graph is connected to only one of the vertex and other end of the vertex is simply left |
| Regular graph | Every vertex has same degree |
| Complete graph | Each pair of vertices is connected by edge |
| Finite graph | When vertex and edge of the graph is finite it is called finite graph |
| Oriented graph | Directed Graph |
| Connected graph | **Connected graph in Undirected graph:**<br>when we take any two vertices in the graph and if there is a connection between two vertices in the graph, then it is called connected graph.<br><br>**Connected graph in directed graph:**<br>There are two types<br>1. Strongly connected graph<br>2. Weakly connected graph<br><br>For any ordered pair in the graph, if there is directed connectivity between them then it is called strongly connected graph.<br><br>For any ordered pair in the graph, if there is no directed path between two vertices, and if connectivity happens when directed path in the graph is replaced with undirected path, then it is called weakly connected graph. |
| Bipartite graph | "Bi"<br>Entire graph is divided in two groups<br>"Partite"<br>Partitioning<br>In bipartite graph, no two vertices in a same group will be connected by a same edge. |
| Complete Bipartite graph | When every vertex of one group is adjacent to every vertex of other group it is called as complete bipartite graph. |
| Di graph | It is also called as directed graph |

## Circuit,walk,path,cycle,trail

In the following table, for closed sequence, only one node will repeat. If only one node is repeated then edges in the following table will not be represented as Repeat.

| Terminology | Vertices | Edges | CLOSED/OPENED | Mind Map |
|---|---|---|---|---|
| Path | Don't repeat | Don't Repeat | open | Consider straight road with no signal from one destination to other. |
| walk | Repeat | Don't Repeat | Closed or open | Consider straight road with signal from one |

| | | | | |
|---|---|---|---|---|
| | | | | destination to other. So that they divert to reach the same node again to avoid traffic. |
| Circuit | Repeat | Don't Repeat | Closed | Same as walk but at end we will be reaching the same destination where we started. |
| Trail | Repeat | Don't Repeat | Open | Same as walk but at end we will not be reaching the same destination where we started. |
| Cycle | Don't Repeat | Don't Repeat | Closed | Same as path, but reaching the same destination where we reach. |

## Graph operations

There are two types of graph operations.

1. Unary operation
2. Binary operation

### Unary Operation:

New graph is produced from the operation performed on the single graph is called unary operation

Some common unary operations are

1. Adding an edge
2. Deleting an edge
3. Merging an edge or vertices
4. Edge contraction

   Edge contraction is the process of removing the edge by combining the two vertices in to one.

5. Graph edit distance

   Graph edit distance is calculation of similarity or dissimilarity between two graph. Graph edit distance is used to transform the one graph to other.

## Binary Operation

New graph produced from the operation performed on the two graph is called binary operation

## Graph Minor

Graph which is formed by removing some vertices of the graph is called as graph minor

## Graph Traversal

There are two types of graph traversal,

1. Depth first search

2. Breath first search

## Depth first traversal:

To perform depth first traversal for the graph, pick one arbitrary node, visit it then from the neighbor of that arbitrary node visit one arbitrary node, print it and continue the process. Ensure that visited node is not visited again during this process by maintaining the visited node list.

```
Any node in graph(Say B) -> Any neighbor node(say E) of the B -> Any neighbor
node(say M) of the E
```

### Depth first search for disconnected graph:

Picking up the arbitrary node and visiting all its neighbor will not help to visit all nodes in disconnected graph. For disconnected graph, don’t pick the arbitrary node, on the other hand visit all nodes in the graph and visit their corresponding neighbor.

### Tree Vs Graph

Depth first search for graph is similar to the tree but unlike tree, graph will contain the cycle. So there is a possibility that it may loop in the cycle. To avoid this depth first search for graph will contain the Boolean array of vertices to mark the visited node. This array is checked before visiting the node to ensure that visited node is not visited again.

And another problem will be tree is connected where as graph can be disconnected.

## Breath first search

Breath first traversal is similar to the depth first traversal but the difference is, in breath first traversal on picking some node, we will be keep on print the arbitrary neighbor of the node which we have choosen where as in the breath first search, on picking some node we will be printing all the neighbor of the node we picked first and we will go for the next node.

```
Any node in graph(Say B) -> visit all neighbor nodes(say E,M,N) of the B -> Any
neighbor node(say G,W) of the E
```

## Applications of graph

1. Networking such as phone network,computer network e.tc
2. Social media like face book, twitter and LinkedIn

## NUMBER SYSTEMS

### Generic concepts:

Number of possible numbers than can be represented using number of digits can be calculated using the following formula

```
(Base of number system) ^ (Number of digits)
```

For Eg, consider Decimal number with 3 digits, then maximum number of possible values can be calculated as

```
10 ^ 3 = 1000
```

I.e., 000-999 can be represented

### Binary Number system

List of number of possible values based on number of digits

| Number of digits | Number of possible values |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1024 |

## BANKERS ALOGORITHM - OS

Consider your traditional bank. Where you can take or deposit money.

Consider people in particular branch was only taking money and finally all money will be exhausted.

So when next person comes to withdraw a money then subscriber has to wait until other person has to deposit money.

Bank head office will allocate some max fund for each branch daily and if particular branch crosses that limit others have to wait until someone deposit.

Likewise in operating system, kernel will allocate the resources such as File descriptor, memory, processor based on the request of the process.

Kernel will maintain the maximum number of resources that can be allocated for each process, number of resources allocated and available free resources.

## DATA AND INFORMATION

### Ascii:

- 0-127 ascii code
- 128-255 Extended ascii code

# DATABASE

## ORACLE PRO*C/C++

## ORACLE SQL

### QUERY PUZZLES

#### Second maximum salary in table

```
Select max(salary) from table where salary <
(select (max) salary from table)
```

```
select salary from table
offset 1 rows
fetch next 1 rows only

select salary from
( select salary from table order by desc limit 2)
 where rownum=1
```

## VIEW

View is kind of virtual table which can hold the columns of more than one table.

### Advantages

1. When we want to share the table name to others, we can share in terms of views
2. Granting particular operation on the table.
3. It stores only the statement and actual data is not stored

### Disadvantages

1. When we delete table, view will become invalid.

### Types – Based on Data Storage

1. Materialized View
2. Non-Materialized View

### Materialized View

Materialized view will store the result of the query for better performance.

# ORACLE-PL/SQL

## CONCEPTS

### Advantages of PL/SQL

1. Tightly integrated with SQL
2. Increase the performance by reducing the traffic between application and database
3. In build optimizer

## STRING MANIPULATION

### Find a string in a string

```
instr(inputString, searchString, searchStartPos Default 1, OccurencePosition
Default 1);
```

### Get a substring in string

```
substr(inpustring, startPos, LengthofStringFromStartPos)
```

## CONCEPTS

### Handling NULL RELATED FUNCTIONS

1. Is null
2. Decode
3. Coalesce
4. Nullif

### Difference between sql and PL/SQL

| SQL | PL/SQL |
|---|---|
| It is used to interact with database with limited manipulation functions. | It is used to interact with database with extended manipulation functions. |
| Data oriented | Procedure oriented |
| Used to write Query | Used to write program blocks |
| It cannot embed PL/SQL blocks | It will embed SQL blocks. |

## INDEX

### Types of index

1. B-Tree Index
2. B-Tree Cluster Index
3. Bit map index
4. Function based index
5. Hash Cluster index
6. Reverse key index
7. Global and local index

## TRIGGER

### Types of trigger

1. Row trigger or statement trigger
2. Before and after trigger
3. System and User events trigger

## CURSOR

### Advantages

1. When we use cursor for large dataset, then entire dataset will not be fetched at a time

### Disadvantages

1. It consumes the process memory
2. When group by is used with the SQL statement, then entire dataset will be processed at a time and it will b placed in memory.

### Difference between function and procedure

| | FUNCTION | PROCEDURE |
|---|---|---|
| RETURN VALUE | YES | NO |
| SQL STATEMENT | Can be used | Cannot be used |
| Purpose | Computation | Business logic |

# ORACLE-SQL

## CONCEPTS

### Difference between truncate and delete

1. Delete is DML command and Truncate is DDL command, so truncate will perform auto commit.
2. Delete does not recover space whereas truncate will recover space.
3. Changes cannot be roll backed in truncate whereas in delete it is possible
4. DML triggers are not possible while performing truncate

### COUNT

**What is difference between count(*) and count (1)?**

Both are same. Both of the statement gives the same result.

```
select 1 from dual;
```

The above statement will print 1 one time. Consider the below statement

```
select 1,EmpId from EmployeeInfo;
```

Consider there are 10 records in the table and now  it will print ten'1' s along with employee id. So count(1) gives 10.

### Data Integrity

Avoiding a data from corruption and maintain the consistency of data.

## STANDARD PL/SQL LIBRARY

### FILE READING

```
declare

v_openMode VARCHAR(10):='R';
v_maxFileSize BINARY_INTEGER:=20000;
filePointer UTL_FILE.FILE_TYPE;

v_buffer VARCHAR(1000);

begin
UTL_FILE.FOPEN('/myDirectory/','myFileName',v_openMode,v_maxFileSize);
UTL_FILE.GET_LINE(filePointer,v_buffer,1000);
UTL_FILE.CLOSE(filePointer);
end;
```

# COMMON THERORITICAL CONCEPTS

## TERMINOLOGY

### RAC

RAC is real application cluster. When oracle application runs on multiple machine and if it is connected to common storage then it is called as RAC (Real application cluster).

### Execution plan

The way in which queries are get executed.

1. FULL TABLE SCAN

2. INDEX SCAN BY ROW ID

3. INDEX RANGE SCAN

4. INDEX FAST FULL SCAN

### Difference between Case and decode

1. Case is PLSQL construct where as decode is not
2. Only equality operator can be used in decode where as we can use other operators in case
3. Case in ANSI-SQL Compliant where as decode is not
4. Datatype consistency is checked in case where as it is not checked in decode
5. Null is handled differently in case and decode

## PERFORMANCE

### Ways to improve performance when load increases

1. RAC can be used to segregate the load to various machines, when CPU utilization is more.
2. Adding a index to query

## KEYS IN DATABASE

### Super key

Column or combination of column which uniquely identifies the row in the table. There can be multiple super key.

### Candidate key

Minimum number of column which is required to uniquely identify the record in the table.

### Primary Key

Minimum number of column which is required to uniquely identify the record in the table, but only one candidate key can be primary key.

## DATABASE NORMALIZATION

### 1 st Normal form

1. Each field in the table should contain the atomic values. For example, address should be broken in to city, state, country
2. There should not be any repeating values.

### 2nd Normal form

1. Should satisfy the 1st Normal form
2. Each row should be uniquely identified by the one column. For example adding primary and foreign key

### 3rd Normal form

1. Should satisfy the 1st Normal form
2. There should not be any transitive functional dependency( No non key fields should depend on other non key fields)

## USING PARTITION CLAUSE FOR AGGREGATE FUNCTIONS

Are you looking for alternate of group by clause?

Consider the scenario, We have to fetch the count of birthday of all students in the particular dept along with the college_id field.

If college id field was not there then, we can use the following query

```
select dept_id,count(*) from profile where Bday='08/29/1992' group by Bday;
```

The above query will work fine but, If we want to include College_id field in the result_set the below query will return error. This is because group by is used for aggregate functions. Here For particular dept there may be several college_id's. As like CSE is present in various colleges.

```
select college_ID,Dept_id,count(Bday) from profile where Bday='08/29/1992' group
by Bday;  -- ERROR QUERY --
```

So workaround is , you have to use the partition clause

```
select college_ID,Dept_id,count(Bday) over (partition by dept_id) from profile
where Bday='08/29/1992' group by Bday;  -- CORRECT QUERY --
```

This will return the following output:

```
COLL_ID        DEPT_ID        COUNT

1              10             5

2              12             2
```

This shows that, at college 1 , in dept 10, 5 students have birthday at  01/29/1992

Here the college id  and dept id should be primary.

# UNIX AND UNIX RELATED PROGRAMMING

## MULTITHREADING-UNIX

### PTHREAD LIBRARY

POSIX (Portable operating system interface) threads provide an API to create and manipulate threads. POSIX threads usually called as a pthreads.

POSIX is an family of IEEE standard which provides the lots of API's.

All the implementation of the POSIX thread are under the pthread.h

All functions of POSIX threads begin with a prefix "pthread_".

**Creating a thread:**

**PROTOTYPE:**

```
pthread_create(pthread_t *thread_info, const pthread_attr_t *attribute,
function_name, (void *)myfunction_args)
```

**EXAMPLE:**

pthread_t mythread;

pthread_attr_t thread_attr;

```
pthread_create(&mythread, &thread_attr, my_function, &i)
```

| argument | Description | Need for this argument |
|---|---|---|
| **pthread_t *thread_info** | Unique identifier for the thread. Contains the information about the thread you were creating. It is of Pointer type. Pass the variable of struct pthread_t. | You will have an multiple threads in your program, to identify and manipulate a single thread this argument becomes useful. |
| **const pthread_attr_t *attribute** | Attributes to the thread are set here. Attributes in thread like priority, stack size, scope, detach state etc | Among several threads if you need to set priority to the particular thread, you can configure with this pointer |
| **myfunction** | This method is called after the successful creation of thread. The code that should be executed under this thread can be placed inside that function. If you need to specify the argument. Don't do it here. The next parameter is used for passing a argument for the function "myfunction" | The code that is need to execute under the created thread can be placed inside the "myfunction". |
| **(void *)myfunction_args** | The arguments for the myfunction is given here. It is of void pointer type. The reason why given as void pointer is, there is no possibility that programmer will pass a int or float to the function, so in general it is passed as a void pointer. We | This argument is needed because if programmer intend to pass argument for a myfunction then he can use this argument to do so. If programmer wishes no argument is needed then he can pass NULL. |

can convert to any datatype

further.

## passing paramter as null:

You can configure the parameter of the pthread_create as NULL which indicates that the default values should be taken.

For instance,

```
pthread_create(&mythread, NULL, my_function, NULL)
```

In above statement, mythread will be created by taking the default attributes. Because the the second parameter is given as NULL. If you give the fourth parameter as NULL. Then the void pointer variable will be assigned as NULL.

This not means the creating the myfuntion with no arguments. Don't confuse with that. It will just assign the NULL value to the void pointer.

Then your thinking will be what about configuring the 1st and 3rd parameter as NULL. If you pass them as NULL then your code will become useless, this is because if create thread without thread function, then where the code for that thread created.

## Default attributes to thread if attribute is not defined

1. Prioirty wil be same as the parent priority
2. Stack address will be 1MB
3. Stack address will be set by the system

## synchronization in threads:

pthread_join(myThread,NULL);

## Thread Attributes:

Properties of the thread can be defined using the thread attributes. Thread properties need to be specified while creating the thread. Thread properties cannot be modified after creation of the thread. Thread attribute structure is the opaque datatype, whose information is hidden for the end-user. So thread attributes cannot be assigned directly to the pthread_attr_t structure members. Appropriate built in functions need to be called to change the properties of the thread.

If there is an n number of threads need to be created in the application and if same property need to be applied to all the threads, single attribute objects can be created and passed to all threads while creating the thread.

## What is the scope of the attribute objects?

Once attribute object is created and initialized, scope of the resources accessed is limited to process. It will exist until the process was alive.

Usually following functions will be called one after the another.

1. Thread initialization(Default thread attributes)
2. Helper functions that used to modify the thread attributes
3. Thread creation.

How to return the resource

## Helper functions to set the thread attribute:

### Pthread_attr_init()

Before setting any thread attribute by using the helper function, it should be initialized by calling the pthread_attr_init(). Because pthread_attr_t is the structure and not an class. It don't have an constructor facility to initialize its member variable.

Callling the helper functions to set the attribute without calling this initialization will result in the undefined behavior.

### pthread_attr_setdetachstate

Thread can be either created as detached(separated) or joined state.

If thread property is defined as detached then it means it becomes independent of the parent thread. All the resources allocated for the thread will be returned. This detached threads are not joinable means that we cannot use the join function to wait until the child thread finishes its work.

```
pthread_attr_setdetachstate(&t1_attr,PTHREAD_CREATE_DETACHED)
```

If thread property is defined as joinable then the parent thread will wait until the child thread finishing the work. Parent thread should call the pthread_join function by passing the child thread. If not called then parent thread will not wait for the child thread to finish the work.

```
pthread_attr_setdetachstate(&t1_attr,PTHREAD_CREATE_JOINABLE)
```

### pthread_attr_getdetachstate

we cannot find the detach state of the running thread. Instead we can use the attribute parameter that is used for creating the thread for finding whether thread is attached or detached

```
pthread_attr_getdetachstate(pthread_attr_t *attr, int detachstate)
```

## pthread_join

pthread_join will suspend the execution of the parent thread until the child thread finishes its work. If child thread finished the work then pthread_join will not wait.

Following results are undefined,

1. If pthread_join function is called simultaneously with the same thread id
2. If pthread_join is called with the parent thread id
3. If pthread_join is called with the thread whose property of the detach state is "detachable"

```
pthread_join(pthread_attr_t threadAttr,void ** ValuePtr)
```

## pthread_attr_destroy:

The above helper function is called to  release the resource acquired by the thread during thread allocation.

```
pthread_attr_destroy(&thread_attribute)
```

### Advantages of giving attribute for thread creation:

1. Priority between two threads can be set
2. Properties of the thread can be modified, It helps at many situations.

## THREAD POOL:

Thread pool is the series of threads which is usually placed in the queue for performing some tasks. The input and output to the thread pool might be queue. Thread in thread pool will process the input queue one by one and places the result in result queue.

Threads are not destroyed after the completion of its tasks. After completion of task thread will take a new task or it will goes to sleep state.

The following scenario may occur depend upon the needs,

Thread will terminate or sleep when allocated task in completed

Thread pool is much useful because.

It reduces the overhead of creating and destroying threads.

Thread creation will take much time. So It reduce the wait time for the client.

## MISCELLANEOUS CONCEPT ABOUT THREADS

### DIFFERENCE BETWEEN SYNCHRONIZATION AND SERIALIZATION:

Both are different concepts.

Synchronization is process of allowing only one thread to enter in to the shared area for manipulation to ensure that  multiple threads accessing the shared area at a same time.

Eg: Shared data accessed by all threads in the process.

Serialization is the process where the object is converted to other form using some encoding for future purpose. In future it can be retrieved, and further decoded to original data.

Eg: Convert some structures data in to binary and store in file. Whenever structure is required retrieve from the file and convert it to form of structure.

### SPURIOUS WAKEUP IN THREADS:

There is a possibility that thread can wakeup from sleeping state, even though thread was not instructed to wakeup. This is caused due to some internal hardware problem or during blackout of system. So condition for waking up of thread should be checked  at the place where thread is about to wakeup.

```
if(!condition_satisfied)
        pthread_cond_wait(&l_cond_var,&mutex_c1)
```

#### pthread condition for wait and signal

1. pthread_cond_wait should be called only under the mutex lock. This is because the state of the conditional variable will be modified during conditional wait.

2. Mutex will be released automatically, once pthread_cond_wait is called.

3. Mutex will be locked again, when it receives the signal while waiting.

## DIFFERENCE BETWEEN MUTEX, LOCKS, SEMAPHORE

| | Lock | Mutex | Semaphore |
|---|---|---|---|
| Ownership | It should be released by the same thread that locked. | It should be released by the same thread that locked. | It can be released by the othre threads by calling semaphore give. |
| Synchronization | | Mainly used for exclusive access | Used for sending signals after finishing the job(Synchronization) |
| Number of resource | Allow only one resource at a time | Allow only one resource at a time | Allow many resource at a time |

## DIFFERENCE BETWEEN BINARY SEMAPHORE & MUTEX

| | Mutex | Semaphore |
|---|---|---|
| Ownership | It should be released by the same thread that locked. | It can be released by the other threads by calling semaphore give. |

## MAXIMUM THREAD ALLOWED PER PROCESS

1. OS doesn't impose any limit on maximum number of threads which can be created in the process.

Maximum thread which can be allowed in the system simultaneously can be estimated indirectly using the number of process which can be allowed in the system. Because thread is also an kind of process

## WHAT HAPPENS WHEN PARENT THREAD IS DIED BEFORE CHILD THREAD IS ALIVE?

1. When main thread is terminated, then process will get terminated and ultimately all threads will terminate

2. When threads other than main terminates without calling exit will not affect other threads

3. When pthread exit is called from main thread other thread will continue to exists.

### TASKS

close fd where possible

return false at all failure

invalid fd in select what happens

Backlog variable

# SOCKET PROGRAMMING

## BSD SOCKET

There are several implementations in c for socket programming. But all the implementation will closely relate to a Berkeley socket model.

### sys/socket.h:

It contains basic socket functions and data structures. The following definitions are found under this header file.

socklen_t (int type)

sa_family_t (int type)

sa_family_t (strcut type with members)

### Ports used for sending:

In server for listening, the port should be choosed by the developer.

Server request the free port to operating system for sending the response to client.

But client selects the random available port for sending the data to server.

### Protocol Family:

The protocol family is collection of protocols that are used in the socket programming. All protocol family will prefix with a “PF_”. Protocol family includes,

### File Descriptor:

File descriptor is simply an position in array of pointer(File describtor table).

whenever the socket function is called it returns a integer value. Probably 3 on the 1st socket call, it means that 0,1,2 are reserved.

