1. **NESS TECHNOLOGY**

You may be familiar with the tail command on Linux. Similar to tail, implement the **lastnumlines()** function to return the last **num** lines (2nd parameter, e.g. 10 lines) in the file pointed to by **inputfp** (1st parameter):

char \*\* lastnumlines(FILE \* inputfp, unsigned int num)

{

char \*\* tailbuf = calloc(num, sizeof(char \*));

...

return tailbuf;

}

Assume that inputfp is not NULL.

**inputfp** could be referring to a Unix/Linux pipe, which means fseek() **cannot** be used.  **Also, lines cannot be read twice from inputfp**.

Assume that contents of **inputfp** cannot be read into memory all at once.

Assume maximum length of a single line is 255 characters. For the **num** parameter assume: 0 < num < 100.

Allocation of tailbuf is done in the snippet above, assume it will not fail. Assume that further heap allocations will not fail. Assume that caller of lastnumlines() will free the memory of the tailbuf variable. Ensure the any extra heap memory allocated in lastnumlines() is freed before return.

Use C standard library functions as required, add #include statements for those. Implement this function as efficiently as possible: in terms of CPU & memory utilisation.

See below a possible way to use lastnumlines().  Note that the use will change based on how lastnumlines() is implemented.  You can provide sample code on how your implementation of lastnumlines() could be used.

int main()

{

FILE \* fp = fopen("input.txt", "r");

// Assume fp is not NULL

char \*\* lastlines = lastnumlines(fp, 10);

for (int i = 0; i < 10; i++) {

if (lastlines[i] != NULL) {

printf("%s", lastlines[i]);

}

}

// Code to free lastlines will come here

}

1. **Siemens**

Datatypes

Diff 32 bit compiler and 64 bit compiler

Linux Camand like

Df -h

Kill -9 PID

Ps -ef

Which is correct syntax

* Cons int \*ptr;
* Int const \* ptr;
* Int \* cons ptr;

ptr++;

sockets

IPC

Thread

Number factorial code

Sprintf() function in c

1. **TechM**

Reverse string code

What is Typedef

Inline functions

Explain volatile keyword.

Structure

We can change header file or not? if yes how we can change with standard provision.

Reverse string code.

1. **Oracle**

Write a query for print second highest number.

Correct Sequence of SQL keywords as follow

* where
* groupof
* having
* orderby

how many constrains in SQL

write a query to remove duplicates from table.

1. **Vodaphone**

Shell scripting for loop syntax

Df -h

Grep -a

Zgrep

Telcom billing project

Production issue

Crontab

1. **Capegemini**

Syntax for

* constant Pointer to integer

Int \*const ptr

* Pointer to constant integer

Const int \*ptr

OR

Int const \* ptr

Find linklist is round as follow

1|1000|🡪2|2000|🡪3|3000|🡪4|5000|

Addr 5000🡨 <------------------------|



Write code.

* Guess

printf(“len=%d\t sizeof=%d”,strlen(“string”),sizeof (“string”)); the output

Struct val

{

Int a:3;

Int b:4:

Int c:4

};

Void main()

{

struct Val obj;

obj.a=2;obj.b=3;obj.c=5

print(“%d%d%d”,obj.a,obj.b,obj.c);

}

* Guess the output

Void main()

{

Int a=2

Switch(a)

{

Case:1

Printf(“MY\n”);

Case:2

Printf(“NAME\n”);

Case:1

Printf(“is\n”);

Default:

Printf(“Mahesh\n”);

}

}

process segments and memory for int a=0 allocate in wich section.

guess output:

printf(“len=%d\t sizeof=%d”,strlen(“string”),sizeof (“string”));

Aray to pointer conversion.

when we initialize pointer in c compiler allocate the memory or not?

→ As per my knowledge compiler only allocate that memory block which only handle addresses not data.

ex.

int \*ptr = 0;--we cannot store value in pointer

int a = 10;

ptr = &a;

cout << \*ptr << endl;

* guess output

#include<stdio.h>

void main(void)

{

struct student

{

int a;

}var={1};

struct student first;

printf("a=%d\n",first.a);

}

→Garbage value

1. **Fujitsu**

* structure memory allocation and syntax.
* padding
* Union syntax and memory allocation
* Delete node from Doubly link-list without head
* Delete node from Singly link-list without head
* constant pointer to character syntax
* Volatile for turn off optimization on variable.
* Dangling pointer
* Malloc
* strcmp() write a code
* null pointer

