**C Theory:**

1.

int main()

{

char str1[10],str2[10];

int ind1,ind2=0,n;

puts(“enter str1:”);

gets(str1);

printf(“enter number of byte to copy:”);

scanf(“%d”,&n);

for(ind1=0;ind1<n;ind1++)

str2[ind2++]=str1[ind1];

str2[ind1]=’\0’;

puts(str2);

}

memcpy:

void \*memcpy(void \*dest,const void \*src,size\_t n)

it copies n bytes from source to destination,memory area shouldn’t overlap.

Memmove:

void \*memmove(void \*dest,const void \*src,size\_t n)

it copies n bytes from source to destination,memory area may overlap.

Memset:

void \*memset(void \*s,int c,size\_t n)

it will fill the first n byte of s with c

2.

whenever an interrupt occured in the middle of the execution of the program,it goes to isr and invokes the main program after the execution of isr is called as re-entrant function

void isr()

{

printf(“alarm raised\n”);

}

int main()

{

alarm(2);

signal(SIG\_ALRM,isr);

while(1);

}

output:

after 2 seconds it will print “**alarm raised**”

3.

void pointer :

it is the pointer which is declared with void \*,using this pointer we can point to any datatype,but we need to typecast it.

Char ch;

int n;

Void \*ptr;

ptr=&ch;

ptr=&n;

4.

Linear search:for this algorithm the array need not to be is sorted order,it will search the value from starting index to last index.

Binary Search:for this algorithm the array should be in sorted order.

->in this algorithm first we find mid value

->we will aways compare the value with the mid value.if that value is less than mid value we will search in the array which is left to the mid value,else searches in right array

5.

typedef struct dll{

int data;

struct dll \*prev;

struct dll \*next;

}node;

node \*delete(node \*head,int del)

{

node \*temp;

temp=head;

while(temp)

{

if(temp->data==del)

{

temp->next->prev=temp->prev

temp->prev->next=temp->next;

free(temp);

temp=NULL;

return head;

}

temp=temp->next;

}

puts(“data not found”);

return NULL;

}

**Linux Kernel:**

1.Booting sequence:

A:

poweron->romcode->xloader->u-boot->kernel->init

romcode:

it will initialises the hardware.

X-loader:

initialisea the dram and loads the u-boot image.

U-boot:

bsp initialisation and loads the kernel image.

Kernel:

initialises the device drivers and file system.

Init:

it is the first process runs in the kernel

6.

data segment:

all local variables will be stored.

Bss:

all global variables will be stored.

Head:

used for dynamic memory allocation.

7.

shared memory:

this IPC is used to share the memory between the different processes.i.e you can use the same memory which is already used by other process.

We have Pipes,fifo,mesg queues to communicate between different processe.

Pipe:

it is used communicates between related processes.

Fifo:

used to communicates between unrelated processes.

Msgqueue:

using ids it will communicate with other processes.

8.

**Mutex:**

**->**owner has to unlock the critical region

->dead lock,priority inversion problems will occure

binary semaphore:

->any one can unlock the critical region

->dead lock issue is solved here

9.

Priority inversion:

let lower priority process is executing,after some time highest priority process has arrived,the highest priority process has to wait untill the lowest priority process completes is called as priority inversion.

10.

multithreading:

means sharing the task i.e sharing the process,process will have multi threads

executing no.of threads under one process

multitasking :

means different task i.e different processes

executing multiple processes

12.

COPY\_TO\_USER:copies from kernel space to userspace.

COPY\_FROM\_USER:copies from userspace to kernel space.

**C-Coding:**

1.

#include<stdio.h>

#include<stdlib.h>

typedef struct stack{

int data;

struct stack \*next;

}NODE;

void add(NODE \*\*head,int num)

{

int v=0,data;

NODE \*nu,\*temp;

while(v<num)

{

nu=malloc(sizeof(NODE));

printf("enter data:");

scanf("%d",&nu->data);

nu->next=NULL;

if(\*head==NULL)

\*head=nu;

else

{

temp=\*head;

while(temp->next)

temp=temp->next;

temp->next=nu;

}

v++;

}

}

void print\_mid(NODE \*head)

{

NODE \*fast,\*slow;

fast=slow=head;

while((fast->next!=NULL)&&(fast->next->next!=NULL))