The two process can communicate via terminal,pipes,socket. Each process has own file descriptor table.

UNIX always deals with file. so in socket programming too file descriptor.

## Compilation:

Include –lsocket at runtime for compiling a socket program.

## Address family and Protocol family:

There is no major difference between the address family and protocol family. In past it is envisioned that there may be large classification between the address and protocol family. so they segregated it.

## FUNCTIONS AND THEIR USAGE

### Socket();

The socket function returns the certain file descriptor. That is going to be used in the program for further operation. Upon that, it allocates the system resources for passed parameters. Actually it takes the three parameters,

```
int socket(int Address_family,int connection_Type,int
specific_protocol_to_be_used);
```

Parameter1 : Address family:

Address family is the collection of address types that are used in the socket programming. All Address family will prefix with a “AF_”. Address family includes a AF_INET(IPv4),AF_INET6(IPv6),AF_UNIX etc.,

Paramter 2 : Connection type:

The connection type can be of many types, commonly used are

Connection oriented (SOCK_STREAM)

Connection less (SOCK_DGRAM)

Parameter 3 : Specific Protocol to be used:

The first parameter specifies which address family to be used. In Third parameter you have to pass the specific protocol that you are going to use. It may be TCP,UDP,SMTP etc., Macro are defined as IPPROTO_TCP, IPPROTO_UDP.

Return Type :

Here the return type is the file descriptor. Based on the file descriptor all the further socket communication will be made. For eg

```
int FD=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
```

The values 0,1,2 in file descriptor is pre-defined for the stdin,stdout,stderr etc., The number 3 will be assingned if you call socket function for the 1st time in your program. All successful socket function call will return the incremented file descriptor. For eg 4 after 3.

## bind();

The bind function is used for the binding the IPAddress and port number to the socket resources allocated in the socket function. Here binding refer to assigning.

The prototype of the bind function is

```
int bind(int socket_FD, const struct sockaddr *address, socklen_t lengthof(struct
sockaddr));
```

| BIND FUNCTION | |
|---|---|
| PARAMTER | DESCRIBTION |
| int socket_FD | Here the file descriptor of the socket is passed. It is usually return value by the socket function. The FD will point to some number and system use that number as a index to the file descriptor table. |
| const struct sockaddr *address | It is a struct of type sockaddr_in. In this parameter we will pass the information about IP address, port number, family type. |

The members of sockaddr_in are

This is for IPV4:

```
struct sockaddr_in
{
 short sin_family;   // address family type
 unsigned short sin_port; //port number to
be passed
 struct in_addr sin_addr; // IPAddress to
be passed
 char sin_zero;
};

struct in_addr
{
unsigned long s_addr;
};
```

In program Just don't  confuse with the struct sockaddr. It is used for casting only. Note that struct cannot be typecasted but the struct pointer can be type casted.

sin_family:

Here the address family type is passed, Possible values are AF_INET,AF_INET6 etc.

sin_port:

Port number is passed here. Always use htons() function. It is abbreviation for host to network short. Withoout use of that function can lead to endianness problem. Similarly when you want to retrieve the port number to the host use ntohl() or ntohs().

sin_addr:

```
stSockaddr.sin_addr.s_addr=(INADDR_ANY);
```

This is used to pass the IPAddress. so possibly you don't know about INADDR_ANY constant. when you want to accepts at all interface, then you should pass the INADDR_ANY. This is nothing but constant zero. If you want to restrict only the

| | |
|---|---|
| | connections from localhost then you have to use the following<br>`my_sockaddress.sin_addr.s_addr = inet_addr("127.0.0.1");`<br><br>This is for IPV6:<br>`struct sockaddr_in6 { u_int16_t sin6_family; u_int16_t sin6_port; u_int32_t sin6_flowinfo; struct in6_addr sin6_addr; u_int32_t sin6_scope_id; };`<br>`struct in6_addr { unsigned char s6_addr[16]; };`<br>In addition to this sockaddr_storage also available. |
| socklen_t<br>lengthof(struct sockaddr) | Just pass the length of the second parameter. Use the size of operator to sort it out.<br>`sizeof(stSockaddr)` |
| Return Value | The return value here is always -1 or 0 .  0 for success and -1 for failure. |

Next question is, which IP address I can assign for binding the socket. Can I assign any IP address. Absolutely no. You can assign only your IP address. Then you may ask how my socket program will run on other systems if I assign my IP address? Yes, for this reason you have to use INADDR_ANY constant or you can use any one of the following, Assume that your IPAddress is 192.111.55.22,

The following assignment is possible, because you were binding your own ip address to the socket. But major disadvantage is your cannot be portable. You can run your program on your system because you were binding to the own IP Address. But if you run your program with your neighbors system. It wont work.

```
stSockaddr.sin_addr.s_addr=inet_addr("192.111.55.22");
```

The following will work in all system,

```
stSockaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
stSockaddr.sin_addr.s_addr=htonl(INADDR_LOOPBACK);
stSockaddr.sin_addr.s_addr=
```

You can use INADDR_ANY option too,

```
stSockaddr.sin_addr.s_addr=htonl(INADDR_ANY);
```

Here INADDR_A

## LISTEN FUNCTION:

You need to instruct the program to listen for connections. By calling this function you can instruct to listen it.

| LISTEN FUNCTION | |
|---|---|
| PARAMETER | DESCRIBTION |
| int file_describtor | The file descriptor generated by calling the socket function |
| int back_log | This number decides the number of connections to be allowed at a time. |
| return value | 0 on success and errno is returned based on the error occurs |

## ACCEPT FUNCTION:

The accept function extract the first connection on the queue of pending connections request and establishes the connection between them.

This accept function will create the new socket pair(new file descriptor). The file descriptor properties will be similar to the listening socket.

The old socket which will remain on the same port and it will listen for further connections.

Although the listening socket and newly created socket uses the same port. The connection will be unique because each connection is identified by the five tuples. (serverip,serverport,clientip,clientport,protocol)

| ACCEPT FUNCTION | |
|---|---|
| PARAMETER | DESCRIPTION |
| int socket_fd; | The file describtor of one which is listening for clients |
| sockaddr_in *remote_client; | The client details of the new connection accepted is filled to this structure |
| socklen_t *sizeof_sockaddr_in | pass the size of sockaddr_in structure as address |
| return value | The return value is non negative integer upon successful acceptance. This non negative integer represents the new fd for the connection established. All further communications will be made with that file descriptor.  The errno will be updated when error occurs in the program. |

## HELPER FUNCTIONS:

### recv, recvfrom, recvmsg:

Recv is mostly used for TCP. The fourth parameter represents some flags and it can be set to zero if no specific operation required.

```
recv(int file_Descriptor,void *buffer,size_t length_of_buffer,int flags);
```

Recvfrom is mostly used for UDP Sockets.

```
recvfrom(int file_Descriptor,void *buffer,size_t length_of_buffer,int flags,
struct sockaddr *from,socklen_t fromlen);
```

Recvfrom provides the from address and from length as a additional parameter when compared to recv.

Recv will receive the data if received data is less than the specified length, but recvall will throw error as "Resouce not available " in blocking mode

Recvmsg is also used for connectionless sockets with three parameters. Recv msg uses the msghdr structure

```
struct msghdr {
        caddr_t msg_name;
        u_int msg_namelen;
        struct iovec *msg_iov;
        u_int msg_iovlen;
        caddr_t msg_control;
        u_int msg_controllen;
        int msg_flags;
};
```

### send , sendto, sendmsg

Recv is mostly used for TCP. The fourth parameter represents some flags and it can be set to zero if no specific operation required. Flags can be MSG_DONTWAIT etc.,

```
send(int fd,const void *buffer,size_t buffer_length,int flags)
```

sendto is mostly used for UDP Sockets.

```
sendto(int fd,const void *buffer,size_t buffer_len,int flags,struct sockaddr
*dest_addr,socklen_t addr_len);
```

sendmsg is also used for connectionless sockets with three parameters. send msg uses the msghdr structure.

### Read and write:

```
 ssize_t read(int fd,const void *buf,size_t sizeof_bufer);
 ssize_t write(int fd,const void *buf,size_t sizeof_bufer);
```

### Diff between read/write and send/recv:

All performs the same operation. But the send and recv comes with additional parameter to serve for various purposes. Both are same when flag is set as 0 in the send and recv.

## Miscellaneous Information:

Only socket and accept function returns the new file descriptors allocated.

## SELECT OR POLL:

Select or poll function are used to handle the multiple socket connections.

When there is a need to handle multiple clients, then we cannot wait for client response and make other clients in a waiting process. We have to serve the clients in parallel fashion. Say suppose you have processed the request for client A and response for the client A is sent. In this mean time you cannot wait for client A for another request, we should handle the request of client B until the client A responds.

So what select do is , it will say us which fd was ready for read or write at that moment. Based on that we can process the request so that we don’t want to handle the other fd since they were not responded.

```
int select(int fd_Count,fd_set read_fd,fd_set write_fd,fd_set exceptional_fd,const
struct timeval *timeout);
```

fd_set is the structure which contains information required for the library.

```
typedef struct fd_set
{
  u_int  fd_count;
  SOCKET fd_array[FD_SETSIZE];
} fd_set;
```

| SELECT FUNCTION | |
|---|---|
| PARAMETER | DESCRIPTION |
| fd_count | Max_number of file descriptor allocated. This Count is need to be generated by the programmer at the time connecting to the client.  If fd_count is 10 then it means that there are 10 active connections to the server. If one client is closed then fd_Count becomes 9. So it means only the active connections to the server . |

| | |
|---|---|
| | File descriptor allocated for the listening port should be passed to the fdcount while calling the select function for first time. On successful cal the max file descriptor allocated should be passed. |
| fd_set read_fd | This read_fd structure contains the fd's which are ready to read. Read to read in the sense that client was sending some request and servers has to receive those. |
| fd_set write_fd | This write_fd structure contains the fd's which are ready to send. Ready to send in the sense sending the response to client |
| fd_set expectional_fd | This exception fd will be set when some error occurs. while processing the fd. The standard call this as a error fds. |
| Return Value | It returns the number of file descriptor that is ready . It will return zero when the timeout happens on selecting the any socket. It returns -1 when error in reading the socket. |

## Helper macros that is used for select function:

To add a file descriptor to the fd set use the following macro. This will add the fd in the fd_set. This macro will be used while accepting the new client and store the fd.

```
FD_SET(int fd,fd_set *fd_set);
```

To remove the fd in the fd_set use the following macro. This macro will be used while closing the client connection.

```
FD_CLR(int fd,fd_set *fd_set);
```

After calling the select function, to check whether the fd is set or not use the following macro. Returns true if fd is set else it will return the false

```
FD_ISSET(int fd,fd_set *fd_set);
```

To clear all the entries in the FD_SET use the following macro.  This function is called every time when select is called because the after the select call some fd will be set. To reset those set this macro should be called everytime before calling the select function.

```
FD_ZERO(fd_set *set);
```

## Poll method:

Poll method is similar to the select method. Both poll and select tells which fd are read to read and recv.

But the signature of the poll method is different

```
int poll(struct pollfd *ufds,int number_of_fds,int timeout);
```

The definition of struct pollfd is

```
struct pollfd
{
 int fd;
 short events;.
 short revents;
};
```

Unlike select, for calling poll function every fd generated need to be stored in the struct pollfd structure. It is stored as array of struct for multiple fd.

## get socket options

This option is used to get the details about the socket.

```
int getsockopt(int socket, int level, int option_name,void *restrict option_value,
socklen_t *restrict option_len);
```

| getsockopt FUNCTION | |
|---|---|
| PARAMETER | DESCRIPTION |
| socket | Fd for which socket options need to be retrieved |
| level | This is socket Layer. Socket layer is nothing but the |

| | |
|---|---|
| | abstraction of the Socket levels.<br>This can be set with independent of socket we were handling.<br>Protocols independent options can be set using this level |
| | |
| | |
| | |

## Difference between sockaddr and sockaddr_in

## UTILITY FUNCTIONS:

| UTILITY FUNCTIONS | |
|---|---|
| FUNCTION | DESCRIPTION |
| inet_ntoa() | Network to ascii value conversion (binary to dotted format) |
| ntohs() | network to host byte order short |
| inet_aton() | ascii  to network value conversion (dotted to binary format) |
| inet_pton | printable to network  (dotted to binary). This is same as ntoa. But ntoa is depreciated because it does not support IPV6. |
| inet_ntop | network to printable.(Binary to dotted) ). This is same as ntoa. But aton is depreciated because it does not support IPV6. |

## close();

This was defined under the unistd.h. It takes the file_descriptor as input parameter. Actually unistd contains the miscellaneous definitions of standard symbolic constants.

## MISCELLANEOUS

### Major program difference overview for TCP and UDP

| TCP-SERVER | UDP-SERVER | TCP-CLIENT | UDP-CLIENT |
|---|---|---|---|
| socket – SOCK_STREAM | socket –SOCK_DGRAM | socket – SOCK_STREAM | socket –SOCK_DGRAM |
| Bind | Bind | | |
| Listen | | | |
| Accept | | connect | |
| receive or read | Receivefrom | send or write | sendto |
| send or write | Sendto | receive or read | receivefrom |

### Blocking system calls

| TCP-SERVER | UDP-SERVER | TCP-CLIENT | UDP-CLIENT |
|---|---|---|---|
| Listen | | | |
| Accept | | Connect | |
| receive or read | Receivefrom | send or write | sendto |
| send or write | Sendto | receive or read | receivefrom |

### System calls used in TCP and UDP

| TCP-SERVER | UDP-SERVER | TCP-CLIENT | UDP-CLIENT |
|---|---|---|---|
| socket – SOCK_STREAM | socket –SOCK_DGRAM | socket – SOCK_STREAM | socket –SOCK_DGRAM |
| Bind | Bind | | |
| Listen | | | |
| Accept | | connect | |
| receive or read | Receivefrom | send or write | sendto |
| send or write | Sendto | receive or read | receivefrom |

## Functions which return FD

| TCP-SERVER | UDP-SERVER | TCP-CLIENT | UDP-CLIENT |
|---|---|---|---|
| socket – SOCK_STREAM | socket –SOCK_DGRAM | socket – SOCK_STREAM | socket –SOCK_DGRAM |
| Accept | | | |

## Difference between TCP and UDP

| TCP | UDP |
|---|---|
| Connection oriented protocol | Connection less protocol |
| It will send the packet is same order in which it is sent | It is not guaranteed that packet will be in same order as sent |
| Slower packet delivery when compared to UDP | Faster packet delivery when compared toTCP |

## When to use TCP Over UDP?

1. Reliability: When reliability is required, then TCP need to be used over UDP communication.

2. Speed: When packet delivery need to be faster then UDP need to be preferred.

3. When we want to broadcast any information UDP will be better choice.

## Real time application of TCP and UDP

| TCP | UDP |
|---|---|
| Internet Banking | Voip Call |
| | System Heart Beat |
| | I am alive message |
| | SNMP Network<br>Video Streaming |

## TCP IMPLEMENTATION

### SERVER PROGRAM

```
#include"stdio.h"
#include<sys/socket.h> // socket(),bind(),listen(),accept()
#include <arpa/inet.h> //inet_hton
```

```
#include<errno.h>  //Perror
#include<stdlib.h>
#include<string.h>
#include<unistd.h>  // close()

int main()
{
   int socketFD=-1;  // here assigned to -1 because the function will return 0 if
success
   int bind_status=-1;
   char buffer[100];

   printf("\nGoing to allocate resource for socket! We have passed family
type,connection type,protocol name");

   socketFD=socket(PF_INET,SOCK_STREAM,IPPROTO_TCP);

   if(socketFD!=-1)
   {
      printf("\nSuccessfully allocated resources and created FD with id
%d",socketFD);
   }
   else
   {
      perror("\nError in socket() ");
      exit(EXIT_FAILURE);
   }

   printf("\n\nGoing to allocate the IPaddress and port number to the socket
resources allocated.");

   struct sockaddr_in stSockaddr;
   memset(&stSockaddr,'\0',sizeof(stSockaddr));
   stSockaddr.sin_family=AF_INET;
   stSockaddr.sin_port=htons(1200);
   stSockaddr.sin_addr.s_addr=inet_addr("192.168.151.71");


   //Not to invoke  there

   bind_status=bind(socketFD,(struct sockaddr *) &stSockaddr,sizeof(stSockaddr));

   if(bind_status==-1)
   {
      perror("Error in binding the socket with given socket fd and ipaddress ERR
DESC:\n");
      close(socketFD);
      exit(EXIT_FAILURE);
   }
   else
   {
      printf("\nSuccessfully binded the  IPADDR %s  and PORT %d to the given File
descriptor",inet_ntoa(stSockaddr.sin_addr),ntohs(stSockaddr.sin_port));
   }

   printf("\n\nStill the connection is not open... Going to open the connection at
specified port and going to listen " );

   int listen_status=-1;

   listen_status=listen(socketFD,1);  // vino change here
   if(listen_status == 0)
   {
      printf("\n\nListening for clients (Only 12 connections permitted at a time)
....");
   }
   else
   {
      perror("Error in listening :: ");
   }

   struct sockaddr_in remote_client;
   socklen_t  rem_client_size=sizeof(remote_client);

   while(1)
   {
      int new_client_fd=accept(socketFD,(struct
sockaddr*)&remote_client,&rem_client_size);
      printf("\n\nAccepted the client %s Client Port %d ",
inet_ntoa(remote_client.sin_addr),ntohs(remote_client.sin_port));
      printf("\nNew Fd allocated for the incoming client FD Id:
%d",new_client_fd);

      memset(buffer,'\0',sizeof(buffer));
      sprintf(buffer,"HAI CLIENT.... I AM SERVER[%s] from PORT
%d",inet_ntoa(stSockaddr.sin_addr));
      send(new_client_fd,buffer,strlen(buffer),0);
      memset(buffer,'\0',sizeof(buffer));
      recv(new_client_fd,buffer,sizeof(buffer),0);
      printf("\n%s ",buffer);
   }
   return 0;
}
```

## CLIENT PROGRAM

```
#include"stdio.h"
#include<sys/socket.h>
#include<errno.h>
#include<stdlib.h>
```

```
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include <netdb.h>

int main()
{
   struct sockaddr_in sock_addr;
   int socketFD=-1;
   char buffer[100];
   int bytes_sent=0;

   socketFD=socket(PF_INET,SOCK_STREAM,IPPROTO_TCP);

   if(-1 == socketFD)
   {
      perror("\nError in socket() ");
      exit(EXIT_FAILURE);
   }

   struct sockaddr_in stSockaddr;
   memset(&stSockaddr,'\0',sizeof(stSockaddr));
   stSockaddr.sin_family=AF_INET;
   stSockaddr.sin_port=htons(1200);
   int res=inet_pton(AF_INET,"192.168.151.71", &stSockaddr.sin_addr);

   if(res<0)
   {
      perror("Invalid address family");
   }
   else if(res == 0)
   {
      perror("Invalid IP: ");
   }

   if(-1 == connect(socketFD, (struct sockaddr *)&stSockaddr, sizeof(stSockaddr)))
   {
      perror("\nError in connecting to server Error Description:: ");
      close(socketFD);
      exit(EXIT_FAILURE);
   }

   recv(socketFD,buffer,sizeof(buffer),0);
   printf("\n%s ",buffer);

   sprintf(buffer,"HAI SERVER I AM CLIENT[%s]",inet_ntoa(stSockaddr.sin_addr));
   send(socketFD,buffer,strlen(buffer),0);

  close(socketFD);
```

```
   return 0;
}
```