1. **netscout**

* Roles and responsibility of the current project.
* memory leakage.(valgrind)
* what is TCP
* What is UDP
* Diff TCP and UDP
* tcpdump command
* how to know unknown IP
* OPEN SSH
* System admin functionality
* Unix socket VS socket
* FTP

1. **Calsoft**

* convert int to binary
* STL
* polymorphism
* Type of polymorphism
* find the middle node in the link list in a single iteration.
* Threads and synchronization
* IPC

1. **Quest Engineering**

* Virtual function
* Design Patterns
* Inheritance
* Run time polymorphism

1. **Dasult system**

* input Arr={ -1, 0, 3, -3, 2, 0, 5, 4, 0, -4} Out= {0, 0, 0, -1, 3, -3, 2, 5, 4, -4}
* input aabcdccb output "a2b1c1d1c2b1"
* Design pattern
* singleton pattern

1. **Mouse Electronics**

* Alternative way for #pragma pack(1) for memory padding
* Fetch 3 rd bite of structure int using Union
* Why we use Function pointer and how to use
* Ways for sharing variable from one file to another file and static variable can share in multiple file or not
* Life for static variable
* What is dangling pointer
* How we can do unit testing
* Agile or waterfall model follow in your project?
* difference between

char arr[10]=”mahesh”->static memory allocation

char arr[]=”mahesh” →Dynamic memory allocation

* memory segments for process
* In stack section wich member get memory?
* Where global variable gets memory?
* where static members gets memory
* How to know memory leakage in c
* How to write robust code in c

1. Parelell wireless

* Write a program for reverse string without local variable recursion.
* Write program for synchronize two threads to read alternate index of array
* Memory layout for process
* static global variable and global variable difference in C program
* Without #pagma pack(1) avoid structure padding
* singly linked list has only one node address to delete node

1 -> 2 -> 3 -> 4 -> 5-> NULL

ptr

temp=ptr->next;

n\_next=temp->next;

ptr->val=temp->val;

1 -> 2 -> 4 -> 4 -> 5-> NULL

ptr temp n\_next

ptr->next=n\_next;

free(temp);

1 -> 2 -> 4 -> 5-> NULL

ptr

* int arr[]={3,4,5,6,7,0,1,2},element=0 binary search for circular array
* TCP socket
* Thread states
* Write SQL query to fetch max ID 4 recourd

select \* from emp where emp\_id>=(select (MAX(emp\_id) -4 ) from emp)

* Print 2D array with pointer

1. GS Lab

**Process frame**

cammand line

heap

stack

data

unintillize

text

inline void fun3()

{

...

}

class complex

{

int \*ptr=nullptr,num;

public:

complex(int numMemory)

{

ptr=new int[numMemory];

num=numMemory;

}

complex(complex &obj)

{

this->ptr=new int[obj.num];

this->num=obj.num;

}

void fun()

{

cou<<"from perant\n";

}

virtul void fun1()

{

cou<<"from perant virtual fun\n";

}

};

class complex2: public complex

{

void fun1()

{

cou<<"from perant virtual fun\n";

}

};

struct Node

{

int val;

struct Node\* next;

};

mutex lk;

mutex\_lock(lk);

mutex\_unlock(lk);

int main(void)

{

complex2 obj;

complex obj1=obj;//obj1(obj;

obj.fun1();

vectore<int> vec={1,2,3,4,5,6,7};

for( auto i=vec.begin();i!=vec.end();i++ )

{

cou<<\*i<<endl;

}

complex \*ptr=new complex2;

ptr->fun1();

free(ptr);

ptr=nullptr;

return EXITSUCCESS;

}

class Singlton

{

private:

Singlton()

{

}

public:

static Singlton \*ptr=nullptr;

static Singlton\* getSingletonOBJ(void);

};

static Singlton Singlton::\*ptr=nullptr;

static Singlton\* getSingletonOBJ(void)

{

if(ptr)

return ptr;

else{

ptr=new Singlton;

return ptr;

}

}

insert into table\_name values(1,'Mahesh','Barate');

vec : 34,4,12,5,2,3

sum=9

pair<int,int> p;

for(int i=0;i<size-1;i++)

{

for(int j=i+1;j<size;j++)

{

if(vec[i]+vec[j]==sum)

p={i,j};

}

}

if(p.empty())

return false;

return true;

grep -rnI "pattern"

2nd Round:

#include <iostream>

#include <thread>

using namespace std;

class Singleton

{

protected:

Singleton(const std::string value): value\_(value)

{

}

static Singleton\* singleton\_;

std::string value\_;

public:

Singleton(Singleton &other) = delete;

void operator=(const Singleton &) = delete;

static Singleton \*GetInstance(const std::string& value);

void SomeBusinessLogic()

{

// ...