{

fast=fast->next->next;

slow=slow->next;

}

if(fast->next==NULL)

printf("mid node value is:%d\n",slow->data);

else

printf("mid values are:%d, %d\n",slow->data,slow->next->data);

}

int main()

{

NODE \*head=NULL;

int num;

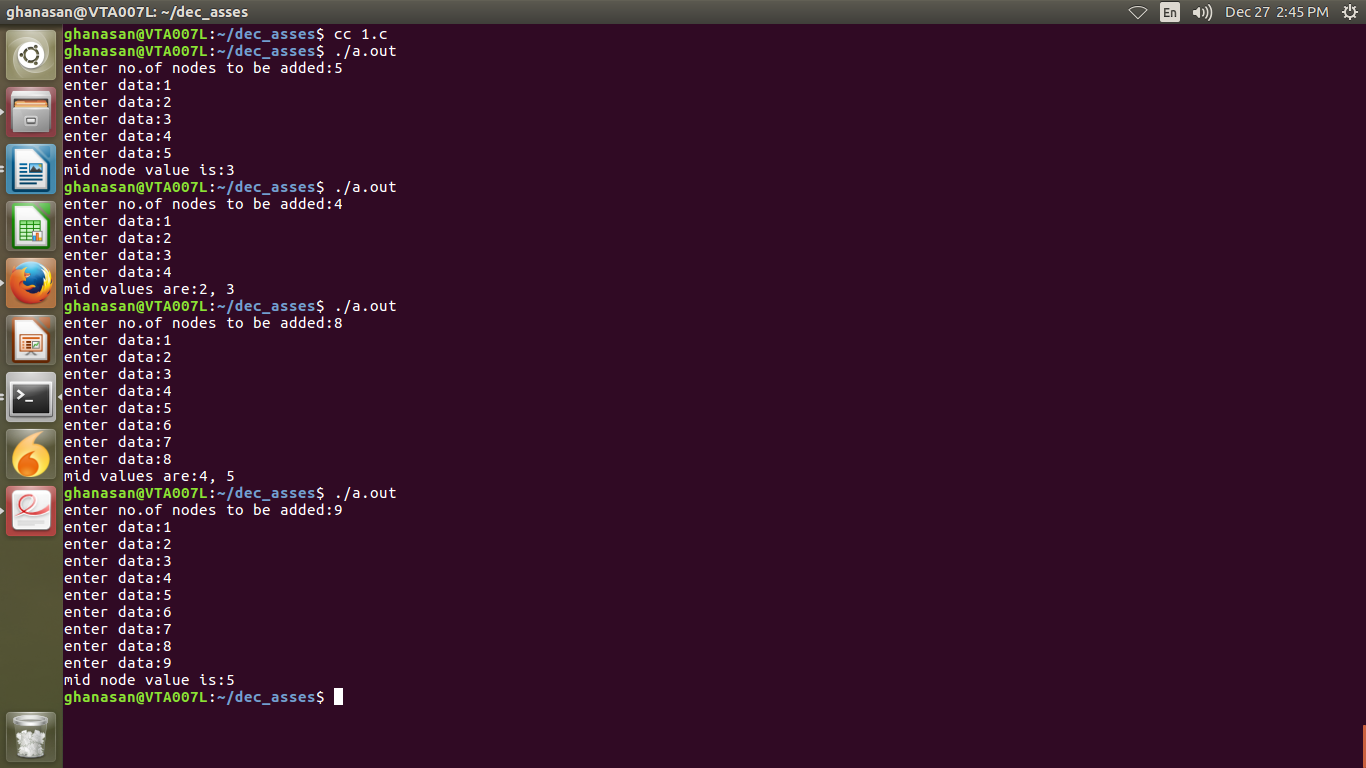
printf("enter no.of nodes to be added:");

scanf("%d",&num);

add(&head,num);

print\_mid(head);

}

****

2.