## UDP PROGRAM

### SERVER PROGRAM

```
#include"stdio.h"
#include<sys/socket.h> // socket(),bind(),listen(),accept()
#include <arpa/inet.h> //inet_hton
#include<errno.h>  //Perror
#include<stdlib.h>
#include<string.h>
#include<unistd.h>  // close()

#define SERVER_IPADDRESS  "127.0.0.1"
#define SERVER_PORT 4557
#define EXCHANGE_BUFFER_SIZE 1024

int udpServerConnectionHandling(int *socketFD,struct sockaddr_in *stSockaddr);
void handleConnection(int client_fd);



/* WARNING : PROGRAM IS NOT TESTED */


int main()
{
 int socketFD=-1;
 struct sockaddr_in stSockaddr;
 memset(&stSockaddr,'\0',sizeof(stSockaddr));

 udpServerConnectionHandling(&socketFD,&stSockaddr);
 handleConnection(socketFD);


 return 0;
}




int udpServerConnectionHandling(int *p_socketFD_ptr,struct sockaddr_in
*stSockaddr)
{
 if(NULL == p_socketFD_ptr)
 return 0;

 int   socketFD=-1;  // here assigned to -1 because the function will return 0 if
success
```

```
 int bind_status=-1;


 printf("\nGoing to allocate resource for socket! We have passed family
type,connection type,protocol name");

 socketFD=socket(PF_INET,SOCK_DGRAM,IPPROTO_TCP); //SOCK_STREAM for TCP

 if(socketFD!=-1)
 {
       printf("\nSuccessfully allocated resources and created FD with id
%d",socketFD);
 }
 else
 {
       perror("\nError in socket() ");
       exit(EXIT_FAILURE);
 }


 printf("\n\nGoing to allocate the IPaddress and port number to the socket
resources allocated.");


 memset(stSockaddr,'\0',sizeof(struct sockaddr_in));
 stSockaddr->sin_family=AF_INET;
 stSockaddr->sin_port=htons(SERVER_PORT);
 stSockaddr->sin_addr.s_addr=inet_addr(SERVER_IPADDRESS);


 //Not to invoke  there

 bind_status=bind(socketFD,(struct sockaddr *) stSockaddr,sizeof(struct
sockaddr));

 if(bind_status==-1)
 {
       perror("Error in binding the socket with given socket fd and ipaddress ERR
DESC:\n");
       close(socketFD);
       exit(EXIT_FAILURE);
 }
 else
 {
       printf("\nSuccessfully binded the  IPADDR %s  and PORT %d to the given File
descriptor",inet_ntoa(stSockaddr->sin_addr),ntohs(stSockaddr->sin_port));
 }


 *p_socketFD_ptr=socketFD;
 return 1;
}

void handleConnection(int p_udpSocketFd)
{
 char buffer[EXCHANGE_BUFFER_SIZE];
 memset(buffer,'\0',sizeof(buffer));

 struct sockaddr_storage l_serverStorage;
 socklen_t  sockaddr_storage_size=sizeof(l_serverStorage);



 recvfrom(p_udpSocketFd,buffer,EXCHANGE_BUFFER_SIZE,0,(struct sockaddr
*)&l_serverStorage, &sockaddr_storage_size);
 printf("\n%s ",buffer);

 strcpy(buffer,"RECEIVED");

 /*Send uppercase message back to client, using serverStorage as the address*/
 sendto(p_udpSocketFd,buffer,EXCHANGE_BUFFER_SIZE,0,(struct sockaddr
*)&l_serverStorage,sockaddr_storage_size);
}
```

**CLIENT PROGRAM**

```
#include"stdio.h"
#include<sys/socket.h> // socket(),bind()
#include <arpa/inet.h> //inet_hton
#include<errno.h>  //Perror
#include<stdlib.h>
#include<string.h>
#include<unistd.h>  // close()
#include <netdb.h>


#define SERVER_IPADDRESS  "127.0.0.1"
#define SERVER_PORT 4557
#define EXCHANGE_BUFFER_SIZE 1024

int udpClientConnectionHandling(int *socketFD,struct sockaddr_in *stSockaddr);
void handleConnection(int client_fd);


/* WARNING : PROGRAM IS NOT TESTED */
```

```
int main()
{
 int socketFD=-1;
 struct sockaddr_in stSockaddr;
 memset(&stSockaddr,'\0',sizeof(stSockaddr));

 udpClientConnectionHandling(&socketFD,&stSockaddr);
 handleConnection(socketFD);



 return 0;
}

int udpClientConnectionHandling(int *p_socketFD_ptr,struct sockaddr_in
*stSockaddr)
{
 if(NULL == p_socketFD_ptr)
 return 0;

 int   socketFD=-1;  // here assigned to -1 because the function will return 0 if
success




 printf("\nGoing to allocate resource for socket! We have passed family
type,connection type,protocol name");

 socketFD=socket(PF_INET,SOCK_DGRAM,IPPROTO_TCP); //SOCK_STREAM for TCP

 if(socketFD!=-1)
 {
       printf("\nSuccessfully allocated resources and created FD with id
%d",socketFD);
 }
 else
 {
       perror("\nError in socket() ");
       exit(EXIT_FAILURE);
 }

 *p_socketFD_ptr=socketFD;

 return 1;
}

void handleConnection(int p_udpSocketFd)
{
 char buffer[EXCHANGE_BUFFER_SIZE];
 memset(buffer,'\0',sizeof(buffer));
 struct sockaddr_in l_serverAddress;
 struct hostent *server;
 socklen_t  sockaddr_storage_size=0;


 /* gethostbyname: get the server's DNS entry */
 server = gethostbyname(SERVER_IPADDRESS);
 if (server == NULL) {
       fprintf(stderr,"ERROR, no such host as %s\n", SERVER_IPADDRESS);
       exit(0);
 }

 /* build the server's Internet address */
 bzero((char *) &l_serverAddress, sizeof(l_serverAddress));
 l_serverAddress.sin_family = AF_INET;
 bcopy((char *)server->h_addr,
 (char *)&l_serverAddress.sin_addr.s_addr, server->h_length);
 l_serverAddress.sin_port = htons(SERVER_PORT);
 sockaddr_storage_size=sizeof(l_serverAddress);

 /*Send uppercase message back to client, using serverStorage as the address*/
 strcpy(buffer,"HAI UDP SERVER");
 sendto(p_udpSocketFd,buffer,EXCHANGE_BUFFER_SIZE,0,(struct sockaddr
*)&l_serverAddress,sockaddr_storage_size);


 recvfrom(p_udpSocketFd,buffer,EXCHANGE_BUFFER_SIZE,0,(struct sockaddr
*)&l_serverAddress, &sockaddr_storage_size);
 printf("\n%s ",buffer);
}
```

## SIGNALS

### Error which occur in signals

| ERROR TYPE | DESCRIPTION | SITUATION |
|---|---|---|
| EINTR | ERROR INTERRUPT | 1. When interrupt is occurred during blocking function call. |

## HARD AND SOFT LIMIT IN UNIX

Normal user in linux or unix is restricted to create file descriptor after a certain value.  Normal user cannot create a fd more than the allowed limit.

This limit was set by the root user to the normal users. To know threshold of fd can be created, just  use the following command

```
ulimit -n
```

There are two types of limit

Hard limit

soft Limit

## Hard Limit

Hard limit was set by the root user. Hard limit  says maximum number of fd that the user was allowed to create.

Normal user can modify the hard limit but normal user can decrease the hard limit but normal user cannot increase it.

To know about the hard limit

```
ulimit -Hn
```

To set the hard limit

```
ulimit -Hn 100
```

This means that hard limit is set to 100.

## Soft Limit

soft limit  says maximum number of fd that the user was allowed to create.

Then what is the difference between hard and soft limit?

If hard and soft limit both were same then limit to user is also same value. But If hard limit is higher and soft limit is low then soft limit is the limit value.

Then why we need a two types of limit?

Hard limit is used to restrict the Normal user to go beyond the allocated limit. Soft limit is used to set the limit according to their need. Say suppose Root user allocates limit as 100. then if you don’t want 100 to be set. Then you can change the soft limit to 50. So your limit is 50 now.

Normal user can  change the value of the soft limit (Not more than the hard limit). So the user has the control to set the limit according to their needs.

Soft limit is one which is equal or less than the hard limit. Soft limit cannot be greater than the hard limit.

To know about the soft  limit

```
ulimit -Sn
```

To set the hard limit

```
ulimit -Sn 50
```

This means that soft limit is set to 50.

The soft limit and hard limit you set during that session  is not permanent. It will be reseted to default value while starting a new session.

## STANDARDS

1. XSI – open system interface
2. Single Unix Specification
3. POSIX
4. Open Group

## SEMAPHORE

1. In general Semaphore means sending a signals to others by some form, whereas in programming semaphore is mechanism of restricting the access to the shared resource by more than n resources at a given point of time.
2. There are multiple implementation available for the semaphore. Few among them are sys/sem.h for System V, semaphore.h for POSIX
3. Classic example of semaphore is producer consumer problem(Bounded buffer problem)

### How semaphore is used for producer consumer problem?

## Snapshot of function calls

| Server | Client |
|---|---|
| semget | semget |
| semop | Semop |

## Implementation of semaphore server

```
#include<sys/sem.h> // System V implementation
#include<cstdio>
#include<unistd.h>
#include<pthread.h>
#include<errno.h>

#define THREAD_COUNT 2
#define SEM_COUNT 1 // No of semaphores need to be created

void semaphore_wait(int semid)
{
  struct sembuf sem_op[1];
  sem_op[0].sem_num=0;
  sem_op[0].sem_flg=SEM_UNDO;
  sem_op[0].sem_op=-1;

  semop(semid,sem_op,1);
}

void semaphore_signal(int semid)
{
  struct sembuf sem_op;
  sem_op.sem_num=0;
  sem_op.sem_flg=SEM_UNDO;
  sem_op.sem_op=1;

  semop(semid,&sem_op,1);

}



void init_semaphore(int semid)
{
  struct sembuf sem_op;
  sem_op.sem_num=0;
  sem_op.sem_flg=SEM_UNDO;
  sem_op.sem_op=1; // No of Resource can access at a time a single semaphore
  semop(semid,&sem_op,1);

}

void *thrfunction(void *param)
{
  int semid=*((int *)param);
  printf("\nThread %u created! waiting to aquire critical
region...",(int)pthread_self());
  sleep(1);

  semaphore_wait(semid);
  printf("\nThread %u in critical section ",(int)pthread_self());
  printf("\n Free Resources at semaphore 0 after calling wait :
%d",semctl(semid,0,GETVAL));
  sleep(5);
  semaphore_signal(semid);

  printf("\nThread %u out critical section ",(int)pthread_self());

}

int main()
{
  int sem_key=ftok("key.txt",'A');

  if(0 > sem_key)
  {
    perror("Error in getting key! (or) \"key.txt\" File not found. Cause: ");
    return errno;
  }

  int semid=semget(sem_key,SEM_COUNT,IPC_CREAT|IPC_EXCL|0600);
  /* IPC_CREAT - used to create a new semaphore
  IPC_EXCL - Fail if semaphore is already created
  */



  if(0 > semid)
  {
    perror("Error in creating semaphore! Cause:");
    return errno;
  }


  init_semaphore(semid);

  printf("\n Free Resources at semaphore 0 : %d",semctl(semid,0,GETVAL));
  /*Creating two threads */

  pthread_t threads[THREAD_COUNT];

  for(int i=0;i<THREAD_COUNT;i++)
  {
    pthread_create(&threads[i],NULL,thrfunction,&semid);
  }

  for(int i=0;i<THREAD_COUNT;i++)
  {
    pthread_join(threads[i],NULL);
  }

  if(semctl(semid,0,IPC_RMID) < 0)
  {
    perror("Could not delete semaphore. Cause:");
  }
  return 0;
}
```

## Implementation of semaphore client

```
#include<sys/sem.h> // System V implementation
#include<cstdio>
#include<unistd.h>
#include<pthread.h>
#include<errno.h>

#define THREAD_COUNT 2
#define SEM_COUNT 1 // No of semaphores need to be created

void semaphore_wait(int semid)
{
  struct sembuf sem_op[1];
  sem_op[0].sem_num=0;
  sem_op[0].sem_flg=SEM_UNDO;
  sem_op[0].sem_op=-1;

  semop(semid,sem_op,1);
}

void semaphore_signal(int semid)
{
  struct sembuf sem_op;
  sem_op.sem_num=0;
  sem_op.sem_flg=SEM_UNDO;
  sem_op.sem_op=1;

  semop(semid,&sem_op,1);

}



void init_semaphore(int semid)
{
  struct sembuf sem_op;
  sem_op.sem_num=0;
  sem_op.sem_flg=SEM_UNDO;
  sem_op.sem_op=1; // No of Resource can access at a time a single semaphore

  semop(semid,&sem_op,1);

}

void *thrfunction(void *param)
{
  int semid=*((int *)param);
  printf("\nThread %u created! waiting to aquire critical
region...",(int)pthread_self());
  sleep(1);

  semaphore_wait(semid);
  printf("\nThread %u in critical section ",(int)pthread_self());
  printf("\n Free Resources at semaphore 0 after calling wait :
%d",semctl(semid,0,GETVAL));
  sleep(5);
  semaphore_signal(semid);

  printf("\nThread %u out critical section ",(int)pthread_self());

}

int main()
{
  int sem_key=ftok("key.txt",'A');

  if(0 > sem_key)
  {
    perror("Error in getting key! (or) \"key.txt\" File not found. Cause: ");
    return errno;
  }

  int semid=semget(sem_key,SEM_COUNT,0); // IPC_CREAT BIT SHOULD NOT BE SET
  if(0 > semid)
  {
    perror("Error in creating semaphore! Cause:");
    return errno;
  }

  printf("\n Free Resources at semaphore 0 : %d",semctl(semid,0,GETVAL));

  /*Creating two threads */

  pthread_t threads[THREAD_COUNT];

  for(int i=0;i<THREAD_COUNT;i++)
  {
    pthread_create(&threads[i],NULL,thrfunction,&semid);
  }

  for(int i=0;i<THREAD_COUNT;i++)
  {
    pthread_join(threads[i],NULL);
  }


  return 0;
}
```

# MISCELLENEOUS

## SYSTEM RESOURCES

### SYSTEM FILES

**.profile:**

It will store the settings which is related to session profile

E.g. default shell after logging in to the system.

# PRO *C PROGRAMMING

## TERMINOLOGY

**OCI**

OCI stands for Oracle call Interface. It is an C language library for communicating with oracle database.

## DYNAMIC SQL STATEMENT IN PRO *C/C++

Dynamic sql statement is one in which the sql statement is defined at run time.

### LIFE CYCLE OF DYNAMIC SQL STATEMENT:

The Dynamic sql statement process will parse the sql statement from the string and then assign the value for the host variable in the string one by one. After assigning the value to the host variable it checks for the stament syntax. If syntax check passes then the sql statement will be executed.

This step repeats for the next set of values.

### COMPOSING THE DYNAMIC SQL STATEMENT:

The dynamic sql statement is first stored as string and they are read on the run time for execution. While building the dynamic sql statement one should exclude the statements like exec sql and terminator(;)

#### Place holder for host variable:

Host variable takes the place holder in composing the dynamic sql statement.

Consider the following two statements,

```
select name from emp where empid=:empid;
select name from emp where empid=:e;
```

In above both the host variable are same and meaningful,  because host variable in the dynamic sql statement are just considered as the placeholders.

## MORE ABOUT VARCHAR AND SQL_CONTEXT

As of now we know  varchar in pro *c/c++ is nothing but the structure which is used to pass the input string.

Varchar is nothing but the structure definition internally with two members len and arr.

When you precompile your Pro *C/C++, you will get like this,

```
typedef struct { unsigned short len; unsigned char arr[1]; } VARCHAR;
typedef struct { unsigned short len; unsigned char arr[1]; } varchar;
```

## BASICS OF PRO C

Pro c is the embedded SQL programming language. It is used in the oracle databases for manipulating the databases. It is based on the ANSI C compliance and also supports the multithread Programming. The extension of Proc file is .pc

First precompile the proc file to produce a respective c/c++ file.

To precompile the pro *C/C++ file you need just type the following command.

```
proc filename.pc
```

While precompiling the proc file the oracle gives some options in form of name value pairs. This options were used to configure according to our needs.

There are several parameters to be configured before you precompile the proc file.

Make sure that $ORACLE_HOME environmental variable is set Correctly.

If not path is correctly set, Set using the following command in your terminal.

$ORACLE_PATH=/…/…/../../Oracle/10.0.0.1/orcl

To verify the path set use the following command

echo $ORACLE_PATH

Export the $ORACLE_HOME to your currently working shell.

Eg export $ORACLE_PATH

LD_LIBRARY_PATH. This is the environmental variable in linux that contains the list of dynamic linking files. Make sure that LD_LIBRARY_PATH is set correctly.

In case the ELF(Executable and linking format) error occurs then you are making some mistakes with the portability (32 to 64 bit portability ,64 bit 32 bit portability)

when that error occurs i.e., while compiling the precompiled code, use m-64 or m-32 option in gcc. It is used for compiling the code as 32 or 64 bit.

## After Precompilation:

After Precompiling the proc file,  then corresponding cpp file will be generated. Then that cpp file need to be compiled like this

```
g++ connect.cpp  -L/path/to/oracle/10.2.0/db_1/lib -lclntsh -m64
```

## CONNECTING WITH DATABASE

The sample code to connect the database…

```
#include<stdio.h>
#include<string.h>
//#include"sqlca.h"

EXEC SQL INCLUDE sqlca;
EXEC SQL BEGIN DECLARE SECTION;
 char *username="vinoth_db";
 char *pwd="vinoth_db123";
 char *str="vino";
EXEC SQL END DECLARE SECTION;

int main()
{
   EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
   EXEC SQL COnneCT :username IDENTIFIED BY :pwd USING :str;
   printf("Success Connecting with database!");
   return 0;

sqlerr:
   printf("Error occured in connecting with database");
}
```

OUTPUT:

```
Success Connecting with database!
```

## DESCRIBTION OF THE PROGRAM:

```
EXEC SQL INCLUDE SQLCA;
#INCLUDE "/PATH/TO/SQLCA"
```

The above file is to be included in the proc file to connect with the databases. The above file will contain the information to connect with the databases. The difference between the first and second line is, the first line is a SQL statement. In that we no need to specify the path of the file SQL will refer it internally. But in the second line we need to specify the path of the file.

```
EXEC SQL BEGIN DECLARE SECTION;
EXEC SQL END DECLARE SECTION;
```

All the variable that are used with the EXEC SQL statements are need to be declared under these two lines. If variables are not declared under this lines then SQL precompile will show error. For eg in above code, if variable username is declared outside this then the precompilation will throw error.

```
EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
```

Whenever the error occurs in the database then program control will be directed to sqlerr label. For instance when  you give the username or password wrongly then sqlerr label will be executed.

```
EXEC SQL COnneCT :username IDENTIFIED BY :pwd USING :str;
```

This statement is used to connect with the database. Here I have given connect as COnneCT. there is no special meaning for that. Since the proc is not case sensitive I have written like that. Using statement defines the string that can be used to connect the oracle DB’s.

you can use the input variable for username, pwd as char pointer or VARCHAR.

## other forms of connect statement:

```
 EXEC SQL CONNECT : “vino_db/vino_db123@vino”
```

This is simplest form of connect statement. You can connect like this too. Just give the connection string to connect phrase.

Here the connection string is combination of username,pwd,oracle string.

## connecting to database:

```
CONNECT :username
IDENTIFIED BY :oldpwd
AT :Dbname
USING :connectionstring
ALTER AUTHORIZATION :newpassword
IN :sysdba/sysoper MODE;
```

alter authorization clause is used to change the password at runtime.

### advanced connection technique:

```
EXEC SQL DECLARE DB_NAME1 DATABASE:
```

The above statement will give a unique name to the database connection. we have to use the above declared stamen while connecting to the database. For instance

```
exec sql connect:"abc/abc123" at DB_NAME1;
```

The above code will connect to the datebase with a name of db_name1;

Don't use semicolon for at clause; The db_name1 is not host or program variable. It is just a identifier used by the oracle. No need to declare too. As a exception we can use host variable for at clause too. If we use a host variable no need to use the "Declare db_name1 database" statement.

```
EXEC SQL AT DB_NAME1 INSERT IN TO …………
```

Execute the sql operation by preceeding with the at clause. For cursors no need to use at clause for open,fetch etc. we need to mention only at the Declare statement of the cursor.

## DATATYPES IN PRO*C/C++

There are two types of datatypes in pro *c/c++. They are

Internal Datatype

External datatype

### Internal Datatype:

#### long:

Long datatype was not used now. They are depreciated. It belongs to the char family. We can use the long datatyoe if we have backward compatability.

It is now replaced by LOB.

## varchar2 & varchar

varchar is reserved by oracle for future use. ANSI says that “NULL” and empty string are two different things. So if you rely on that use Varchar if not use VARCHAR2.

size of the varchar can be represented in two forms.

memory size

Number of char

Always rely on number of char because we cannot be sure that how many bytes the single character will take. It depend upon the area we use.

Varchar is defined as structure by the precompiler.

Varchar contains the two members  such as arr and len. arr member contains the value where as len contains the len of the arr. while fetching the database the two fields will be updated. But while giving as input to database, it is the responsibility of the programmer to fill those members.

Typically, the arr is  filled with strncpy statement.

```
char username=”vino_db”;
Varchar user_DB[10];

user_DB.len=strlen(username);
strncpy((char *)user_DB.arr,username,user_DB.len);
user_DB.arr[user_DB.len+1]=’\0’;
```

Don’t fail to give the declaration of VARCHAR under the declare section. The size to be mentioned while declaring the varchar.

## FLOAT DATATYPE

It is an alias for an real or double precision datatype. Real in precision is 23 or less. Whereas Double is 24 or greater.

## Varibles in Pro *C/C++

There are two types of variables in pro *C/C++.

Host variable

Indicator Variable

Both the Host and Indicator variable will prefix with the semicolon to differentiate with the normal variable. We can also use the struct as host variable by prefixing the struct with the semicolon.

### Host Variable:

The host variable is used for communicating between the oracle Database and proc language. For eg if we want to retrieve the data from the database and use that data in programming we can use the host variable.

```
EXEC SQL select  AGE into :age_HV  :age_IV  from PROFILE where  EMP_ID =
:EMPID_HV;
```

In this statement age_HV is the host variable. After successful execution of select statement, age of the given employee will be copied to age_HV.

### Indicator variable:

The indicator variable is one which states the condition of the  host variable. It is the short integer.

```
EXEC SQL select  AGE into :age_HV  :age_IV  from PROFILE where  EMP_ID =
:EMPID_HV;
```

In this statement age_IV is the indicator variable. If this variable  is assigned as 0 after executing the select statement then it indicates that age is copied to host variable age_HV. If it is -1 , then it indicates that value is not fetched. That's why it is called as indicator variable.