}

std::string value() const{

return value\_;

}

}

//static Singleton\* Singleton::singleton\_= nullptr;

Singleton \*Singleton::GetInstance(const std::string& value)

{

//if(singleton\_==nullptr){

//singleton\_ = new Singleton(value);

//}

static Singleton instance;

return instance;;

}

void ThreadFoo(){

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

Singleton\* singleton = Singleton::GetInstance("FOO");

std::cout << singleton->value() << "\n";

}

void ThreadBar(){

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

Singleton\* singleton = Singleton::GetInstance("BAR");

std::cout << singleton->value() << "\n";

}

int main()

{

std::thread t1(ThreadFoo);

std::thread t2(ThreadBar);

t1.join();

t2.join();

return 0;

}

* coding question

void TestFunc(const int& t)

{

t = 10;

cons\_cast<int>ptr=t;

\*ptr=15;

}

int main()

{

int val;

TestFunc(val);

}

* coding question

Declare a integer

pointer in main function and allocate memory using malloc/new to it in

some other function defination

bool fun(int \*&ptr)

{

ptr=(int\*)malloc(sizeof(int));

if(ptr)

{

return true;

}else

{

perror("malloc/new is failles\d");

exit(1);

}

}

int main()

{

int \*ptr=nullptr;

bool ret=fun(ptr);

if(ptr==nullptr)

{

cout<<"memory is not allocated\n";

}

else

cout<<"memory is allocated\n";

if(ret)

{

logic....

}

free(ptr);

ptr=nullptr;

return 0;

}

* coding question

1 -> 2 -> 3 -> 4 -> 5

add

1 -> 3 -> 3 -> 4 -> 5

add mid next

1 -> 3 -> 4 -> 5

add next

C++ repo

<https://github.com/DopplerHQ/awesome-interview-questions#c>

* access spicifier Queuen in depth

//Pass access specifier using virtual table

#include <bits/stdc++.h>

using namespace std;

class Base

{

public:

virtual void print()

{

cout << "print from base\n";

}

};

class Derive : public Base

{

private:

void print()

{

cout << "print from Derive\n";

}

};

int main()

{

Base \*bptr = new Derive();

bptr->print();

return 0;

}

#include <bits/stdc++.h>

using namespace std;

class demo

{

private:

~demo();

};

int main()

{

demo \*ptr = new demo;//only create object dynamically

//delete ptr; it will throw error in compile time cause

// we use private destructore

return 0;

}

reference implemented using pointer

#include <bits/stdc++.h>

using namespace std;

void memAlloc(int \*\*ptr)

{

\*ptr = new int(2);

}

int main()

{

int \*ptr = nullptr;

memAlloc(&ptr);

cout << "\*ptr: " << \*ptr << endl;

return 0;

}

Reverse a word “Hello World” .. first reverse Hello then world..

IN a given array find a number of combinations where the last digit of the first number should be equal to the first digit of the last number.

find duplicate letters in given string

check two strings are anagram or not

how to synchronize the multithreading

what is thread pool how do you implement it

difference between thread and process

diamond problem in c++

reverse a string using vector or linked list

linux signals

how linux bootup

Vector

List

Map

Diff between map and multimap

Set specific bit of given integer

Socket programming

System calls used for socket programming

Git commands

Linux commands.

Shell script

Object oriented concepts (detailed with code, example)

Constructors(copy constructor and it’s behaviours specifically)

Destructor

TCP/UDP(GS Lab project is in networking)

Friend functions/classes

Design Patterns ( Factory and Singleton Specifically)

Smart Pointers

SQL queries (Joins wagaire)

Memory layout

Stack overflow issue

Program flow till execution (purn toolchain)

Recursion and it’s working

Class baddal veglya veglya goshti, diamond issue..

Virtual baddal khup kahi..

Pure virtual...

Static functions,members

Delegate constructor

Thread

Synchronization

Auto

Initialisation list

Std::asynchrony

Move symantics

Rule of three

Casting

Lambda expression

Explicit virtual override function

Final specifier

Std::thread

Smart pointer

Types of Mutex

1. Normal Mutex - access by multiple thread and not use in recursion.
2. Recursive mutex - Access by single thread and use in function recursion.

Vtable for virtual function. <https://www.youtube.com/watch?v=hS7kPtVB1vI>