#include<stdio.h>

#include<stdlib.h>

struct bst

{

struct bst \*left;

int data;

struct bst \*right;

};

void insert(struct bst \*\*node,int data)

{

struct bst \*nu;

if(\*node==NULL)

{

nu=malloc(sizeof(struct bst));

nu->data=data;

nu->left=NULL;

nu->right=NULL;

\*node=nu;

}

else

{

if(data<(\*node)->data)

insert(&(\*node)->left,data);

else if(data>(\*node)->data)

insert(&(\*node)->right,data);

}

}

void print\_leaf(struct bst \*root)

{

if(root==NULL)

return;

if((root->left==NULL)&&(root->right==NULL))

{

printf("%d ",root->data);

return;

}

print\_leaf(root->left);

print\_leaf(root->right);

}

int main()

{

struct bst \*root=NULL;

int n,i=0,data;

printf("enter no.of elements:");

scanf("%d",&n);

printf("please enter values is the form of BST tree only\n");

while(i<n)

{

printf("enter data:");

scanf("%d",&data);

insert(&root,data);

i++;

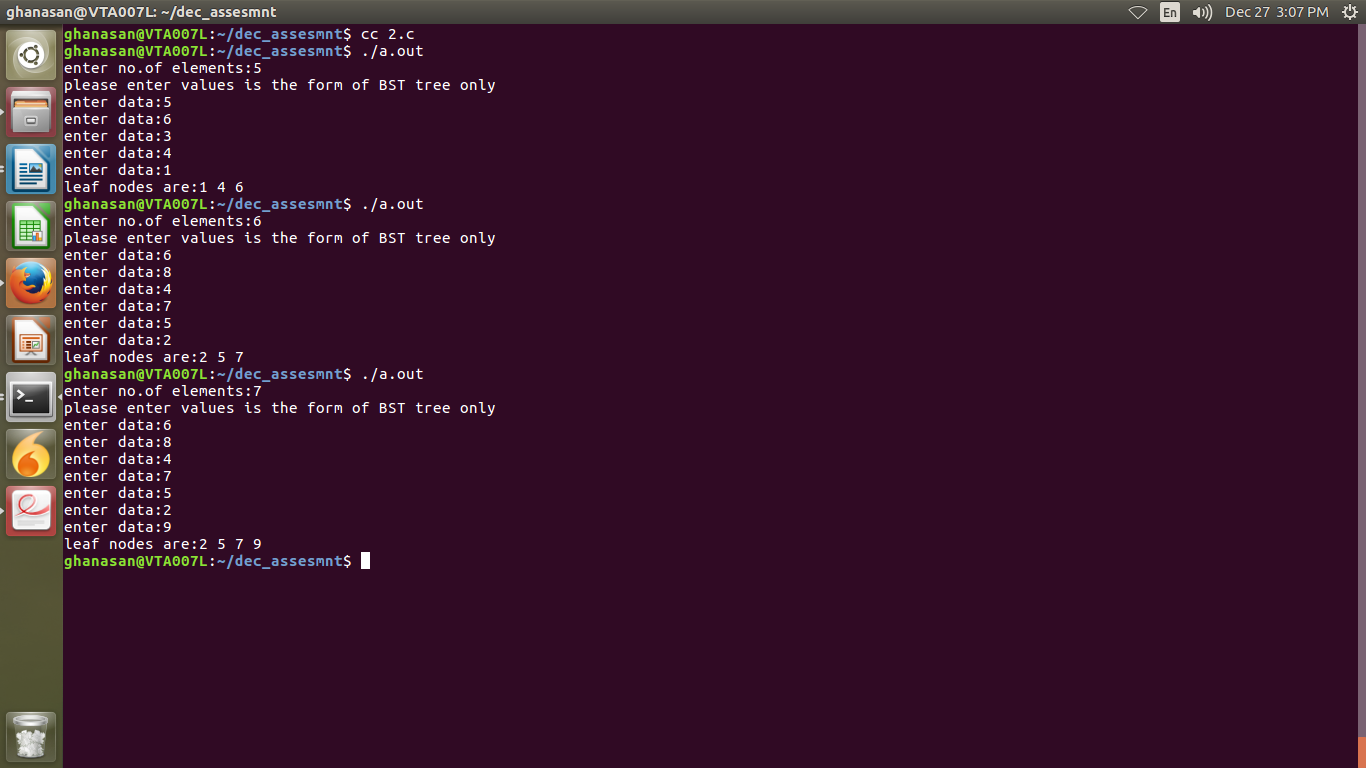
}

printf("leaf nodes are:");

print\_leaf(root);

printf("\n");

}

****

3.