### Host array:

It is used to handle the multiple data at a stretch.

## SQL STATEMENTS IN PROC

In PRO *C/C++, the pre-compiler will replace the SQL statements as C statements by calling the SQLLIB.

### Declare statement in sql:

The declare statement is used to declare the indicator and the host variable.

```
EXEC SQL BEGIN DECLARE SECTION;
/* dECLARATION GOES HERE*/
```

```
EXEC SQL END DECLARE SECTION;
```

### DELIMITERS:

The sql statement uses the single quotes to delimit the values and the double quotes to delimit the Identifiers containing the lower and upper case characters.

## DATABASE CONCEPTS USED IN PRO *C/C++

### COMmIT:

```
EXEC SQL COMMIT WORK RELEASE;
```

COMMIT and COMMIT WORK are one and same. No difference in it. It is made for standards. Release is to release all the resources hold by your program.

We can also comment on commit statement using the COMMENT clause followed by soft coded literals

```
EXEC SQL COMMIT COMMENT 'SAVED EMP SALARY';
```

Comment clause is depreciated in the future release of oracle and transaction based on the name is introduced.

### SET TRANSACTION

```
EXEC SQL SET TRANSACTION READ ONLY
```

The above code makes the transaction to the read only state. In order to exit from the read only state we have to Commit/Rollback the work.

### explicit lock:

```
EXEC SQL LOCK TABLE EMP IN ROW SHARE MODE NOWAIT;
```

### handling char data:

While using char data in the pro C/C++ you must be careful  that unwanted blank spaces will be added.

To control that oracle uses the CHAR_MAP pre-compiler option. Char map pre-compiler option determines how to treat the data.

Possible options are

| Option | Blank Padded | Null terminated |
|---|---|---|
| **varchar** | yes | |
| **string** | | yes |
| **charz** | yes | yes |
| **charf** | yes | |

Where f in charf represents the fixed length.

### inline precompiler option:

```
EXEC ORACLE (option=value);
```

## Naming Files:

The filenames should not contain spaces for proc. For Eg some of the filenames or directory names will contain space in windows.

## Placing inside the program:

For embedding the sql statemet in PRO *C/C++

```
EXEC SQL /* SQL STATEMENT GOES HERE */ ;
```

For embedding the PL/SQL statement.

```
EXec sql execute /* pl sql block goes here */ end-exec
```

## Comments lines:

/**/ Multiline comment

You can also use the ANSI-Style comments such as (--…)

## datatype equivalencing:

Giving the flexibility of using the both oracle datatype and c datatype as same type.

## shared & Private sql areas:

When two users execute the same type of query then the oracle will execute in the shared sql area. But they will have a separate sql statement in the sql private areas. In Pro *C/C++ Cursor will name the sql statement and to some extent it controls the sql statement.

## Cursor

used for handling result of more than one dataset. The set of rows returned is called as an active set. There are two types of cursors

Implicit cursor

Explicit cursor

The Explicit cursor are used to find which row is currently processing.

## transaction:

The changes that are made at two or more than two end points due to some activity. Endpoints may be the bank accounts and the changes may be the amount.

## Line continuation:

If you want to  continue the same sql statement in next line then use a backslash. Maximum line length is 1299.

```
exec sql insert into marks values(‘vino\
th’,100,12,4\
5);
```

### maxliterals:

Constant used to restrict the precompiler for Maximum literal length to be allowed for the string because the C compiler may be restricted to certain length.

### conditional precompilation:

Conditional precompilation statement will include

| STATEMENT | DEFINITION |
|---|---|
| **EXEC ORACLE DEFINE** | DEFINES THE STATEMENT |
| **EXEC ORACLE IFDEF** | IF DEFINITION WAS FOUND |
| **EXEC ORACLE IFNDEF** | IF DEF NOT FOUND |
| **EXEC ORACLE ELSE** | ELSE CONDITION |
| | |

### CUrsor in modular programming

Pro c allows for the modular programming. Where the programs can be built in different areas and we can link them together. Cursor should be defined in  each modules because the precompiler will convert the all cursor declaration to a C statements.

## CURSOR

Cursor is used for handling the queries. Each result set in the cursor is handled by the cursor variable. You must declare and open the cursor before using it.

### declaring the cursor

```
sql_cursor my_cursor;
```

The above declaration should be done in the declaration section.

### open the cursor:

Cursor cannot be opened by the exec sql statement. The cursor can be opened in one of the two forms. They are

wrting procedure to open the cursor

use anonymous block inside the pro *c/c++

The syntax for using anonymous block inside the pro *c/c++

```
EXEC SQL EXECUTE
OPEN  :my_cursor FOR SELECT NAME INTO :emp_name FROM EMP_TABLE;
END-EXEC;
```

## CONTEXT

Context is used to connect two or more databases and it allows to switch between them in run time.

### CREATING CONTEXT:

Context is created using the pseudo column sql_cursor.

```
sql_cursor mycursor;
```

The above declaration should be done inside the declare section of pro c/c++.

### ALLOCATE THE CONTEXT:

The context should be allocated before they were used in the program.

```
EXEC SQL CONTEXT ALLOCATE :mycursor;
```

Note that for cursor allocation the same above statement is used but the clause context is removed.

### USE THE CONTEXT:

We can switch between the context by using the following declaration.

The use statement should come before connecting to the database. Allocate is much similar to declaration.

```
EXEC SQL CONTEXT USE :my_cusor;
```

### DEFAULT IN CONTEXT:

You can swith to the default cursor by the following statement.

```
EXEC SQL CONTEXT USE DEFAULT;
```

The default is the keyword no semicolon comes before that.

```
#include"stdio.h"
Exec sql include sqlca;
int  main()
{
   exec sql begin declare section;
   char *usr1="usr1/usr1pwd@servicename1";
   char *usr2="usr2/usr2pwd@servicename2";
   char *usr3="usr2/usr2pwd@servicename3";
   sql_context usrcontext1;
   sql_context usrcontext2;
   int avg_mark;
   exec sql end declare section;
   exec sql whenever sqlerror go to sqlerr;

   // Connecting to default context

   EXEC SQL CONNECT :usr1;

   exec sql select AVG_MAR into :avg_mark from MARK_LIST where STUD_ID LIKE '121';
   printf("\n Avg Mark at connected database: %d",avg_mark);

   //allocating and connecting to usrcontext1

   EXEC SQL CONTEXT ALLOCATE  :usrcontext1;
   EXEC SQL CONTEXT USE :usrcontext1;
   EXEC SQL CONNECT :usr2;


   exec sql select AVG_MAR into :avg_mark from MARK_LIST where STUD_ID LIKE '121';
   printf("\n Avg Mark at connected database: %d",avg_mark);


   //allocating and connecting to usrcontext2


   EXEC SQL CONTEXT ALLOCATE  :usrcontext2;
   EXEC SQL CONTEXT USE :usrcontext2;
   EXEC SQL CONNECT :usr2;


   exec sql select AVG_MAR into :avg_mark from MARK_LIST where STUD_ID LIKE '121';
   printf("\n Avg Mark at connected database: %d",avg_mark);

   // Now i am connecting to default context database

   EXEC SQL CONTEXT USE DEFAULT;
   exec sql select AVG_MAR into :avg_mark from MARK_LIST where STUD_ID LIKE '121';
```

```
    printf("\n Avg Mark at connected database: %d",avg_mark);

    // Now i am connecting to usrcontext2  database

    EXEC SQL CONTEXT USE :usrcontext2;
    exec sql select AVG_MAR into :avg_mark from MARK_LIST where STUD_ID LIKE '121';
    printf("\n Avg Mark at connected database: %d",avg_mark);

    //By this way you can connect mutiple database and you can swith between them

    return 0;

sqlerr:
    printf("\nError in DB \n");

    return 0;
}
```

**OUTPUT:**

```
Avg Mark at connected database: 85.55
Avg Mark at connected database: 51.22
Avg Mark at connected database: 90.15
Avg Mark at connected database: 85.55
Avg Mark at connected database: 90.15
```

In above output note that 1st and 4th line avg was same. This indicates that are connected to same database schema.

## POINTERS & MEMORY MANAGEMENT

```
const char *ptr; // char is constant
char const *ptr; // char is constant
char * const ptr; //pointer is constant
```

### NEW OPERATOR IN C++

## creating the object without new:

```
myclass obj;
```

This creates the object for the myclass. This is similar to initializing the variable (int a;). Here myclass is the type name and obj is variable. This is static storage duration and uses the stack memory. Here the scope of the variable obj expired at the end of the block.

## creating the pointer to class:

```
myclass *obj;
```

This is the pointer to class. i.e., we have created obj Pointer to myclass. you can initialize the obj in two ways

Method 1:

This is static method. This kind is passing a address of one object to initialize the pointer object.

```
myclass obj1,*obj2;
obj2=&obj1; //static allocation and  calls the constructor with no arguments
```

Method 2:

This is dynamic method. This creates a dynamic memory and need to be removed explicitly otherwise the memory leak will occur.

```
myclass obj1,*obj2;
obj2=new obj1;  //Dynamic allocation and  calls the constructor with no arguments
……….
delete obj2;
```

## invalid use of new operator:

```
myclass obj=new myclass;
```

This is wrong method of creating the object. If you want to use this signature, then the obj should be a pointer to the object.

## difference between giving class name and funtion after new operator:

```
myclass *obj=new myclass;  //Type1
myclass *obj=new myclass();  //Type2
```

Both are same. If we fail to give the paranthesis then the compiler will take it as an constructor with no parameter.

## Using new operator for basic datatype:

```
int *a;
a=new int(5);
```

Here the memory for a is created at run time, and the a is initialized with value of 5 at runtime. a is stored under the heap memory and a is to be deallocated explicitly.

## heap memory and static memory:

Heap memory for the dynamic memory allocation.

Stack memory is for the static memory allocation.

```
#include<iostream>

using namespace std;

class new_demo

{



};

int main()

{

  cout<<"Creating Object without new!!!\n";

  new_demo obj1;



  cout<<"Creating a pointer to class\n";

  new_demo *obj_pointer;



  cout<<"Assigning a pointer to class with obj1\n";

  obj_pointer=&obj1;
```

```
    cout<<"Creating a object with new operator \n";

    obj_pointer=new new_demo;



    cout<<"Wrong: Creating a object with new operator and assigning to a normal object instead of pointer
\n";

    // obj1=new new_demo;  Obj is object here we cannot use new for that

    return 0;

  }
```

Output:

```
Creating Object without new!!!

Creating a pointer to class

Assigning a pointer to class with obj1

Creating a object with new operator

Wrong: Creating a object with new operator and assigning to a normal object instead of pointer
```

## MAKE FILE

Make file is used to compile and link a program with large source code. It was developed by a Fieldman at AT &T Bell Laboratory.

The name of the make file will be usually Makefile or makefile but we can change name of that using a command line option.

## Running a make file

To run a make file simply type the make command on the directory

```
make
```

This command will work if your current directory contains the make file with name "Makefile". If you don't want to keep the name of the make file as "Makefile". use the following command with –f flag

```
make –f mymakefile
```

By this way you can use make file at any directory.

## Command Line in Make:

# used for command line in make file

We can also give the comment at the mid of the file for eg

.suffixes  #Here too comment works

## Targets  and prerequisite:

```
targets : prerequisite
receipe
```

Target is the final executable file. (.o file or .exe file). Targets can also be action to be carried out.

prerequisite is the source which is used to produce the targets.

Receipe is the action that make carry out. It can have more than one command to be carried out. Recipe should be followed by the targets and each receipe should start with the tab character. If tab was missed then receipe will not be recognized.

By general it says if prerequisite was changed then build the targets using the receipe rule

## Rule in make:

Rules tells how to make a file. Sometimes in a large code only few files will be modified. Its just a waste of time to compile the entire project in such cases the rules are highly helpful.

```
hello.o: hello.cpp
    g++ -c hello.cpp
```

The above code checks whether the modified date of hello.cpp was newer than the hello.o. If it is new then it execute the next line “g++ …”

## Dependency lines:

The lines with the “:” is called as dependency lines.

```
hello.o: hello.cpp
```

Here left side are dependencies and right side are source files.

There occurs a case where two or more files will depend upon the single file. For such cases do it like this.

```
hello1.o hello2.o: dependme.cpp
```

## SHELL lines:

The line that immediately follows the dependency line is called shell lines. Shell lines must intend with the tab.

## MACROS:

To expand the macro in make file use a $( ).

The macro is expanded while typing the make command and macro are assigned in the form of name value pairs.

By default if no rule is matched then it will compile using the g++

## Using shell command in make file

```
MY_VAR := $(shell ls)
```

use := for assignement. if normal equal sign is used then only the command is copied  and if :=  is used then result of the command is copied

## Automatic variables:

$@ -  Name of the targets

$< - Name of the prerequisite

$* target name minus the suffix i.e, stem (in foo.cpp, foo is the stem and $* gives the foo)

%-For matching any characters

$(@F) – get the file name from the targes. Replace @ with < for getting file name from the prerequesties

### PHONY Targets:

The target which doesnot contain a file is called as phony target,

```
clean:
 rm myobject.o
```

Here the clean is the phony target.

Make file does not do anything with the phony targes until we explicitly specified.

Note that the following command is not a phony target

```
runme : myobject.o
 g++ myexe -o myobject.o
```

## MISCELLANEOUS

## UNEXPLORED C AREAS

## HEADER FILES

### CONDITIONAL COMPILATION:

Conditional compilation is one which is used to produce the different executables based on the parameters passed.  It useful when we design the program to run on the different  platforms.

Meaning for different executables: for eg when we coding for socket programming, for windows we should use the “winsocket.h” header file and for linux and unix we should use the “sys/socket.h” header file. So the compiler will produce different executables at the different platforms. So we are calling as a different executables.

It always begins with # symbol followed by conditional statement.

| Condition | Description |
|---|---|
| **#Ifdef** | If this macro is defined |

| | |
|---|---|
| **#Ifndef** | If this macro is not defined |
| **#If** | Test if a compile time condition is true |
| **#Else** | The alternative for #if |
| **#Elif** | #else an #if in one statement |
| **#Endif** | End preprocessor conditional |

# UNIX CONCEPTS

## HOW TO CHECK THE PORT WAS LISTENING(UNIX)?

There will be several ports listening on your computer. To check which port was listening use the following syntax

```
netstat -a | grep 8881
```

where a is active tcp and udp listening ports

## LIBRARY FILES IN UNIX

Library are precoded,precompiled files that are ready for use.

Programmers in addition to their own source code they will use the others code by including the file in their source code. It is possible to do like that if only one or two programs have to be referenced. In order to include multiple files it becomes more complex.

So, UNIX allows the property of creating the library objects. There are two kinds of library objects.

Static library (.a extension)

Dynamic  Library (.so extension)

### static library:

Archive library is a static library. These files end with extension .a , then archive library are called along with the program so they become the part of the program execution. The archive library are smaller in size when compared to the shared library.

Static library is linked at compile time

You can create your own library files by using the following,

STEP1: Create your own object file by compiling your program.

```
gcc –c file1.c file2.c
```

The command line option –c is given to produce the object file. The above command produces the two object file file1.o and file2.o

STEP2: ARCHIVE THE FILES TO BE USED FOR LIBRARY OBJECT

```
ar -q libmyarchive.a file1.o file2.o
```

The above command creates the library libmyarchive.a with two object files file1.o and file2.o. The command line option q is used for quick append. It is used to append the member to the archive.

To view the members of the obect use the

```
ar -t libmyarchive.a
```

This will display the members of the libmyarchive such as file1.o and file2.o

Other useful command line options:

-d : deletes the member of archive

-x : extracts the member

-V : modifies the version number of the archive

After the successful creation of libray you can include in a source file by the following

```
gcc newfile.c libmyarchive.a
gcc newfile.c –lmyarchive
```

The second option will work if you include the library under the /opt/usr/lib and remember that the file name you have created should prefix with a lib.

Using ranlib option:

The ranlib option is used for creating the index for the archive file you have been created. The ranlib stands for randomize library.

If you have 100 object file in archive then loader needs to look at all the 1000 files to load the library object. In order to speed up the process ranlib program in GNU creates the index for the archive. Its like the table of content in a book.

```
ranlib libmyarchive.a
```

## Dynamic library:

shared library is a dynamic library. These files end with extension .so, then shared library is not called along with the program, they are called whenever they are needed. The archive library are larger in size when compared to the archive library.

Dynamic library are linked at run time. Need to update the LD_LIBRARY_PATH Environment variable for the path of the Dynamic Library. At run time it will refer that path for loading the dynamic library.

At compiling also we need to include the path to dynamic library using -L Compiler option.

No need to compile the client programs(Programs using the shared library) when shared library code is changed.

STEP1 : Compile the program as PIC:

So are you wondering what is the PIC. PIC is the position independent code. It means it will not depend on the memory where the code was stored. All shared library objects should be a PIC.

```
g++ -c –fpic test1.cpp
```

Yes fpic is the command line option that is used for creating the process independent code and PIE stands for process independent executable.

STEP2: Creating the shared Library using –shared

Now you have to modify the .o file in to .so file. I mean object file in to the shared object file. To do so,

```
g++ -shared –o libtest_shared.so test.o
```

Here libtest_shared.so is created from file test.o

STEP3: Linking and exporting to LD_LIBRARY_PATH(Loader Library path)

Now you have to inform the compiler about the shared library about where he is present to access it. For that unix uses the LD_LIBRARY_PATH environmental variable.

**SET LD_LIBRARY_PATH='/PATH/TO/SHARED/LIBRARY/'**

After informing the compiler you should export the LD_LIBRARY_PATH to your current shell.

Try to learn about rpath too. It is the command line option that is used to give the path of the shared library at run time.

## Difference between static and Dynamic(Shared) Library

If static library code is changed then entire source code need to be compiled again where as in dynamic library source code no need to be compiled if dynamic library is changed,

### library naming convention:

Library are stored under the /usr/lib/. Every lbiary object is prefixed with a lib.

Eg: libm.a (maths libary), libpthread.a (thread library)

They are called while compilation by the following format.

```
g++ file1.cpp -lm -lpthread
```

## UNIX COMMANDS

### NETWORK REALTED COMMANDS

| Command | Description | Options |
|---|---|---|
| ss | Command to get the socket statistics | l – List the process<br>p – Display process Information<br>t – get only tcp sockets<br>n-show numeric hosts instead of name |
| lsof | List of opened files and associated information | -i website:portnumber |
| | | |

### Command to get the process id of given port number

1. Socket statistics command

```
ss -pn
```

2. Netstat command

```
netstat -pn
```

3. lsof

```
lsof -ip :portnumber
```

## GREP COMMAND IN UNIX

### Extracting a piece of text in file:

```
grep pattern filename
grep vinoth  testfile.txt
```

The above  grep pattern will fetch the lines which contains the text like vinoth, vinothkumar, haivinothkumar and so on.

Note that it will not fetch the vino, vinod. To fetch like this text use the regex pattern.

### Extracting a piece of text in directory:

```
grep pattern *
grep vinoth  *
```

Move to the directory where you want to search and type the above command.

This will display the result by prefixing with the file name.

### Extracting a piece of text in same kind files:

```
grep pattern *.filetype
grep vinoth  *.c
```

This will certain text only in c files of the present working directory.

### Extracting a piece of text in multiple specified files:

```
grep pattern filename1 filename2
grep vinoth  test1.c test2.c
```

The pattern vinoth will be searched in files test1.c,test2.c

### Extract all except the given pattern:

```
grep -v pattern filename
grep –v vinoth  testfile.txt

```

This will display all lines except the lines which contains the word vinoth.

### Extract the words which contains the spaces:

```
grep  'int m' test.txt
```

In above grep command the words with spaces are searched. Aware that int will not be searched in the above case. But the ‘int main’ will be searched.

### Stroing the grep pattern to file:

```
grep pattern filename > storefile.txt
grep vinoth  testfile.txt > greped.txt
```

The above pattern will store the greped pattern to grepped.txt

### case-insensitive search:

```
grep -i pattern filename
grep -i vinoth  testfile.txt
```

The above pattern will search for both vinoth  and Vinoth.

### Match the whole words only:

```
grep -w pattern filename
grep -w vinoth  testfile.txt
```

This will search for only vinoth, all words such as vinothkumar will be ignored.

### Count the number of lines that matched with grep:

```
grep -c pattern filename
grep -c vinoth  testfile.txt
```

This will output a number of lines that matched.

### List the files names for which the grep pattern matches

```
grep -l pattern filename
grep -l vinoth  testfile.txt
```

This will list all files that contains the given text.

### Matching with starting word of line:

```
grep  ^pattern filename
grep  ^vinoth  testfile.txt
```

This will print all the lines that start with word vinoth

### Matching with ending word of line:

```
grep  pattern$ filename
grep  vinoth$  testfile.txt
```

This will print all the lines that end with word vinoth

### Extracting the line contains the given word

```
grep  ^pattern$ filename
grep  ^vinoth$  testfile.txt
```

This will print the lines that contains the only the word vinoth.

### search for blank line in grep

```
grep  '^$' filename
grep  '^$'  testfile.txt
```

This will search for blank lines in the given file.

### searching the special character

You cannot usually search for the special character normally. For eg if you want to search for the 'hai$' then $ will take the special meaning here.

To switch off the special meaning use backslash.

```
grep  'pattern\$' filename
grep  'vinoth\$'  testfile.txt
```

This will search for vinoth$ instead of the lines which end with the word vinoth.

## Matching any one of the given character

```
grep  '[matchlist]pattern' filename
grep  '[Vvac]inoth'  testfile.txt
```

This will search for either the vinoth ,Vinoth,cinoth etc.

You can define range like this

```
grep  '[0-9]inoth'  testfile.txt
```

## Ignoring the match of given character set

```
grep  '[^ignorelist]pattern' filename
grep  '[^V]inoth'  testfile.txt
```

This will ignore the Vinoth, but accepts the vinoth.

## Match with prefix or suffix

```
grep  'pattern*' filename
grep  'vinoth*'  testfile.txt
```

This will search for the word that starts with vinoth.

## Grep that excludes some pattern

```
grep  -v 'vinoth*'  testfile.txt
```

This will search for the word without vinoth.

### variants of grep:

There are several variants of grep available in grep such as grep, ggrep, egrep.  The grep commands can be found in /usr/bin/grep or /usr/xpg4/bin/grep

| OPTION | Description |
|---|---|
| **h** | Suppress the file name |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

## SAND BOX:

/usr/xpg4/bin/grep

## UNIX COMMAND LINE MEANING

| Option | Commands | Lower case | Upper case |
|---|---|---|---|
| A | | | |
| B | | | |
| C | | | |
| D | Touch | Change the date in yyyymmdd | |
| E | | | |
| F | | | |
| G | | | |
| H | | | |
| I | List<br>Listening process | Ls<br>ss | |
| J | | | |
| K | | | |

| | | | |
|---|---|---|---|
| L | | | |
| M | | | |
| N | | | |
| O | | | |
| P | | | |
| Q | | | |
| R | Grep<br>Touch | Recursive copy<br>Copy the attributes | |
| s | | | |
| t | | | |
| u | | | |
| v | | | |
| w | | | |
| x | | | |
| y | | | |
| z | | | |

## MISCELLANEOUS COMMANDS

| Command | Description |
|---|---|
| **ssh user@ip** | |
| **uname** | |
| **uname –a** | |
| **id** | |
| **du –sh *** | used for finding the disk usage<br>-h for human readable format |
| **ldd executablename** | Shows where all the library fiels are linked |
| **pkg-config** | |
| **cpp -v** | |
| **truss** | This is usually used for debugging the running application. Listing the system call made by the executable at certain point of time. Unfortunately this is not part of POSIX Single unix specification |
| **shutdown** | To shutdown the unix system |
| **gcore** | To generate the core file based on the process id. |
| **cut** | cut –d delimiter, cut –f field, cut –c range |
| **gcc –version** | Shows the version details of GCC |
| **sort** | Sort the given input |
| **uniq** | To remove the repeated lines. |

## List Command

| Command | Description |
|---|---|
| **ls –1** | One by one |
| **ls –R** | Includes Subdirectory |
| **ls -lrt** | |
| | |

## substitution Command

| Command | Description |
|---|---|
| **:1,100s/find/replace/g** | finds and replace the word (all occurances) |
| **:1,100s/find/replace/gc** | **finds and replace the word (confirm each time)** |
| **:%s/find/replace/gc** | Find and replace in entire file |
| | |

## Miscellaneous Command

| Command | Description |
|---|---|
| **passwd** | Changes PWD |
| | |
| | |
| | |

## FIND COMMAND

| Command | Description |
|---|---|
| **find curdir –name “filename”** | finds file with case sensitivity |
| **find curdir –name “filename”`** | finds file with w/o case sensitivity |
| **find . /home/dir1 /home/files –name “filename”** | Search all listed directory |
| | |

## EditOr command

| Command | Description |
|---|---|
| **vi filename** | vi editor |
| **vi file1 file2** | Open Multiple files |
| **ed file** | Line editor |
| | |

## Changing directory

| Command | Description |
|---|---|
| **cd ~** | Return to login directory |
| **cd /** | Returns to the Entire system root directory |
| **cd ~username** | Takes to the other user login directory if permission given |
| | |

## Process status

| Command | Description |
|---|---|

| | |
|---|---|
| **ps** | process status |
| **ps –e** | List the process that is running now |
| **ps –f** | Generates the full listing |
| | |

**Copy command**

| Command | Description |
|---|---|
| **cp –r dir1 dir2** | copy one directory with other |

**Move command**

| Command | Description |
|---|---|
| **mv ../Current directory/* target directory** | Move all files from source director |

**Delete commands**

| Command | Description |
|---|---|
| **rm –r dirname** | Removes the directory |

## CRON TAB

The cron tab is used to do the schedule some activity at the unix like systems. Its was much useful for the administrators for doing some repeated tasks like backup of the system, sending mail,downloading mails at scheduled time etc.

It automates the process of maintenance and administration.

The cron name was derived from the greek name time.

Normal users are restricted for the use of cron tab.

### cron tab:

The expanded form of crontab is cron table. All the shell commands that are needed to execute at regular intervals are placed under the cron table.

Cron can be configured at both user and system level.

Whenever the time and date in the cron tab mathches the current date and time then the jobs will get executed.

Cron tab entry will be

10 15  1 * 1-5 /path/to/scripttorun/

| Field | Description |
|---|---|
| **1st value** | Minute(0-59) |
| **2nd value** | Hour(0-23) |
| **Day of month** | (1-31) |
| **Month of year** | (1-12) |
| **day of week** | (0-6) where 0 is Sunday. In some versions it vary. 7 is Sunday |
| * | All values. For instance if * is configured for day of week. Then it will execute on all days of week |
| - | Used to define the range. if given as 1-5 from 1st day to 5th day |

# APTITUDE

## QUANTITATIVE APTITUDE

### TRAIN PROBLEM

### TERMINOLOGY

#### RELATIVE SPEED

How speed of one object affects the other object

##### Same direction

When two objects move on same direction, then to capture the object running at first, we have to reduce the speed of forward object. This is because, every few KM we move forward, object which is running first also will move forward.

##### Opposite direction

When two objects move on opposite direction, then to meet each other, we have to add the speed of objects. This is because, every few KM we move forward, object which is in opposite will also move forward.

### FORMULA

1. Km/hr to m/sec = x * 5/18
2. m/sec to Km/hr = x * 18/5
3. Time taken by train to cross the pole is equal to time taken by the train to cover the distance L, where L is the length of the train
4. Time taken by train to cross the object of length B is equal to time taken by the train to cover the distance L+B, where L is the length of the train
5. When two bodies running on same direction relative speed is U-V Km/hr
6. When two bodies running on opposite direction relative speed is U+V Km/hr
7. Time taken by cross each other running on opposite direction is (A+B)/(U+V) hr
8. Time taken by cross each other running on same direction is (A-B)/(U+V) hr

# TELECOM AND NETWORKING

## SOCKETS - NETWORKING

Socket is basically an end point of an Inter process communication.

Wonder what is Interprocess communication?

It is the process of communicating between the two threads(process), that run on the system concurrently.

Sockets are bidirectional. That is we can send and receive data at both of the ends. This is point to point communication and communication in socket will happen as connection oriented(stream) or connection less(datagrams).

### Socket address:

Socket address is the combination of IP address and port numbers.

### DIFFERENCE BETWEEN IP ADDRESS AND PORT (REALTIME EXAMPLE 1)

In general terms, socket is something a hollow structure where we can insert something. Eg: electrical socket. This is same for networking too.

For establishing a electrical socket we need a electrical power and switch board. Here the IPaddress is your home and port number is your electrical socket location or electrical socket hole. It may be in kitchen or bed room of your home(IP ADDRESS)

### DIFFERENCE BETWEEN IP ADDRESS AND PORT (REAL TIME EXMAPLE 2):

IP address is address that uniquely identifies your computer. when you are transferring the packet from one computer to other computer. you need a IP address to transfer it. It like a address which we give to courier man. Upon reaching the destination, you need to find for which member of the family the courier has been arrived. It may be for you, or for your brother, mother. How to find that? For that purpose the port number is used. As such there are different members in the family there will be several process running on the system. Here the port number decides for which process the packet has to be delivered. Here the family members can be http, SSH, HTTP secure, web browser. For this some port number will be user defined and predefined also. Predefined are take cared by IANA(internet assigned numbers authority)

### API:

There is no built in programs in c for sockets. Widely there is two API available..

BSD sockets (Berkeley socket distribution)

winsock (Windows sockets)

Socket programming you have written is an part of application layer. All the low level code are covered by Berkeley in BSD socket programming. we use the application programming interface in the application layer.

REFERENCES:

http://www.beej.us/guide/bgnet/output/html/singlepage/bgnet.html#twotypes

```
refer above site and edit it
```

## STANDARDS

### OSI LAYER

OSI stands for open systems interconnection. It is a standard how two machines should communicate each other. OSI says how communication needs to be established and how it needs to be maintained. OSI segregate the data flow between machines A to Machine B in to 7 layers.

Let us take a small example of sending the text message from Machine A to Machine B. Here machine can be laptop, Android mobile etc.

## 1. Application layer

1. Actual data that need to be transmitted is processed at the application layer.

**Protocols:**

HTTP, HTTPS, FTP, SMTP, Bittorrent, CAMEL, Diameter, MAP, SSH,CAP,TCAP

**Example:**

Let say, you want to send a message “Hi” from machine A to Machine B. Here “Hi” is our actual data that need to be transmitted and machine A to Machine B. At sender end, application layers get the message from the user using some application like what’s app, hike etc. and at received end application layer displays the message to end user.

## 2. Presentation layer

It will perform the encryption and decryption of data. Encryption will be performed at source and decryption will be performed at destination

It is called so because it converts the data in to presentable form from encrypted form.

**Protocols:**

TLS(Transport layer security)

**Example**

Let say, I am sending the password to a friend from my phone. Peoples other than me and my friend should not view that password. So encryption and decryption should be performed and that encryption and decryption activity is performed at the presentation layer.

## 3. Session layer

Establishes and manages the session between the source and destination party. Let say when connection is closed then session layer is responsible for reopening of session and when connection is not used for prolong time session layer will close and reopen the connection again.

Apart from this, session layer takes care of dialog control and synchronization. Dialog control is nothing but how transmission is taking place? Whether it is in full duplex or half duplex mode? And synchronization is nothing but adding a check point.

**Protocol**

Session Control protocol, Remote procedure call

## Example

Let say you were transmitting 1000 pages from machine A to Machine B. Session layer will do check point at every 100th page and it sends the acknowledgement to sender.

## 4. Transport layer

Transport layer provides services which are related to transforming the data from one place to other like connection oriented communication, reliability, flow control, congestion.

### Protocols:

TCP, UDP

### Example:

1. When message "HI" is transmitted from one mobile to other mobile, then one may use 4G and other may use 2G. Here sender sends at high speed and received receives at low speed. This flow control is done by Transport layer.

2. When user sends "Hi" and if it is received as "Hello". Then we need to request the data again from sender. This is called reliability checking.

## 5. Network layer (packet)

Network layer is responsible for packet forwarding, routing and QoS(Quality of Service). Frames will be changed as packets and IP address of the destination party is added.

### Protocols

1. IPV4, IPV6

## 6. Data link layer (frame by frame delivery)

Data link layer is responsible for transfers data between adjacent networks in same LAN or WAN.

1. Bits are converted in the frames

2. Error check will be performed on received bits for correctness of data  received on physical layer

3. MAC Address of the source and destination will be added to the frame.

### Protocol:

Ethernet, ARP, MAC

## 7. Physical Layer (bit by bit delivery)

This is layer, in which so far packed information like actual data, IP address, and number is transmitted through physical medium like copper wire, cable, etc. Let say when data is transmitted in wireless network, then how data need to be transferred in wireless network and what protocol should be used when data is transferred in wireless network is part of physical layer.

### Protocol

Bluetooth, USB, DSL, Wi-Fi, Infrared

### Example

Wireless network connected by cell phone tower.

## LOCAL HOST

Before saying "what is local host?", You should know about what is host first of all.

Host or node is nothing but a point in a network. Yes there is relation between the host and local host.

Here the Local host is nothing but that refers to the same node or host.

In other words,

local host is the host name(default host name) given to the ipaddress 127.0.0.1.

Consider you were sending the packet to 127.0.0.1 nothing but the localhost, now the packet you send will be received by yourself.

### MECHANISM BEHIND LOCALHOST:

LOOPBACK is the interface that lies behind the local host mechanism.

what is loop back interface?

one of the main objective of networking is process of  sending and receiving the packet between the hosts.

In LOOPBACK the packets you were sending will not be delivered to your neighbor hosts. It will just deliver to your own machine. yes it like a throwing a ball on a wall infront of you. It will just come back to you.

### NEED FOR LOCALHOST:

If your server and client are at same station(computer) you can use the localhost.

Localhost never enters the network communication. It will be looped back by your system itself.

### CAN WE USE LOCAL HOST WITHOUT NETWORK?

Yes you can very well use it.

what you will do for determining the network connection between the two hosts. Probably you will type just ping followed by ipaddress. Just check it for local host like “ping 127.0.0.1”. Packets will be received on running this command and shows that network connection is available for loopback.

```
ping 127.0.0.1
```

## BASICS

Telecom is a telecommunication network which is used to communicate between the two different areas by means of t

## TERMINOLOGY

### URI

URI is the combination of URL and URN

### SSID

Service set identifier

### NIC

NIC is the network interface card. NIC acts as an interface between the network and computer.

### Packet & circuit switching

Circuit switch transfer the information through dedicated connection where as packet switching transfers the data in terms of packets by chunking the data.

## COMMANDS

### Ping

Ping is used to check whether there is a communication between the two machines. It will not check the communication between to application running two different machines.

### Telnet

This command can be used to check whether the application is listening to some port.

## IP ADDRESSING

### PROPERTIES

1. IP address refer to the physical machine and IP address doesn't refer to the any application in the system.

## WIRELESS LAN

### TERMINOLOGY:

#### AP

Access point

#### BSS

The device which is connected to the access point is called as Basic service set.

### STANDARD

WLAN 802.11

# NETWORK PROTOCOLS

## TCP/IP

### When to use TCP over UDP?

1. If loss of packet is ignorable and if speed takes the higher priority than the loss of packet, then UDP should be preferred.

2. UDP should be preferred for multicast

## MISC PROTOCOLS

### SOAP

Simple object access protocols

# UNCATEGORIZED TOPIC

## PROJECT BASED QUESTIONS

### Explain your current project

#### Big picture of application

My current project is diameter credit control application and basically it is a billing and revenue management application. Then main objective of the application is managing the credit of the subscriber based on the resource usage. Credit can be of any form of monetary value. Any one wants to consume the resource should raise a request to the application and application will undergo an AAA check. First application will authenticate the user once authentication is successful, it will authorize the user for the usage and finally based on the usage application will perform the accounting.

#### Specialized area in big picture

We are part of SCP in Intelligent network in telecommunication, and we are responsible for end to end management of service management to subscriber. Call flow is like SCP-NG which is nothing but Service control point – Network gateway will raise the request to the OCS (Online charging system). This OCS will perform the AAA check and once it passed we will give positive response to the SCP-NG and SCP-NG will indicate the switch to connect the call.

#### Technical Part

Coming to the technical part of the application, it is a multithread application C++ application. We use oracle database to store the information and we used Oracle Pro *C/C++ to establish the communication between application and database. Apart from this to automate some activity in the application we used SHELL scripting. Used some IPC Concepts like shared memory, message queue internally.

#### Role and challenge

We are basically 6 member's team, with 1 lead and 5 developers. We have divided the modules and each individual in the team has to take ownership of end to end of that module. My role in the project is communication establishment between the Network gateway and our application, prioritizing the credits of the subscriber for service usage, establishing the connectivity between database and application, Thread pool implementation.

Challenges I have faced with the above implementation is like database will keep on change. Some customers will ask oracle and some customers will ask for Mysql. So to handle this I have used DAO design pattern which separates the business logic from database and since any component can integrate with our application, I have used Adapter pattern for communication establishment.

### What are some challenging things you faced in  your project?

1. Since we were in to the revenue based application, once we faced very big revenue problem due to some failure in the application. No customers have complained about the issue, so we are not informed about the issue. Finally we found after 1 year like failure is merely due to the application issue and this got very bad name for us. Anyhow we have got bad name and next action plan is how to avoid this problem in future. Keeping track and monitoring every failure in the application is difficult because there can be positive failures like when subscriber purchase a some service with insufficient fund is positive failure but giving a negative response when subscriber have enough fund to purchase a fund is failure of application. The idea I have is like let us develop one generic library for failure monitoring and in this failure monitoring we should able to configure the list of failure threshold percentage. If failures cross certain percentage, then application will alert the business team. This saves lot of future problems instantly.
2. Separating the data access from the business logic
3. Keeping the data in cache for faster access.

## HR QUESTIONS

### Tell me about yourself

#### Personal Information

I am Vinoth Kumar and working as software developer in Plintron. From childhood, I love to work with computers and that made to become a software developer. Well. Coming to the academic side, I had completed B.E computer science and engineering with distinction in Hindusthan College of engineering and technology, Coimbatore.

#### Impressions

1. Apart from regular office works, I am writing one book called "Programming hand notes" and the book is all about C and C++ programming. I have completed about 230 pages in that book.  I used to learn some new things daily about programming and making it as practice by writing as article in the book.
2. I own one blog called arraynotfound.com and I used to post some interesting topics on that.
3. Got two times spot light award with my current employer.

#### Leadership

I love to work with team as well as leading the team.  Some instances in life for leadership is,

1. Upon developing new features in the product, it affected some of the existing features in the project. To avoid this my boss advised the team to use proper design patterns in the product and problem is people in the team had no knowledge about much of the design pattern and I have initiated the weekly technical discussion in team and task is to take design pattern classes and implementing the same in application. As a result of this we have implemented most of the design patterns in our application.

## Any questions need to be asked?

1. Can you give a feed back of my interview?
2. Answers for questions I failed to answer?

## Why should I hire you?

1. I am quick and motivated learner. Mean to say that I will learn the product quickly and I believe that increase the productivity in the team.
2. I love take ownership of the work which is assigned to me. I will take care of end to end activities.
3. Designing an application is art. Here we need to concentrate on reusability, extendibility. If any project is assigned I have experience of designing the product individually by considering all the design principles of OOAD like SOLID Principles and I can create respective design documents.
4. I have worked with product, which handles huge set of customers and I hope that your project will also handle huge set of customers. I will be helpful to the product when we need to increase the scalability.
5. I have knowledge of debugging the load results. If any problem with load tests of your application, I will help you there.

## Why  you are leaving the current company?

1. I want to broaden my professional skills set.

### Why your current is not providing opportunity to enhance your professional skill set?

We can improve our skill set by self but getting hands on experience on that particular skill should be possible while applying it somewhere. My current company cannot change the skills they are using, I Think that won't be good.

### Will you leave us when you want to broaden your skills further?

No, I will not leave because I cannot extend my skill set so long. I need get expertise in my skill set after this  extent. So, this situation will not occur again.

## GIT

### bisect

Which caused the bug?

## ONLINE PROGRAMMING TESTS

### Attitude towards target

1. Plan with some time buffer
2. Focus only on target
3. Use macro wherever necessary
4. Corner case and new program decide
5. Write repeatable logic as function
6. Don’t compile without clearing most of the error

### Macros

```
#define _pe  std::cout<<"["<<__LINE__<<"]  "<<"============"<<std::endl
#define _p(a)  std::cout<<"["<<__LINE__<<"]  "<<a<<std::endl
#define _p2(a,b)  std::cout<<"["<<__LINE__<<"]  "<<a<<" : "<<b<<std::endl;

#define _for(a) for(int i=0;i<a;i++)
#define _for(a,b) for(int i=a;i<b;i++)
```

### Tokenizing the string

```
 string s="HAI, HOW ARE YOU?",temp;
 std::stringstream ss(s);

 std::vector<std::string> l_tokens;

 while(getline(ss,s,' '))
       l_tokens.push_back(s);
```

### Getting input from user

```
int tc;
 std::cin>>tc;
 std::vector<int> l_vecList;
```

```
while(tc--)
{
      int ic;
      std::cin>>ic;
      while(ic--)
      {
            int ip;
            std::cin>>ip;
            l_vecList.push_back(ip);
      }
      /* Solution to problem */
}
```

## FILE EXTENSION AND MEANING

| FILE EXTENSION | EXPANSION |
|---|---|
| .zip | compressed archive file |
| .rar | compressed archive file |
| .mpeg | motion picture expert group |
| .avi | audio and video interleave, developed by Microsoft in 1992 |
| .art | clip art files |
| .bak | backup files |
| .cbl | cobol code |
| .doc | document file for word |
| .gif | graphics interchange format developed by compuserve for web |
| .htm | hypertext markup |
| .mp3 | mpeg audio layer 3 |
| .pdf | portable document file by adobe |
| .xml | extensible markup language |

| | |
|---|---|
| .3gp | 3rd generation partner ship project developed for 3g phones |

## ABBREVATION OF COMPUTER TECHNOLOGY TERMS

| ABBREVATION | EXPANSION |
|---|---|
| ARPA | ADVANCED RESEARCH PROJECT AGENCY (FIRST INTERNET) |
| MICR | MAGNECTIC INK CHARACTER RECOGNISATION |
| PDA | PORTABLE DIGITAL ASSITANCE |
| IP | INTERNET PROTOCOL |
| TCP | TRANSFER CONTROL PROTOCOL |
| .EDU | EDUCTIONAL INSTITUTIONS |
| .COM | COMMERCIL ENTITY |
| .ORG | NON PROFIT ORGANISTAION |
| .NET | NETWORK BASED ORGANISTION |
| TLD | TOP LEVEL DOMAIN(.EDU,.NET.ORG) |
| HTML | HYPER TEXT MARKUP LANGUAGE |
| URL | UNIFORM RESOURCE LOCATOR |
| HTTP | HYPER TEXT TRANSFER PROTOCOL |
| ARPA | ADVANCED RESEARCH PROJECT AGENCY (FIRST INTERNET) |
| MICR | MAGNECTIC INK CHARACTER RECOGNISATION |
| PDA | PORTABLE DIGITAL ASSITANCE |
| IP | INTERNET PROTOCOL |
| TCP | TRANSFER CONTROL PROTOCOL |
| OOAD | Object oriented analysis and design |
| SOA | Service Oriented Architecture |

| | |
|---|---|
| WDSL | Web services definition Language |
| URI | Uniform resource identifier |
| URL | Uniform resource locator |
| URN | Uniform resource name |
| | |

## VIM EDITOR

Abbrevation of VI editor is Visual Editor. VIM is the improved version of VIM.  VI editor is the very old editor which is very difficult to use. VIM gets its popularity by the keyboard shortcuts. We can use the vim without using the mouse that the added advantage.