#include<stdio.h>

int invert(int num,int p,int n)

{

int ind;

for(ind=p;ind<=n;ind++)

{

num^=(1<<ind);

}

return num;

}

int main()

{

int num,pos1,pos2,res;

printf("enter number:");

scanf("%d",&num);

printf("enter pos1:");

scanf("%d",&pos1);

printf("enter pos2:");

scanf("%d",&pos2);

if(pos1>pos2)

puts("pos1 must less than pos2");

else

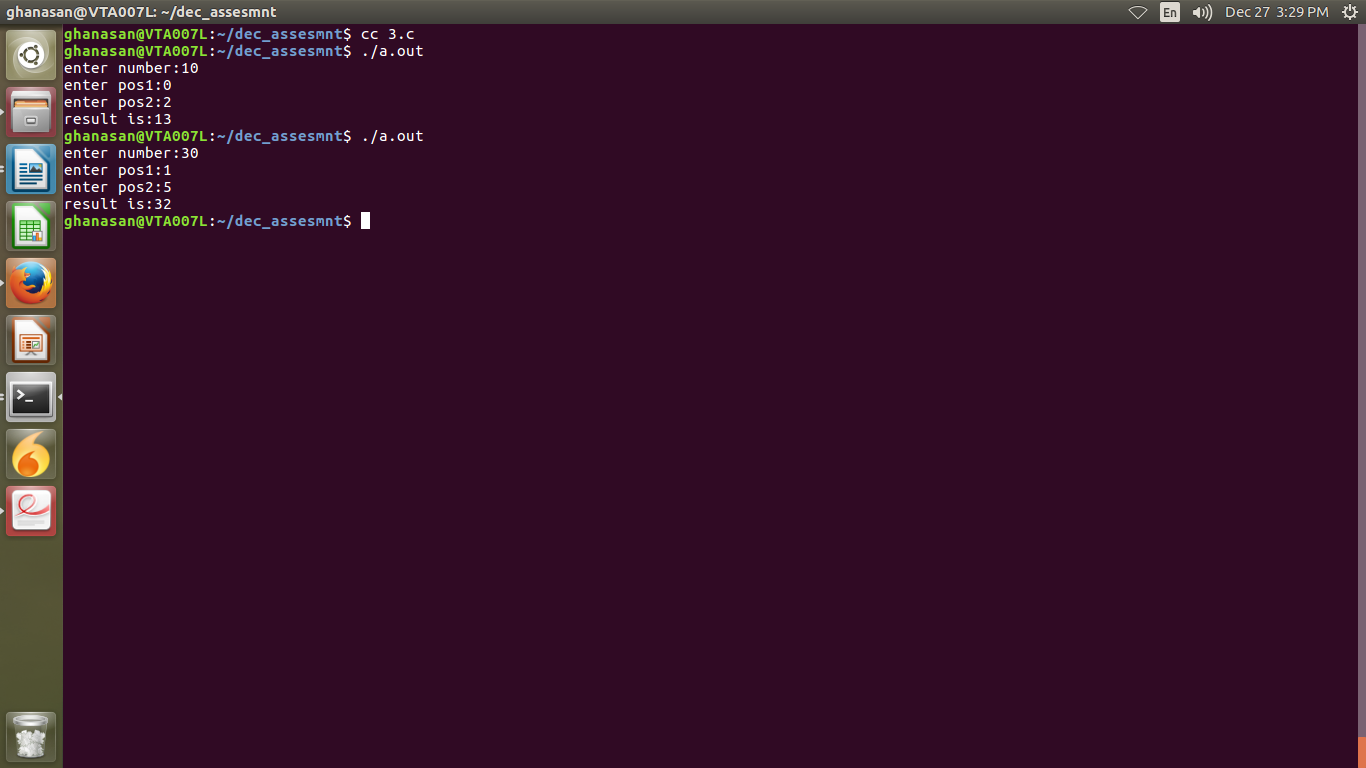
{

res=invert(num,pos1,pos2);

printf("result is:%d\n",res);

}

}

****

4.