VIM editor is written by Bram Moolenaar.

It is used in the UNIX like systems .

It is the free and open source software.

Latest version of vim is 7.4 released in Aug 10 2013.

The logo of VIM editor is

### navigation:

| Command | Description |
|---|---|
| **gg** | Goes To 1st line |
| **Shift+G** | Goes to Last line |
| **number** | moves to( current line+number) |
| | |

### search:

| Command | Description |
|---|---|

| | |
|---|---|
| | |
| **:g wordtosearch** | grep command |
| **// word_to_search** | search command |
| **n** | Find next |
| **SHIFT+n** | Find Previous |

## editor commands:

| Command | Description |
|---|---|
| **:d5** | Deletes the 5 lines |
| | |

## Customize commands:

| Command | Description |
|---|---|
| | |
| **q** | Macro recording |

## miscellaneous commands:

| Command | Description |
|---|---|
| **vimdiff file1 file2** | comapare two files and highlight their difference. |
| **:ab abreevation expansion**<br><br>**Eg :ab name My name is vinoth** | This command is used to abbreviate something. If you need to type some large text in your text editor often you can use the abbreviation to insert that large text automatically. First type ":ab abbreviation expansion_of_that" in command mode. whenever you type the "abbreviation" in text editor it will expand to "expansion_of_that" |
| | |

## ABBREVIATION TERMS

| ABBREVIATION | Description |
|---|---|
| **QOS** | QUALITY OF SERVICE |
| **LTE** | LONG TERM EVALUATION |
| **LDAP** | Light weight directory access protocol |

## PRIME NUMBER PROGRAM FOR LARGE NUMBERS

From childhood, we learned about  prime number program. But still we are not efficiently programmed that one.

Here are the some efficiency handled for prime number program.

This program is designed for finding the prime number of  one followed by ten digits.

```
#include<iostream>

#include<math.h>

using namespace std;

int main()

{

   bool flag=true;

   unsigned long  long int
start=1000000000001ULL,end=1000000010000ULL,i,k,sqrt_num;

   for(i=start;i<=end;i=i+2)

   {

      flag=true;

      sqrt_num=(unsigned int)sqrt(i*2);

      for(k=3;k<=sqrt_num;k+=2)

      {

         if(i%k==0)

         {

            flag=false;

            break;
```

```
          }

       }

       if(flag)

       {

          cout<<"\n PRIME:"<<i<<"\n";

       }

    }

    return 0;

}
```

**EFFICIENY HANDLED 1:**

`bool flag=true;`

Use bool datatype instead of int. Because int takes more bytes than bool. This may be simple but saves some bytes.

**EFFICIENY HANDLED 2:**

`sqrt_num=(unsigned int)sqrt(i*2);`

There is a rule for prime number "If the number cannot be divisible by below the sqrt double that number then it is an prime number".

So check up to the sqrt of double that number.

**EFFICIENY HANDLED 3:**

unsigned

use unsigned datatype. Because prime number cannot be negative. It saves the memory.

**EFFICIENY HANDLED 4:**

Iterate the loop starting with 3. Because expect 2 there wont be any even prime number. so no need to check whether number is divisible by 2.

**EFFICIENY HANDLED 5:**

for(i=start;i<=end;i=i+2)

Don't check the prime number for even numbers. Because other that 2, No even number is prime number.

### EFFICIENY HANDLED 6:

```
  for(k=3;k<=sqrt(i*2);k+=2) // Don't do this mistake
```

When you do like this then for every iteration it will call the sqrt function for checking the condition.

It will eat of much off the execution time.

In addition to that there are lot of efficiency can be handled. There is lots of algorithm for prime number such as Sieve of Eratosthenes and so on. This article is focuses only on the writing basic prime number program with efficiency.

## CREATING EFFICIENT PROGRAMS

### function inside the loops:

Don't call a function inside the loops until it become the necessary. For eg calling function as follows takes much of execution time

```
for(int i=0;i<sqrt(num);i++)
```

when you do like this, then the program will  call sqrt function at every time it checks the condition. So create a variable outside the loop and assign the result of that function in that variable and then use that in a loop.

### assigning variable inside the loop:

Don't assign the variable inside the loop. There is no use in that

```
for(i=0;i<10;i++)

{

        k=10;

}
```

This is not applicable when you want to reset the variable before the beginning of the each iteration.

## PUZZLES

1. Alice came across a lion and a unicorn in a forest of forgetfulness. Those two are strange beings. The lion lies every Monday, Tuesday and Wednesday and the other days he speaks the truth. The unicorn lies on Thursdays, Fridays and Saturdays, however the other days of the week he speaks the truth.

   Lion: Yesterday I was lying.
   Unicorn: So was I.

   On which day did they say that?

2. You have a birthday cake and have exactly 3 cuts to cut it into 8 equal pieces. How do you do it?

3. A father and his son are in a car accident. The father dies at the scene and the son is rushed to the hospital. At the hospital the surgeon looks at the boy and says "I can't operate on this boy, he is my son." How can this be?

## COMMON PROGRAMMING MISTAKES

## SEMICOLON AFTER A FOR LOOP:

While typing fastly programmers will keep a semicolon at the end of the for loop.

```
for(i=0;i<10;i++);
```

## STRAY IN the PROGRAM:

cout<<"\n"\";

The stray is the extra character or symbol that is placed in the program. In above case stray "\" is given in cout. This will be marked as red in most compiler. So compiler will return error of "stray '\' in program"

# DATA STRUCTURES

## TERMINOLOGY USED IN DATA STRUCTURE

### Abstract data type(ADT):

ADT is the mathematical model of representing the data types operations, possible values and so on.

**So, what is Mathematical model?**

Representing the system using the mathematical concepts and mathematical language is called mathematical model.

**Now, what is Mathematical Language?**

Mathematical language is representing some information using mathematical vocabulary. Mathematical vocabulary is nothing but equal sign, arithmetic symbols, and range symbol.

For Eg to represent the range of the integer1

-32768 <= x <= 32767

Other Examples:

Operations: addition, subtraction, division

Operations: push, pop

**Data type** is nothing but classification of data based on their characteristics.

Finally, type which is represented using mathematical concepts and mathematical language is called as an abstract data type.

### What is difference between data structure and Abstract data type?

Abstract data type is logical description of type, where as the data structure is concrete description of type.

In simple, ADT is a Plan and data structure is an implementation of that plan.

Data structure implements the ADT.

ADT designs the type based on point of view of user of the data, where as the data structure is designed based on the point of view of implementer of that ADT.

For instance, Integer is an ADT, which simply says possible values and operation that can be performed on the integer whereas, data structure is the implementation of that integer in any of the programming language. Implementation can be of any form such as binary format at low level, using the 2's complement for the storing the negative numbers and so on.

# LINKED LIST

## REVERSE THE LINKED LIST

```
// Pseudocode

reverseDoubleLinkedList(Node **head)
{
 headPtr=head
 PrevNode=head
 currentNode=head


 if(!headPtr)
       return;

 prevNode->next=NULL;

 while(prevNode)
 {
       prevNode->next=currentNode;
```

```
        currentNode->prev=prevNode;
        prevNode=currentNode;
        currentNode=currentNode->next;

  }

  *head=currentNode
}
```

# HASH

## HASH FUNCTIONS

Functions which is used to generate the hash values is called as hash function.

### MESSAGE DIGEST

The name digests and hash value is used interchangeably. The series of versions are MD4, MD5, MD6 etc.

**Properties**

1. Generates the fixed size hash value irrespective of the input size
2. Message digest is used to produce the alpha numeric hash value.

### SHA (SECURE HASH AL GORITHM)

**Properties**

1. Generates the hexa decimal hash value
2. Output sizes will vary based on SHA 512, SHA 216 etc
3. Generates the fixed size hash value irrespective of the input size

## WHIRLPOOL

**Properties**

1. Takes input of length less than 2 power 256 and returns the hash of 512 bits

## MISCELLANEOUS

5381 is used for hashing the data.

# GRAPH

## Representation of graph:

### Components involved in graph

1. Finite set of vertices

2. Finite set of Ordered pair of vertices – This is nothing but the edges between the two vertices.

### Types of methods to represent graph:

1. Adjacency matrix

2. Adjacency list

3. Edge list

4. Incident matrix

5. Incident list

### Adjacency matrix:

In adjacency matrix graph is represented in two dimensional array. If graph contains the V vertices, then the graph is represented as VxV matrix. If you need to find whether there is an path between Vertex V1 to V5, then non-zero value should be present for the array[V1][V5]. If value is present then there will be path between the Vertex V1 and V5. Weight between the two vertices can be represented by placing the weight in the place of 1.

#### Advantages:

1. Fetching an weight or determining whether there is an path between two vertices takes only O(1) time.

#### Disadvantages

1. Takes more memory even if there is low number of edges in the graph.

### Adjacency List:

In adjacency list, graph is represented in array of lined list. Here for each node in the graph, linked list is created to represent the neighbors and it is attached to the node in the array. Here size of the array is equal to the number of vertices

#### ADVANTAGES:

1. Consumes very low memory

#### Disadvanges

1. To check whether there is any path between Vertex V1 to V2, it takes much time when the V2 is the last element of the linked list.

## TREE

Tree is data structure which is used to manipulate the hierarchical form of data.

### ADT

Tree simulates the normal botanical tree, which contains the root

Tree should satisfy the following properties,

1. It should not contain any cycle.
2. No two different nodes should point to same node.

### Data Structure Implementation:

1. Tree is implemented as a collection of nodes, where as each node is a data structure which contains the value and reference to the child nodes.
2. Tree should not contain cycle. I.e., No two nodes have same children

### How Tree Differs from graph:

| Tree | Graph |
|---|---|
| Tree is an special form of graph with some conditions. | Graph is not special form of tree |
| There is only one path between any two vertices in the tree. | There can be multiple path between any two vertices in the graph |
| Tree will not contain cycle,circuits,trail | Tree will contain cycle,circuits,trail |
| Tree will contain root node and each child will have exactly one parent | No such concept of root node in graph |
| Parent child relationship is present in graph | Parent child relationship not present in graph |
| There will be n-1 edges in the graph, where n is the number of nodes in the graph | We cannot predict the edges in the graph based on the nodes. |
| Tree is hierarchical model | Graph is network model |

### Tree Variants:

#### Planted Tree:

Tree with vertex degree of the root vertex was one is called as planted tree.

## Classification of trees:

### 1. Based on how nodes are directed

a. Directed tree

b. Undirected Tree

### 2. Based on how nodes are ordered

a. Ordered Tree

b. Un-ordered Tree

### 3. Based on root of the tree

a. Rooted tree

   i. In a tree, one special node is designated as root.

b. Free tree

   i. No node in a tree, is mentioned as tree.

## Ordered tree:

Ordered tree is an oriented tree, in which children of a node is ordered in some fashion.

There are three types of ordered tree

1. Fibonaacci Tree
2. Binomial Tree
3. K-ary Tree

### Fibonacci Tree:

1. Fibonacci tree is a collection of tree satisfying minimum heap property. This implies that minimum key is always at the root of one of the tree.
2. Fibonacci tree has no prescribed shape
3. Fibonacci tree is an collection of tree. At extreme situation the fibonacci tree can have each element of the tree to be added as a separate tree

4. Lazy operation: Since individual element can be added as a one tree in the fibonacci tree, some operations can be executed by postponing the work. For example merging heap is done by simply merging the two list
5. It is named so, because fibonacci numbers is used in running time analysis
6. Root of all the tree is linked with circular doubly lined list
7. For each node number of children  is marked
8. Pointer to root containing the minimum key is marked
9. This is also called F-heaps

## Relation of Tree with Graph

1. Trees are acyclic graph
2. Any two vertices in the tree, will have the single path, where as this is not true in the case of graph

## Terminology:

### Sub-Tree:

A tree S, is said to be an sub-tree of Tree T, when an root node of the sub-tree is present in the Tree s, and all the descendants of the root node of sub-tree is exactly same as Tree s.

There are type types of sub-tree

1. Entire tree

   When root node of sub-tree and root node of Tree is **same**, then it is called as entire sub-tree

2. Proper sub-tree

   When root node of the sub-tree and root node of Tree is **different**, then it is called as entire sub-tree

#### Number of SUB TREE OF A nodE:

Number of node connected to the current node is called number of subtree of node.

### Tree Elements:

| | |
|---|---|
| Root | Top most node of the tree |
| Child | Node directly connected to some node, when moving away to top |
| Sibling | Node with same parent |
| Descendant | Node, which is reachable by repeated proceeding from parent to child |
| Ancestor | Node which is reachable by repeated proceeding from child to parent |
| Leaf | Node with no children |
| Internal node | Node with at least one children |
| External node | Node with no children |
| Degree of Node (or) Vertex Degree | Number of sub-tree of a node is called degree of node.<br>Number of sub-tree in a tree is called degree of tree |
| Edge | Connection between one node to other |
| Path | Sequence of node connected together |
| Level | 1+ No of connections between node and root |
| Height of node | Number of edges on the longest path from current node to leaf node |
| Depth of node | Number of edges from root node to the current node |
| Height of Tree (or) Depth of tree | Number of edges on the longest path from root node to the leaf node is called Depth.<br>Number of edges on the longest path from leaf node to the root node is called height.<br>Both are same when used for tree but they were different when calculated for particular node. |
| Forest | More than one disjoint trees is called forest<br>Usually single node can be added to link the multiple sub-trees to form the forest. |
| Rank | Position of the node among the nodes in the tree is called rank. Rank is calculated based on the pre-order traversal method. I.e., first node in pre-order traversal takes the first rank. |
| Select | In select process we will be given a rank, and we need to find the position of the node from the root value. |
| weight | |
| Rooted Vertex | Node is also called as vertex. So root node is nothing but the Root vertex |

| | |
|---|---|
| Vertex | Node is also called as vertex |
| Weight (or) Cost | Effort required to travel from one vertex to other. |

## Tree based Data Structure:

### Heap

Heap is a specialized tree based data structure where nodes are arranged with respect to the heap property. Heap property states if Node A is a parent node of Node, then key of node A is ordered with respect to the key of Node B. This ordering is followed across the heap.

#### Relation of heap with binary search tree:

Binary search is different from heap. In binary search tree there is a rule that element which is greater than the root node is placed right hand side and element which is lesser than the root node should be placed on the left hand side. But there is no such rule is heap. In heap relation is among only child, parent and grant parents.

There are two types of heap property

a. Maximum Heap Property

   Keys of the parent node is always greater than the child nodes

b. Minimum Heap Property

   Keys of the parent node is always lesser than the child nodes

## Traversal in tree:

Since tree is a non-linear data structure, it can be traversed in many ways.

### Classification based on the height and depth

1. Depth wise traversal
   a. Inorder
   b. Preorder
   c. Postorder
2. Breath wise traversal

### Depth wise traversal

Traversal will be concentrated from top to bottom. Naming convention is based on the root of the tree.

## Inorder

Here nodes will be traversed based on the following form

```
Traverse the Left subtree
Display the current node
Traverse the Right subtree
```

Since root is at the middle it is called in-order traversal.

```
 void inorder(TreeNode *p_root_node)
    {
        if(p_root_node->m_left_node != NULL)
            inorder(p_root_node->m_left_node);

        m_traversed_list.push_back(p_root_node->m_key);

        if(p_root_node->m_right_node != NULL)
            inorder(p_root_node->m_right_node);
    }
```

## Preorder

Here nodes will be traversed based on the following form

```
Display the current node
Traverse the Left subtree
Traverse the Right subtree
```

Since root is at the beginning it is called pre-order traversal.

```
  void preorder(TreeNode *p_root_node)
  {
        m_traversed_list.push_back(p_root_node->m_key);

        if(p_root_node->m_left_node != NULL)
            preorder(p_root_node->m_left_node);

        if(p_root_node->m_right_node != NULL)
            preorder(p_root_node->m_right_node);
  }
```

## Postorder

Here nodes will be traversed based on the following form

```
Traverse the Left subtree
Traverse the Right subtree
Display the current node
```

Since root is at the end it is called post-order traversal.

```
void postorder(TreeNode *p_root_node)
 {
     if(p_root_node->m_left_node != NULL)
         postorder(p_root_node->m_left_node);

     if(p_root_node->m_right_node != NULL)
         postorder(p_root_node->m_right_node);

     m_traversed_list.push_back(p_root_node->m_key);
 }
```

## Common uses of Tree

1. Used for representing the hierarchical data
2. Storing in the way it is easily searchable

## Common operations which can be performed in tree

1. Insert an node
2. Delete a node
3. Search for node
4. Pruning: removing the section of the tree
5. Grafting: adding the section of the tree.

## Spanning tree

Spanning tree is a new graph formed by altering some edges from another graph, without removing any vertices.

Consider the following graph,

## REPRESENTATIONS

### REPRESENTING TREE IN ARRAY

#### Formula

1. If height of the tree is H, then array size required will be (2 )-1

2. Let say you are in node N, place the left child of Node N by using formulae 2*N and place the right child for the Node N by using the formulae 2*N+1. Note, that your root node should start with 1.

**Note:**

1. Since array size cannot be modified, while creating a array maximum size should be used, so that nodes for tree can be added to the tree in future.
2. This representation will not fill all elements in the array when tree is not a complete tree.
3. This array representation will be helpful for complete binary tree.

## AVL TREE

AVL tree is a balanced binary search tree.

### Why AVL Tree is better than normal binary search tree?

AVL tree makes it tree balanced during every insertion and deletion, so search in AVL tree is better when compared to normal binary search tree.

### INSERTING NODE IN AVL TREE

**Steps**

1. Insert an element as like inserting an element in binary search tree.  Let that node be 'x'
2. Starting from node x, travel towards root and while passing every node check whether sub-tree rooted at current node is balanced or not.
    a. If sub tree rooted at current node is balanced, then insertion is successful
    b. If sub tree root at current node is not balanced, then perform the rotation.  Let 'Z' be the node which is unbalanced and 'Y' be the node which is child of node Z and 'X' be the grand child of node 'Z'. There can be at most two children and at most 4 grandchildren for node Z. Child and grandchildren has to be selected in the way we have travelled from bottom to top.
        i. Left Left rotation: If C is left child of Z and G is left child of C
            A. Perform the right rotation,

| | CURRENT NODE – 'Z' | CHILD NODE OF Z – 'Y | GRAND CHILD OF Z – 'X' |
|---|---|---|---|
| NODE | Becomes left child of Y | Becomes right or left child | Remains in same position |

| | | | |
|---|---|---|---|
| | | of Z's previous parent | |
| LEFT NODE | Link will broken | Remains in same position | Remains in same position |
| RIGHT NODE | Remains in same position | Becomes left child of Z | Remains in same position |

ii. Right Right rotation: If C is right child of Z and G is right child of C

A. Perform the right rotation,

| | CURRENT NODE – 'Z' | CHILD NODE OF Z – 'Y | GRAND CHILD OF Z – 'X' |
|---|---|---|---|
| NODE | Becomes left child of Y | Becomes right or left child of Z's previous parent | Remains in same position |
| LEFT NODE | Remains in same position | Becomes left child of Z | Remains in same position |
| RIGHT NODE | Link will be broken | Remains in same position | Remains in same position |

iii. Left Right rotation: If C is left child of Z and G is right child of C

A. Perform left rotate and after that perform right rotate

iv. Right Left rotation: If C is right child of Z and G is left child of C

A. Perform right rotate and after that perform left rotate

## SNAPSHOT OF INSERTION AND DELETION

| | INSERTION | DELETION |
|---|---|---|
| OPERATION | AVL ROTATION | AVL ROTATION |
| Z | Unbalanced node | Unbalanced node |
| Y | Child of Z in the travelled path | Greater height child of Z |
| X | Child of Y in the travelled path | Greater height child of Y |
| ANCESSTOR FIX | NOT REQUIRED | REQUIRED |

# RED BLACK TREE

Red black tree is kind of self balancing binary search tree.