#include<stdio.h>

int main()

{

int arr[10],res[10],ind1,ind2,cnt=0;

puts("enter 10 values into array:");

for(ind1=0;ind1<10;ind1++)

scanf("%d",&arr[ind1]);

for(ind1=0;ind1<10;ind1++)

{

for(ind2=0;ind2<10;ind2++)

{

if((arr[ind1]==arr[ind2])&&(ind1!=ind2))

break;

}

if(ind2==10)

res[cnt++]=arr[ind1];

}

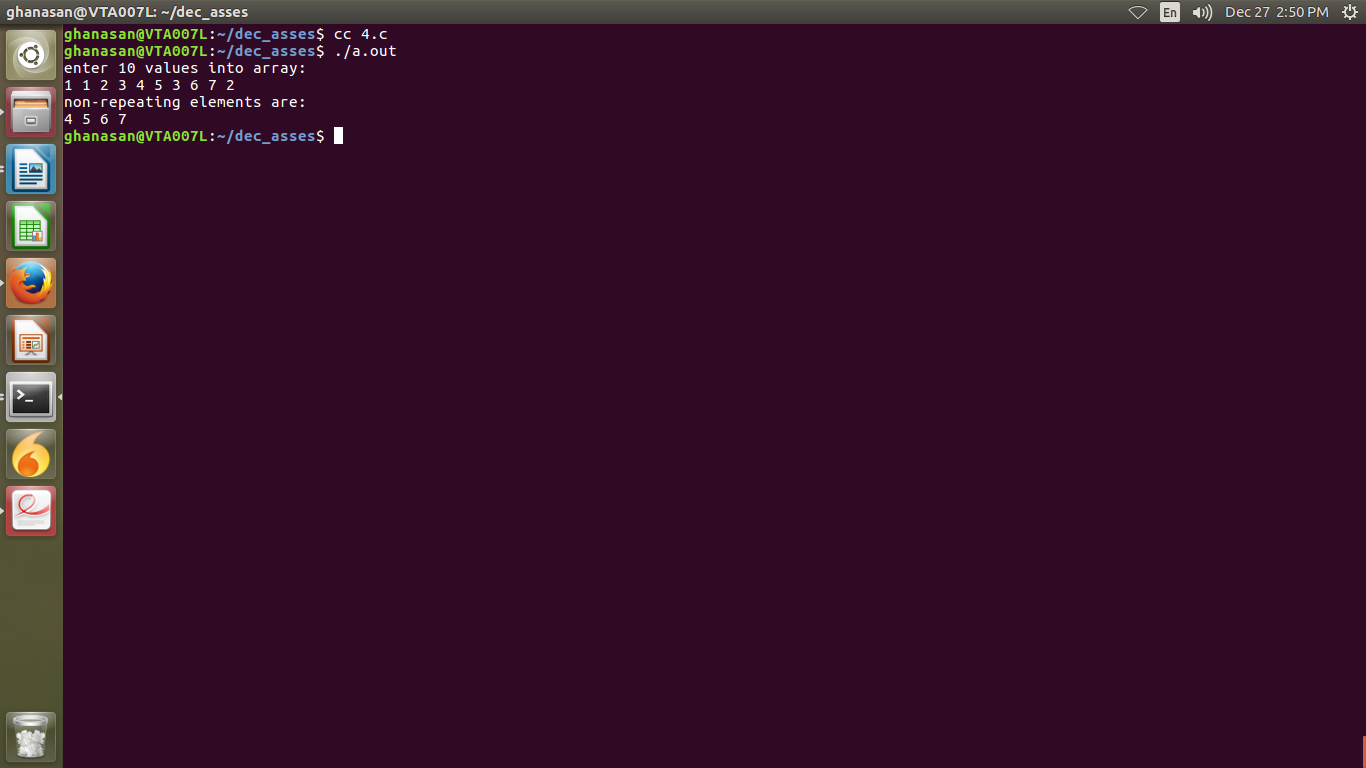
puts("non-repeating elements are:");

for(ind1=0;ind1<cnt;ind1++)

printf("%d ",res[ind1]);

printf("\n");

}

****

6.