### Why RB tree over AVL tree?

Red black tree is kind of AVL tree and it has less number of rotations when compared to the AVL tree during insertion and deletion. When insertion and deletion is less, then AVL tree must be preferred.

## Properties of Red black tree

1. Every node in the tree is either black or red
2. Root node of the tree is always black
3. If node is red, then both children of the node are black.
4. Unique path from node to all descendent of the node should have equal number of black nodes.
5. The entire leaf node should be black.

## INSERT A ELEMENT IN RED BLACK TREE

### Steps:

1. Insert an element as like inserting a element in binary search tree and color the node as red. Let as name this node as 'A'. Add two NULL nodes to the node A and color it as black. If the node A is root of the tree, then simply color it as black and finish the insert operation.
2. Check the color of the parent node of A.
    a. If A's parent node color is black, then finish insert operation.
    b. If A's parent node color is red,
        i. Red Parent, Red uncle: If A's uncle node color is red, then color A's parent and uncle node as black. Color the A's grandparent as red. Change A as A's grandparent and go to step 2.
        ii. Red Parent, Black uncle: If A's uncle node color is black, then perform rotation (as like AVL Rotation) for Node A, Parent of Node A and grand parent of Node A.

## DELETE IN RED BLACK TREE

### Steps

1. Delete the node as like deleting the node in binary search tree. Let u be the node that is going to be deleted and v be the node that replaces the node u.
2. Check the color of the node u and v

a. Black and red combination: If color of node v is red, then just replace it with black. Note that color u will be black, since it cannot be red, if so then red black tree property is not satisfied.

b. Red and black combination: If color of node u is red, then do nothing. Note that color v will be black, since it cannot be red, if so then red black tree property is not satisfied.

c. Double Black: If color of both u and v is black, the mark u as double black. Now the property will get violated. Let S be the sibling of Node U and let P be the parent of Node S. Consider the color of the sibling node S,

   i. If S is black and one of the sibling children is red. Color that node as R. Then perform rotation for node P,S,R

   ii. If S is black and all the sibling children is black. Marl the node S as red and go to P. Got to step 2.c

   iii. If S is red, then perform double node rotation. Perform double node rotation for node P and S. Change the color of node P and S. If it is red mark it as black and vice versa. Go to step 2.c

## SNAPSHOT FOR RED BLACK TREE INSERT AND DELETION

Let U be the node that is to be deleted and V be the node that replaced the node U.

| | INSERT | DELETE |
|---|---|---|
| OPERATION BASED ON | UNCLE of node U. | SIBLING of node U. |
| RED AND RED CASE | Mark both as black and mark grandparent of node U as red. | Do nothing |
| BLACK AND BLACK CASE | Do nothing | Check color of children of sibling.<br><br>One of the sibling children is red, perform rotation. |
| BLACK AND RED CASE | Do nothing | Check color of children of sibling.<br><br>Definitely both children of sibling should be black, perform double node rotation for node S and parent of Node S. |
| RED AND BLACK CASE | Perform rotation | Do nothing. |

## MISC DATA STRUCTURES

## BINARY SEARCH TREE

### Delete a node:

#### Node with no children

Simply assign the null to the parent node of node to be deleted.

#### Node with one children

Link the children of node to be deleted, to the parent of node to be deleted.

#### Node with two children

To delete a node follow the steps

Replace the node to be deleted with the node’s in-order predecessor or in-order successor.

Repeat the step1 until no duplicate node in the tree.

## DATA STRUCTURES

### TERMINOLOGY

#### Linear and non-linear data structure

Linear data structure is way of organizing a data in a sequential fashion

For example: array, linked list, stack, queue

Non-linear data structure is one in which data are arranged in random fashion

For example: Multi-dimensional array, tree, graph

#### Dynamic Data structure

This is similar to dynamic memory  allocation. Memory will be allocated at run time.

### Stack

Stack is an last in first out data structure.

#### When to use data structure?

When data that which is inserted last need to be processed first then you should use the stack data structure.

### Practical application of stack

1. Evaluating the expression.
2. Backtracking

## DIFFERENCE AMONG SIMILAR DATA STRUCTURE

### QUEUE AND DEQUE DIFFERENCE

1. Queue should be used when push happens at back and pop happens at beginning of the string.

## BINARY TREE

### LEVEL ORDER TRAVERSAL

#### Algorithm

1. Push the root element to queue
2. When queue is not empty,
    a. Get the element from queue
    b. Push the left element of tree to queue
    c. Push the right element of tree to queue
    d. Print the current element
    e. Go to step 2

### B+ TREE

1. One node can point to n number of children node
2. Each node will contain ordered list of keys and pointer to children nodes
3. It has keys and value type
4. To insert it finds the adjacent key and searches for value in between
5. Reduces the traversal, since keys are maintained as group.

## REAL WORLD PROBLEMS

### List of data structure to be used for real world programs

#### Graph

a. Travel through multiple combinations.

#### Backtracking

a. When brute force is only option, then we can go for backtracking.

#### Tree

a. When decision should be made on two options.

# ALGORITHM

## ANALYSIS OF ALOGORITHM

Computing the running time and space required to run the algorithm is called as analysis of algorithm.

## ASYMPTOTIC ANALYSIS

### Difference between Big O and small o

1. Growth rate will vary between Big O and small o
2. Big O is grows no faster than constant K, where as Small o grows strictly slower than K.
3. Big O is grows <= constant K, where as Small o grows < K.

## ALOGORITHM METHODOLOGY

## DYNAMIC PROGRAMMING

Dynamic programming is also called as dynamic optimization.

Dynamic programming stores the solution of the sub problems and give it back whenever it is required.

## REAL WORLD PROGRAMS

## SELECTION OF ALOGORITHM

### Dynamic programming

Divide the problem and store the solution of sub problem for next usage.

## DATA STRUTURE ORIENTED

## FIND NEXT LARGEST ELEMENT IN ARRAY FOR ALL ELEMENTS

### Problem

To find the next greater element in array in the appearance of array

### Examples

10 5 25 12 2

10->25

5->25

25->12

12->2

### Algorithm

1. Push all elements to stack
2. Initialize current element to -1
3. Pop the last element and compare to current element
    a. Make largest element of popped element as current element
    b. Compare current element with popped element
        i. When current element is less than popped element replace current element with popped element
    c. Go to step 3

### Pseudocode

## STACK BASED ALGORITHMS

### TERMINOLOGY

**Infix Expression**

When operator lies between the operand, then it is called as infix expression.

**Prefix Expression**

When operator lies between before the operands, then it is called as prefix expression.

**Postfix Expression**

When operator lies between after the operands, then it is called as postfix expression.

### INFIX TO POSTFIX

**Steps:**

1. Prepare the priority table for the list of operators used in the expression excluding the right and left parenthesis operator.
2. Pick element in the expression one by one and do any of the following,
    a. When picked element is operand, then don't push to stack. Just print that element on screen
    b. When picked element is operator, then compare the precedence of picked operator and operator which is on the top of the stack.
        1. If priority of operator which we picked is lower, then pop the element in stack and print it. Go to step 2.b.1  and repeat the process.
        2. If priority of operator which we picked is higher, then push the element in to the stack.
    c. When picked element is left parenthesis then push to stack
    d. When picked element is right parenthesis then pop all the elements in the stack until left parenthesis is found.

### INFIX TO PREFIX CONVERSION

### Steps

1. Reverse the given expression
2. Replace left parenthesis with right parenthesis and vice-versa.
3. Perform the infix to post fix conversion
4. Again reverse the expression.

## REVERSE STACK USING RECURSION

### Algorithm

```
1. If stack is empty, return
2. Until stack is empty
    2.a If not empty
        2.a.i pop the element and keep locally
        2.a.ii Go to step 1
    2.b If empty go to step 2

3. When stack is empty
    3.a Push the local element and enter exit
4. Check status of stack
    4.a If empty, Pop element in stack and keep locally and go to step 4
    4.b If not empty, local element which we got in step 2.a.i
    4.c If not empty, push locally kept element to stack which we got from 4.a
5. Exit
```

## CHECK FOR BALANCED EXPRESSION

### Algorithm

```
1. Build bimap with the list of brackets and associated closing brackets
2. Read the expression character one by one
    2.a If expression character is in brackets list
        2.a.i. if opening brace push to stack
        2.a.ii. if closing brace pop from stack
            2.a.ii.I Seach for corresponding value in brackets list
                2.a.ii.I.A If present, continue to step 2
                2.a.ii.I.B If not present, print Failure and go to step 4
3. If stack is empty
    3.a Print success
    3.b Print Failure
4. Exit
```

## SORTING ALGORITHM

### INSERTION SORT

### Algorithm

```
//Theoritical
Loop from 1 to n
 Start swaping from last sorted element and stop when it placed at right place
Assign the current element in found sort pos

//Code wise
//Ascending order
for i in 1->n
```

```
j=i-1
key=arr[i]
while j > 0 and  arr[j] > arr[i]
      arr[j+1]=arr[j];
      j=j-1
arr[j+1]=key
```

### HEAP SORT

Heap sort is arranging the elements by max heap property or min heap property and sorting the elements.

**How heap sort works?**

1. Arrange the elements as a binary tree and apply the heap property. (For ascending order use Min heap property and for descending order use max heap property)
2. Once arranged, replace first and last element in the tree and remove the last element in tree
3. Continue the process until you were able to form the tree.

## TRAVELLING SALES MAN PROBLEM

Travelling sales man problem is a problem where will be having a list of cities and distance between the each pair of cities and task is to find the shortest path by visiting all the cities exactly once and we have to reach the city from which we have originated.

## PROGRAMMING CONCEPTS BASED ALGORITHM

### STACK RELATED ALGORITHMS

```
// Pseudocode


for each character in word
 stack.push(character)

while(stack.notempty())
 character = stack.pop();
 print character
```

## ARRAY RELATED ALGORITHMS

### PRINT FROM KTH LARGEST NUMBER IN DECREASING ORDER

1. Using bubble sort to sort up to k elements alone
   a. Time complexity : O(nk)
   b. Space Complexity: O(n)
2. Use max heap tree sorting to sort first k elements
   a. Time Complexity : O(n+ klogn)
      i. Reason: To sort the element up to K using max heap sort, complexity is k logn
3. Restricted box of size K: Fill the first k elements in box. Find the minimum element in box. Compare the minimum element.
   a. Time Complexity: O( (n-k)k + klog k)
      i. Reason: To find the minimum element at every iteration it takes (n-k)k. To sort the final result k elements in box it takes k logk
   b. Space Complexity: O(n)

## TREE RELATED ALGORITHMS

### CONVERT BINARY TREE TO LINKED LIST

```
changeBinaryTreeToLinkedList(rootNode,parentNode)

 newNode = buildNewNode(rootNode)

 call recursively when node->left !=NULL

 print currentnode

 newNode->prev=parentNode
 parentNode->next=newNode


 call recursively when node->right != NULL
```

```
Program:

changeBinaryTreeToLinkedList(Node *node,parent)

       newNode=buildNewNode(node->data)

       if(node->left != NULL)
             traverse(node->left,newNode)

       print node->data

       newNode->prev=parent
       parent->next=newNode


       if(node->right != NULL)
             traverse(node->right,BuildLinkedList node(node) )
```

## LINKED LIST RELATED ALGORITHMS

### INSERT A NODE IN SORTED CIRCULAR LINKED LIST

```
//THEORITICAL

Loop until node data is lesser than given data
form new node
new node next is current node next
new node->next=currentNode-next
currentNode->next=new node


// PROGRAM

node=*head
if(*head==NULL)
{
 NewNode=buildNewNode(data);
 *head=&NewNode;
 return;
}


if(!node->next && givenData < node->data)
{
 buildNewNode(givenData)
 *head=Newnode
 Newnode->next=node
}

while(node->next && node->next->data < givenData )
{
 node=node->next
}

buildNewNode(givenData)
Newnode->next=node->next
node->next=NewNode
```

## REVERSE LINKED LIST IN GROUP

```
Node* reverseInGroup(Node *node,int count)

 lcount=count-1
 lNode=node;

 while(
 lcount-- &&
 lNode &&
 lNode ->next !=NULL
 )
 {
       (lNode->next)->next=lNode;
       lNode=lNode->next

 }

 if(lNode->next != NULL)
       node->next=reverseInGroup(lNode,count);
 else
       node->next=NULL;

 return node;
```

## MERGE TWO SORTED LINKED LIST AS SORTED LINKED LIST

```
//THEORITICAL

MergeLinkedLIst(Node* list1,Node* list2)

 take data from both list
 take decision of which list data need to be added

 change the node of list from which data is taken
```

## SPLIT CIRCULAR LINKED LIST IN TO TWO

```
//THEORITICAL

POint the head to two nodes as backward and advacing nodes
increment backward by 1
increment advance by 2


// PROGRAM

splitCircularLinkedList(Node* node,Node *backwardHead,Node *advancingHead)
{
 advancing=backward=node;
 while(backward->next !=head &&
 advancing->next != head && advancing->next->next != head
 )
 {
       backward=backward->next;
       advancing=advancing->next->next;
 }

 if(advancing->next_>next !=head)
       advancing=advancing->next

 advancingHead=backward->next;
 advancing->next=advancingHead;

 backward->next=node;
 backwardHead=node;
}
```

# BATCH SCRIPTING

## PROPERTIES OF BATCH

### File extenstion

.bat

## GRAMMAR

### General commands

#### Echo

To print the given content on console

```
echo “hai”
```

#### To turn off the echo statements

```
@echo OFF
```

### Dereferencing the variable

```
echo %myVar%
```

### xcopy

```
xcopy "Source path" "Destination path"
```

## Differences

### Difference between % and %%

% is used to expand the variable in command line, whereas %% is used to expand the variable in the scripting language.

# PYTHON

Python is object oriented, general purpose, interpreted, high level programming language.

Compilation: python filename.py

Extension: py

## INPUT AND OUTPUT

### stdout

```
print 'Hai Python single quotes'
print "Hai Python double quotes"
print ('Bracket print')

a=10
b=20
print "Hai I am priting variable"%a
print "Hai I am printing two variables as A %d B %d" % (a,b)
```

# SHELL SCRIPTING

## TERMINOLOGY

### Difference between sh and bash

Sh is the specification of shell, where as bash is implementation of shell.

## BASICS OF SHELL SCRIPTING

Shell is the software used to interact with the operating system. to the UNIX operating system.

There are different flavors of shell available based on the operating system. Windows default shell is explorer.exe and bash is one of the shell in the unix.

There are two types of shell

a. Bourne shell($)
b. C shell (%)

Types of Bourne shell

a. Bourne shell(sh)
b. Bourne against shell(bash)
c. Korn shell(ksh)
d. Posix shell(sh)

## Shell Prompt:

Prompt issued by the shell to enter commands or programs.

## shebang:

shebang is the line which starts with the pound sign(#) and Bang symbol(!). The shebang will tell the system to interpret the script based on the specified interpreter. Interpreter should be mentioned after the shebang symbol

```
#!/usr/bin/sh
```

The above line says that to interpret the script based on sh shell script.

Shebang is optional. If not specified then the default sh shell will be used.

## VARIABLES:

Usually variable names should be in CAPS

To make the variable readonly use the prefix readonly before the variable name.

To unset the variable use command unset

Special character cannot be used in the variable name. They are reserved for specific purpose for example $ is used for process id. echo $$ gives the process ID.

```
# $# gives the list of arguments passed
 if [ $# = 0 ]; then
    echo "No Arguments passed!"
    exit 1
 fi
 echo "Your current process number is "$$


 # $@ and $* takes the entire arguments as single list difference is $* takes
entire arguments as single list separated
  by spaces. where as the $@ also takes the entire argumengts but each arguments
can be spliited
 echo "Argument passed "$@
 echo "Argument passed "$*

 echo "Demo of \$@.."
 for V_TOKEN in $@
 do
    echo $V_TOKEN
 done

 echo "Demo of \$*"

 for V_TOKEN in $*
 do
    echo $*
 done

 # $? will say about the exit status of the program

 true
 echo "Last Exit status is "$?

 false
 echo "Last Exit status is "$?
```

## Arithemetic operations in shell scripting

```
#Bourne shell doesnt have the inbuild arthimetic operations it uses programs like
expr or awk

a=10
b=5

echo "Addition of variables"`expr $a  + $b `
echo "Multiplication of variables"`expr $a  \* $b `

#equality of variables

if [ $a == $b ]; then
 echo "Values are equal"
elif [ $a != $b ]; then
   echo "Values are not equal"
```

```
fi
```

### Loops in shell

```

#while loop

a=10

while [ $a -gt 5 ]
do
   echo $a
   a=`expr $a - 1`
#a=$((a-1)) Doesnot work on many shells
done


a=10

#for loop

echo "Demo of for loop"

for V_OUTER_VAR in 10 20 30
do
   for V_INNER_VAR in 40 50
   do
     echo "Iter: $V_OUTER_VAR $V_INNER_VAR"
     if [ $V_OUTER_VAR -eq 10 -a $V_INNER_VAR -eq 40 ]; then
         echo "Breaking..."
         break 2 # breaks two loops
     fi
   done
```

### Including header files for shell scripting

It is mainly used when functions are written in one file.

```
bash script_name.sh
source script_name.sh
```

## DESIGN PRINCIPLES

## SOLID

```
S- Single responsibility principle
O- open closed principle
```

```
L- Liksov substitution principle
I – Inversion of control
D – Dependency inversion principle
```

## DESIGN PATTERNS

### DESIGN PATTERN TERMINOLOGY

#### Double Dispatch

```
class Vehicle
{
 void drive(DriveVisitor*);
};

class Car: public Vehicle
{
 void drive(DriveVisiotor *anyDriver)
 {
       anyDriver->drive(this);
 }
};




class DriveVisitor
{
 void drive(Vehicle *anyVehicle);
};

class ProfessionalDriver : public DriveVisitor
{
 void drive (Vehicle *anyVehicle)
 {
       std::cout<<"Implementing ProfessionalDriver";
 }
};

class AmaeturDriver : public DriveVisitor
{
 void drive(Vehicle *anyVehicle)
 {
       std::cout<<"Implementing AmaeturDriver";
 }
};


int main()
```

```
{
 DriveVisitor *prof_driver= new ProfessionalDriver;
 DriveVisitor *amaetur_driver= new AmaeturDriver;


 Vehicle *anyVehicle=new Car;
 anyVehicle->drive(prof_driver);
 anyVehicle->drive(amaetur_driver);
}
```

## VISITOR PATTERN

1. Making an data structure and operations that has to be done on data structure separate is called visitor pattern
2. Ultimate aim of the visitor pattern is like any new operation that is introduced in the feature should not affect the existing data structure.
3. Visitor pattern works on basis of double dispatch

### How to avoid down casting in programs?

With help of visitor pattern, we can avoid the down casting in programs.

# EXCEL-VBA PROGRAMMING

## PROGRAMMING CONCEPTS

### Comparision with other language

1. It won't terminate with semi-colon unlike c,c++,java

### Sub-routine

Sub routine is a function, which does a some specified job

```
sub function_name()
 MsgBox "Hello World"
end sub
```

If function belongs to the particular object, then it should called with .(dot) operator

### Comment line

```
' This is comment line in vba
```

### Variables:

Variable are used to store some values.

```
Dim variablename as Type
Dim myNumber as Integer
Dim MyText as String
```

### Condtional Logic:

```
If condition then
  Code Logic
ElseIf condition2
 Condition2 Logic
Else
 Final Default Logic
End If
```

### Range in VBA:

Range is one of the inbuilt object in the excel, which helps us to manipulate on the range in excel.

```
Range("A1") ' Single cell range
Range("A1:B2") 'Multiple cell range
Range()
```

# MISC SUBJECTS

# BANKING

## TERMINOLOGY

### LIQUIDITY

Ability to convert to cash quickly.

### LIQUIDATION

Process of bringing the business operation to end and distributing its capital to shared holders and depositors by dividend.

### Counter party

One who involved in opposite of transactions.

### Risk weighted assets

Process of giving weight to assets of bank is called as risk weighted assets.

### Off balance sheet

Assets or earning which is not included in balance sheet is called as off balance sheet.

### Capital Adequacy ratio(CAR)

Ratio of bank capital to risk.

### Equity

Value or share of the security.

### Financial Instruments

Financial instrument are monetary(Money) contracts between parties. This can be deposits, commercial paper, shares etc.

### Financial security

Type of financial instrument which can be traded at stock exchange.

### Sub prime borrowers

Who were not able to pay the loan as promised

## PAYMENT

## GROUPS AND CORPORATIONS

### NPCI – National payment corporation of India

It is the master group and contains small groups such as NACH, IMPS, RuPay, and UPI for managing the payment which happens in the India.  NPCI is like a college and NACH, IMPS, RuPay, UPI is one of the departments in college.

It is also called umbrella organization for all payment system in India.

Example: Paying a salary to the employee, paying for online shopping etc.

Headquarters: Pune, Maharashtra.

#### NACH- National automated clearing house

This group is responsible for automating the financial transactions which is repetitive and periodic in nature.

Example:

Paying a salary to employee is one of the repetitive actions for every month. This can be automated with help of NACH.