#include<stdio.h>

int main()

{

int size1,size2;

printf("enter size of arr1:");

scanf("%d",&size1);

printf("enter size of arr2:");

scanf("%d",&size2);

int arr1[size1],arr2[size2];

int ind1,ind2,cnt=0;

printf("enter values of arr1:");

for(ind1=0;ind1<size1;ind1++)

scanf("%d",&arr1[ind1]);

printf("enter values of arr2:");

for(ind1=0;ind1<size2;ind1++)

scanf("%d",&arr2[ind1]);

for(ind1=0;ind1<size2;ind1++)

{

for(ind2=0;ind2<size1;ind2++)

{

if(arr2[ind1]==arr1[ind2])

break;

}

if(ind2==size1)

{

printf("arr2 is not a subset of arr1\n");

return 0;

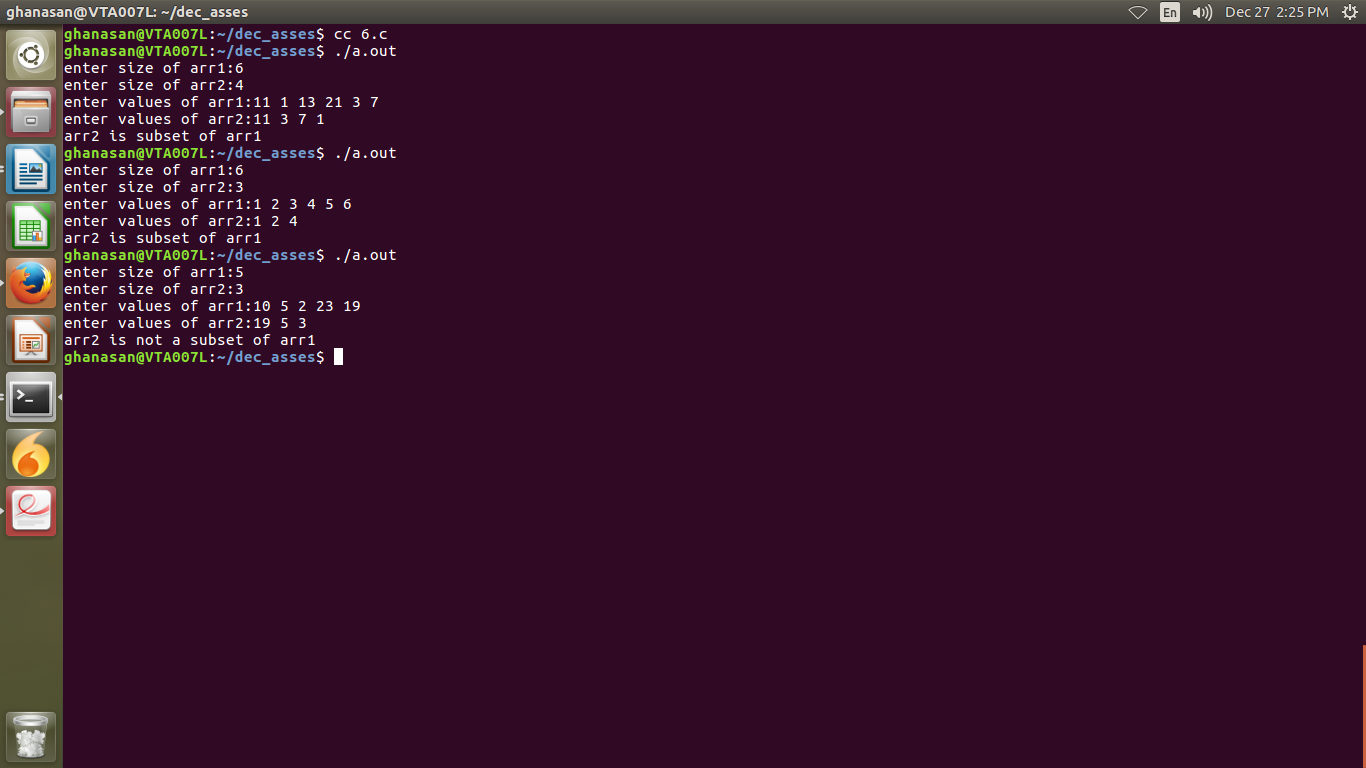
}

}

printf("arr2 is subset of arr1\n");

return 0;

}

****