### AADAR PAYMENT BRIDGE:

People can link the aadhar account to the bank and government will provide the subsidy directly to the people via bank linked to the aadhar account through aadhar Payment Bridge.

## IMPS – Immediate payment system

This is one of the way to transfer the money from one bank to other bank in 24*7. Unique feature is it will beneficiary account will get credited instantly.

Example: If you want to transfer the amount instantly to your father account at midnight then you can use this facility.

### MMID- Mobile money identifier

IMPS can be done via internet banking and ATM. Usually account number will be used for transferring the fund from one account to other account. If required, one can generate the MMID uniquely for the account and transfer can be done from one MMID to other MMID instead of using the account number.

First four digit of MMID uniquely identifies the bank.

Simple Answer: MMID is like another unique account number or transfer number.

## UPI-Unified payment interface

Are you bored with entering account number, IFSC code, beneficiary name?

Are you bored in installing multiple bank accounts in the mobile phone?

Don't worry UPI solves your problem.

This is one of the way to transfer the money from one bank to other bank in 24*7.

UPI is service which is used to manage all your bank accounts in single application and no need to type the account number for transferring the fund to the beneficiary.

Simply type beneficiary as vinothcse123@axisbank.com and transfer the amount.

### Unique features

1. 24*7 payment
2. Virtual address (vinothcse123@axisbank.com) for payment.

## Bharat Bill payment system-BBPS

It is initiated by RBI for co-coordinating bill payments in India. Peoples like airtel, TNEB will contact the BBPS to setup the bill payment.

Example: Paying a telephone bill, paying an electricity bill.

## NATIONAL INSTITUTE OF BANK MANAGEMENT

It is institution which involves in research, educate, training for banking related payments.

Example: People want to transfer the fund to beneficiary at midnight.  What they will do? This kind of research activity will be done by this group.

## CHEQUE

Medium of exchange for money between banks.

## TERMINOLOGY

### Legal amount:

Amount which is written in words is called legal amount.

### Courtesy amount

Amount which is returned in number is called courtest amount.

### CTS-Cheque truncation system

CTS are cheque introduced by the RBI for faster clearance.

## Types of cheque

| Cheque Type | How to identify? | Who can collect cash? |
|---|---|---|
| Bearer or open cheque | Bearer word in the cheque is not cancelled | Anyone can collect cash from bank. |
| Order cheque | Bearer word in the cheque is cancelled and “or order” | |
| Crossed cheque | Two crossed lines on face of cheque | Cannot take cash in hand. Only it can be transferred to recipient bank accounts. |
| Self cheque | The word “self” is written in place of payee name | One who owns the account can withdraw cash. |

### Ante date cheque:

Date which is written on the cheque is less than the current date is called ante date cheque.

### Post date cheque:

Date which is written on the cheque is higher than the current date is called ante date cheque.

### Stale cheque:

Cheque which is issued to pay after three months is called stale cheque.

### Truncated cheque:

Hard copy of cheque is converted to soft copy by special tools is called truncated cheque.

This is used to transfer the cheque from one bank to other bank without carrying the cheque physically.

## CLASSIFCATIONS AND CATEGORY

### CLASSIFICATION OF MSME ENTERPRISES

#### MSME – Micro small and medium enterprises

| ENTERPRISE | Manufcaturing sector LIMIT | Service sector |
|---|---|---|
| Micro enterprises | <25 Lakhs | <=10 lack |
| Small enterprises | >25 lack and <5crore | >10 lack and <2crore |
| Medium enterprises | >5 crore | >=2crore |

### CLASSIFICATION OF FARMERS BASED ON LAND

| FARMER | TYPE |
|---|---|
| Marginal farmer | <=1hectare |
| Small farmer | > 1hectare |

### CLASSIFICATION OF BANKS BASED ON SIZE

#### Payment bank

Bank which is designed by RBI for carrying out the payment related activity by individuals.

#### Small finance banks

We cannot expect the banks to open the branch at all places in the country. So, RBI initiated the small finance banks, which can be easily opened with low investment.

##### Properties

1. Only who have knowledge of banking in 10 years should start a small finance bank.

| BANK TYPE | ATM OR DEBIT CARD | CREDIT CARD | LOAN | ONLINE BANKING | MOBILE BANKING |
|---|---|---|---|---|---|
| Payment bank | YES | NO | NO | YES | YES |
| Small finance bank | YES | YES | YES | YES | YES |
| | | | | | |

| BANK TYPE | DEPOSIT LIMIT | MINIMUM CAPITAL REQUIRED | REAL TIME EXAMPLE | ELIGIBLE PROMOTERS | Demand deposit |
|---|---|---|---|---|---|
| Payment bank | 1 lack and can be raised by performance of bank | | Airtel payment bank | NBFC, BC, Mobile telephone companies, Supermarket chains, | Yes |
| Small finance bank | | 100 Crore | | | |
| | | | | | |

## CLASSIFICATION OF ACCOUNTS IN BANK

| SAVINGS ACCOUNT | CURRENT ACCOUNT | FIXED DEPOSIT ACCOUNT | RECURRING DEPOSIT ACCOUNT |
|---|---|---|---|
| As name suggests, it is used for saving the small amount of money. | Saving huge amount of money | Depositing fixed amount of money without taking back for specific period for getting interest. | Depositing amount at regular intervals of money without taking back for specific period for getting interest. |
| Here transactions and deposit is limited to certain amount. | No limit on transaction and deposits | | |

| | | | |
|---|---|---|---|
| Usually used by people who is getting salary | Usually used by people who is giving salary | | |
| Account can be opened with name of individual | Account can be opened with name of individual or business name. | | |
| Service charge is less | Service charge is high | No service charge | No service charge |
| Overdraft facility(Negative balance) is not available | Overdraft facility(Negative balance) is available | Overdraft facility(Negative balance) is not available | Overdraft facility(Negative balance) is not available |
| No limit on validity period | No limit on validity period | Maximum ten years | Maximum ten years |

## RESERVE BANK OF INDIA

### Roles and Responsibility:

1. Issues license to the banks
2. Responsible for printing currency notes
3. Fixes the interest rates of loan, deposit
4. Responsible maintaining external and internal value of currency
5. Controls foreign exchange transactions.
6. It will not accept deposit from public

## TYPES OF ACCOUNTS IN BANK

## HEADQUATERS

| BANK NAME | HEADQUATERS |
|---|---|
| Reserve bank of india | Mumbai,Maharastra |
| NPCI | Pune, Maharastra |
| | |

## BANK ACCOUNT

### TERMINOLOGY

### INOPERATIVE ACCOUNT

1. Account in which there is no transaction for long period is called inoperative account.
2. Account which is inoperative for 10 years is marked as inoperative account.
3. This can be savings account or deposit account

## DEPOSIT ACCOUNTS

### Senior citizen deposit

Deposit made by the people who has minimum age of 60 years, but 55 years is allowed in case of voluntary retirement.

### Demand liabilities

Amount has to be paid to the customers on demand.

Example: Demand draft. Once we gave the demand draft they have to pay the money

### Time liabilities

Amount has to be paid to customers after certain period.

Example: Fixed deposit.

### Unclaimed deposits

Deposit and interest for the deposit amount which is uncollected after the maturity period is called as unclaimed deposit.

## MONETARY & MONETARY RELATED

### TERMINOLOGY

| TERM | DESC |
|---|---|
| Call Money | Borrowed for one day |
| Notice money | Borrowed for 2 days to 14 days |

| | |
|---|---|
| Term money | Borrowed for more than 14 days |
| | |

## TRADING

### TERMINOLOGY

#### MONEY MARKET

Money market is place for performing trading, where financial instrument with high liquidity and less maturity are bought and sold. It mainly used for trading within overnight to less than 1 year.

Financial instrument can be

1. Certificate of deposit (Depositing some money in bank)
2. Commercial paper( Unsecured promissory note given by big corporate for their short term loans)
3. IBP-Inter bank participation certificate
4. Treasury bills
5. Call/Notice money
    a. Amount should be repaid upon demand
    b. Investment period varies from 2 days to 14 days.
    c. Non bank institutions should not involve in call back or notice money
    d. Interest rate is decided by FIMMDAI

#### Government instruments:

To meet the temporary expenditure, government will issue instruments like treasury bills, cash management bills to raise a fund.

#### Interbank  participation certificate(IBPC):

When bank runs in short of cash, then bank can ask loan with other banks. This is called as interbank participation certificate. There are two types of interbank participation certificate

1. Risk sharing IBPC

2. Non-Risk sharing IBPC

   If it is risk sharing, then lending bank has to ensure the health code status, conduct of account of borrower bank.

### Treasury bills

1. This is short term borrowing of money from  public by the central government of India.
2. Investors: Anyone other than the state government
3. Denomination: Minimum 1 Lack and multiples of 1 Lack
4. Maturity: 91,182,364 days
5. Rate of interest: Zero coupon security.

### CAPITAL MARKET

Capital  market is place for performing trading, where financial instrument with high maturity(Long term investments) are bought and sold. It mainly used for trading within overnight to less than 1 year.

## LOANS & INTEREST

## TERMINOLOGY

### BASE RATE

1. Bank should charge at least minimum rate from its customers for giving a loan. This minimum rate is called base rate.
2. This base rate is set by RBI
3. This is only base rate and bank can give loan for rates more than this base rate.
4. When RBI changes the base rate then for all existing loan it will get affected.

5. Exception: Bank can give loan less than the base rate for the following purposes

    a. Loan to own bank employee

    b. Loan against deposit

    c. DIR Loan

## MCLR(Marginal cost lending rate)

Banks has to prepare the lending rate based on the following factors,

1. Operating cost – Electricity, Employee salary
2. Tentor premium- Higher interest for long term tenture
3. Negative carry on CRR(Cash reserve ratio)
4. Marginal cost of funds

This lending rate is called Marginal cost lending rate.

Base rate will vary depends upon the marginal cost lending rate.

### Exemption of MCLR:

1. Loans under schemes of government of India or government undertaking
2. Fixed rate loans

## DIR Loan(Differential rate loan)

Loan for low income group

## TYPES OF INTEREST

### REPO RATE

Interest for banks which takes loan from RBI

### REVERSE REPO RATE

Interest for RBI for the amount of banks which is held by RBI.

### BANK RATE

Interest charged for loans or credit issued by the commercial banks to customers.

## PSL(PRIORITY SECTOR LENDING)

Providing loan to needy sector of society is called PSL or priority sector lending.

Sectors can be

1. Agriculture
2. Education
3. Export
4. Housing
5. Renewable Energy
6. Micro and small enterprises

### Adjusted Net bank credit(ANBC)

Total amount of bank credit available minus the SLA held to maturity and other factors. Based on this Total lending amount to customers will be decided.

### Targets of PSL

Domestic bank must give 40% of the ANBC to PSL and foreign bank has to give 32% of ANBC to PSL.

| SECTOR | DOMESTIC BANK PERCENTAGE | FORIEGN BANK PERCENTAGE |
|---|---|---|
| Eligibility | Domestic bank should achieve this target when they have 20 branches or more. | Foreign banks with less than 20 branches |
| AGRICULTURE | 18 | No target |
| Weaker section | 10 | No target |
| Export | No target | 10 |
| | | |

### Phased manner achievement

Banks has to achieve the target in phased manner.

| YEAR | AGRICULTURE | MICRO INDUSTRIES |
|---|---|---|
| By Mar 2016 | 7 | 7 |
| By Mar 2017 | 8 | 7.5 |
| | | |

## Limit of loans in PLS

| SECTOR | BUSINESS PEOPLE | INDIVIDUAL |
|---|---|---|
| Renewable energy sector | 15 Crore | 10 Lack |
| Housing | NA | 28 Lack for Metropolitan city and 20 lack for other city |
| Education | NA | 10 lack for domestic and 20 lack for abroad |
| Social infrastructure activity | 5 Crore per borrower for activity like building schools. | |

## Outstanding target

| SECTOR | TARGET |
|---|---|
| Agriculture | 18% |
| Small and marginal farmers | 8% |
| Micro enterprises | 7.5% |
| Weaker sector | 15% |
| | |

## RBI Revised PSL norms to RRB

1. Medium enterprise, Social infrastructure, Renewal energy will be part of priority sector lending
2. PSL will be monitored on quarterly and annual basis
3. 18% of total outstanding should be advanced to activities mentioned under agriculture
4.

## Miscellaneous

1. Any scheduled bank having shortage of money will be allocated with RIDF (Rural infrastructure development) established with NABARD.

# MISCELLANEOUS

## TAXATION ON INTEREST

1. Tax will be deducted for the interest gained via fixed deposit or term deposit

2. Tax will not be levied for interest earned on savings bank account until 10,000

### Reverse mortgage loan

1. This loan is provided for senior citizens by taking their home.
2. Value of the loan will vary depends upon the home.
3. It is not compulsory to repay the loan at regular intervals. After loan period or borrowers death, if loan is not repaid, then bank will acquire the home
4. This loan can be used for medical emergency, leading the life after 60 years etc.,
5. Loan amount can be taken as lump sum or periodic payments
6. Property tax has to be paid by borrower, bank will not pay it.
7. Maximum loan period is 15 years
8. Life of residual property should be at least 20 years. When borrower lives more than 20 years at same place, then periodic payment is not eligible.

## TERMINOLOGY

### Point of sale

Point of sale is the machine which you swipe the card after your purchase.

Example: Machine used in Chennai silks by cashier.

### EFTPOS- Electronic fund transfer at point of sale

Method of paying the payment through point of sale is called as EFTPOS.

Example: Giving the card to cashier in the Chennai silks.

### SLA RATE-STATUATORY LIQUIDITY RATIO

Amount which needs to be maintained by the bank in form of cash, gold or government approved form.

### CRR-CASH RESERVE RATIO

Minimum fraction of total deposit made by the customers in form gold,cash or government approved bonds with the RBI.

## MISCELLEANEOUS

### BASEL NORMS

Basel is one of the city in Switzerland and BASEL norm first got its name because this norm is formed first in the Basel city. BASEL norms are international banking regulatory standard for avoiding financial crisis. It is formed by central bank governors of G-10 countries.

#### How Basel norms are formed?

In Germany one bank went in to liquidation, because of it cannot pay the fund to its customers. So, germany regulators forced that bank to liquidation. To avoid this kind of liquidation, Basel norms are formed.

#### BCBS

BCBS stands for Basel committee for banking supervision.

##### Major roles

1. Standardize banking practice all over the world
2. Calculating minimum capital requirements of the bank

#### Basel I Norms – Minimum capital requirements

Assets of the bank is classified in to five groups according to credit risk,

| Allowed involvement | Risk weight rating |
|---|---|
| Cash, bullion, Home debt like treasury | 10 |
| Mortage backed security | 20 |
| Municipal revenue bonds, residential security | 50 |
| Corporate debit | 100 |
| | 0 |

Bank with international presence, should maintain 8% of risk weighted assests.

#### Basel II Norms

##### Three pillars

1. Refinement of minimum capital requirements
2. Supervisory review process for risk management

3. Risk disclosure and market discipline

## Basel III Norms – Disclosure and market discipline

### Three pillars

1. Enhancement of minimum capital requirements
2. Enhanced Supervisory review process for risk management and capital planning
3. Enhanced risk disclosure and market discipline

## Snapshot of Minimum capital requirement risk of all basel norms

| BASEL | CREDIT RISK | MARKET RISK | OPERATIONAL RISK | Liquidity Risk |
|---|---|---|---|---|
| BASE I | X | | | |
| BASE II | X | X | | |
| BASE III | X | X | X | |

## SECTORS IN INDIAN ECONOMY

1. Primary sector
   a. One who produces the goods
   b. Example: Cultivating the sugarcane
2. Secondary sector
   a. One who transfers the goods in to useful one
   b. Example: Changing sugarcane into chocolates.
3. Tertiary sector:
   a. One who takes the useful products to market.
   b. Example: selling the chocolates.

# OPERATING SYSTEM

## TERMINOLOGY

### Kernel space & User space

Main memory is divided in to two types such as,

1. Use space
2. Kernel space

Kernel code is loaded in to the kernel space of the main memory when operating system is boot up. All kernel activity will happen in kernel space. This is to avoid crash to the operating system when user process misbehaves. User application will run in use space of the main memory

### Difference between kernel and operating system:

Operating system is the complete package which includes the GUI, basic application software, utility tools like device defragmenter apart from kernel, whereas kernel is core part of operating system which handles the system calls, memory management, process management etc.,

### System call

System call is an interface provided by the kernel for performing some job.

When user process makes the call to the kernel, then it is called as system call.

## KERNEL

### Roles of kernel

1. Kernel mediates the access to the system resources.
2. Handles the system call
3. Process management
4. Memory management
5. Inter process communication management
6. Virtual memory management
7. Device driver management

## MULTITHREADING-OS

The thread is the process of executing the set of instructions independent from each other by sharing the resources.

Then process is also a set of instructions, which produce certain output.

What makes the difference between the thread and process? Here are few…

| Thread | Process |
|---|---|
| **Thread share the same address space** | Process will have different address space |
| **Context switching is slow here.** | Context switching is faster between process |
| **Thread depend upon the process.** | Process is typically independent |

In above difference context switching is something storing and retrieving the states of the threads.

### Multithreading:

Multithreading is the parallel execution of the programs by sharing the resources like memory.

## MATHEMATICS

### NUMERIC MATHEMATICS

#### Number types

| Number | Description |
|---|---|
| 10-6 | Micro |
| 10-9 | Nano |

### MISCELLANEOUS

#### TERMINOLOGY

#### ORDERED PAIR:

When reverse of given pair is not true, then it is called as ordered pair

```
(a,b) not equal to (b,a)
```

#### UNORDERED PAIR:

When reverse of given pair is true, then it is called as unordered pair

```
(a,b) equal to (b,a)
```

## SET

Set is an collection of distinct objects

```
{1,25,45}
```

## Reason for pi constant:

Value of pi=3.14

Wondering how 3.14 is computed?

When you make the 22 feet rod line in to circle it will become the 7 feet height circle. So when height of the circle has to be 1 feet then we need 3.14 feet straight rod

## SYMBOLS

| SYMBOL | DESC |
|---|---|
| ** | Power of |
| | |
| | |

## IPC – INTER PROCESS COMMUNICATION

### PIPE

Pipe is used to transfer data from one process to other process as stream. Pipe will transfer the unstructured data from one end to other end.

### SHARED MEMORY

Shared memory is the kind of memory which is shared by the several process with intention to share the information.

It is one of the interprocess communication technique.

One process will create a space in RAM whereas the other process can access the memory using this technique.

### Disadvantages:

1. Shared memory can be accessed only with the machine. It cannot be accessed outside the machine
2. When Operating system crashes, then content of shared memory will get lost.
3. Size of the shared memory is limited to Ram and virtual memory size
4. Size of the memory has to be given in advance
5. We have to manage the memory which is allocated internally for process from the given raw memory.

## MESSAGE QUEUE

This is the one of the inter process communication technique where information can be exchanged from one process to another process in the flavor of queue(FIFO).

### Creating a message queue:

```
msgget(ket_t key,int msg_flg)
```

### COMMANDS USED FOR MESSAGE QUEUE:

ipcrm – removes the message queue

ipcs – Lists all the message queue

## REMOTE PROCEDURE CALL

Remote procedure call is nothing but giving a request to run the program located at the remote machine.

## MISCELLANEOUS

### Difference between message queue and pipe

| MESSAGE QUEUE | PIPE |
|---|---|
| Transfers structured data | Transfers unstructured data |
| Sender and receiver no need to maintain the EOF | Sender and receiver has to maintain the EOF |
| Information can be assigned with priority so that high priority information will be transferred first. | No priority concept |

## MESSAGE QUEUE

### MSGGET:

This will create the message queue with the key provided.

Msgget is the system call

```
msgget(key_t key ,int msgflag)
```

| PARAMETER | DESCRIPTION |
|---|---|
| key | It is the system wise unique identifier that was used to identify the queue.<br>If any body in the system wants to connect to same queue then then can use that identifier<br>Hardcoding a key will cause a problem because there is probability that same key will be used by anyone in the system. |
| msgflag | |
| return value | Returns the message queue id on success and returns -1 on failures.<br>errno will be set upon failures. |

### UTILITY FUNCTIONS:

#### ftok()

ftok was used to generate the key for IPC purposes. This is an id used to identify the ipcs uniquely in the system similar to the file descriptor used to point a file.

But file descriptor is accessed only in the particular process but the ftok will be accessed all over the system.

```
key_t ftok(char *path,int id);
```

| PARAMETER | DESCRIPTION |
|---|---|
| char *path | File path<br><br>This file should have the sufficient permission for the ipcs.<br><br>File should exists when calling this function, if not call will fail |
| char id | To create a bunch of Keys using the single file path |
| return value | return a newly generated key for the path and id specified.<br><br>The ftok will return the same key if path and id was passed as same.<br><br>It is unspecified that key will be same when path is deleted and created with same name.<br><br>On error it will return -1 and corresponding errno will be set. |

## USEFULL COMMANDS:

ipcs- List all the ipc available in the system

ipcrm – Remove the ipc from the system and deallocates the resources acquired by the IPC.

## TRASH CAN

| Color | meaning |
|---|---|
| **Green** | Content is ready – Post to blogger |
| **Yellow** | Under development |

| | |
|---|---|
| **OCEAN BLUE** | Modified the content which is already posted to blogger |