**PROGRAM CODE:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include<stdlib.h>

#include "intstack.h"

#include "charstack.h"

int getPriority(char x)

{

if(x=='/' || x == '\*' || x=='%')

return 1;

else if(x=='+' || x=='-')

return 0;

}

int InfixtoPostfix(char infix[],char postfix[])

{

stack s1=createstack(strlen(infix));

int i=0, j=0;

while(infix[i]!='\0')

{

if(infix[i]=='(')

{

push(infix[i],s1);

i++;

}

else if(infix[i] == ')')

{

while((return\_top(s1)!=-1) && (return\_top(s1)!='('))

{

postfix[j] = topandpop(s1);

j++;

}

pop(s1);//remove left parenthesis

i++;

}

else if(isdigit(infix[i]) || isalpha(infix[i]))

{

postfix[j] = infix[i];

j++;

i++;

}

else if (infix[i] == '+' || infix[i] == '-' || infix[i] == '\*' || infix[i] == '/' || infix[i] == '%')

{

while( (return\_top(s1)!=-1) && (return\_top(s1)!= '(') && (getPriority(return\_top(s1)) >= getPriority(infix[i]) ) )

{

postfix[j] = topandpop(s1);

j++;

}

push(infix[i],s1);

i++;

}

else

{

printf("Invalid element: %c\n",infix[i]);

return 0;

}

}

while((return\_top(s1)!=-1))

{

postfix[j] = topandpop(s1);

j++;

}

postfix[j]='\0';

return 1;

}

int evaluate(char postfix[])

{

int i=0;

int op1, op2, value;

int\_stack s2=create\_int\_stack(strlen(postfix));

while(postfix[i] != '\0')

{

if(isdigit(postfix[i]))

intpush((int)postfix[i]-48,s2);

else

{

op2 = int\_topandpop(s2);

op1 = int\_topandpop(s2);

switch(postfix[i])

{

case '+':

value = op1 + op2; break;

case '-':

value = op1 - op2; break;

case '/':

value = op1 / op2; break;

case '\*':

value = op1 \* op2; break;

case '%':

value = op1 % op2; break;

} //end switch

intpush(value,s2);

} //end else

i++;

} //end while

return(int\_return\_top(s2));

}

void main()

{

int val;

char infix[100],postfix[100];

printf("Enter any infix expression : ");

gets(infix);

if(InfixtoPostfix(infix,postfix))

{

printf("The corresponding postfix expression is : %s\n",postfix);

printf("Enter the postfix expression with the values: ");

gets(postfix);

val = evaluate(postfix);

printf("Value of the postfix expression = %d\n", val);

}

}

**charstack header file:**

typedef struct stackrecord\* stack;

struct stackrecord

{

int capacity;

int topofstack;

char \*array;

};

stack createstack(int maxelements)

{

stack s=(stack)malloc(sizeof(struct stackrecord));

if(s==NULL)

printf("Out of space!");

s->array=malloc(sizeof(char) \* maxelements);

if(s->array == NULL)

printf("Out of space!");

s->capacity = maxelements;

s->topofstack = -1;

return s;

}

void disposestack(stack s)

{ if(s!=NULL)

{ free(s->array);

free(s); }

}

int isempty(stack s)

{

return s->topofstack == -1;

}

int isfull(stack s)

{

return s->topofstack==s->capacity-1;

}

void push(char x,stack s)

{

if(isfull(s))

printf("Stack is full");

else

s->array[++s->topofstack] = x;

}

void pop(stack s)

{

if(isempty(s))

printf("Stack is empty");

else

s->topofstack--;

}

void display(stack s)

{

int i;

for(i=0;i<s->topofstack;i++)

printf("%c",s->array[i]);

}

char return\_top(stack s)

{

if(!isempty(s))

return s->array[s->topofstack];

else

return s->topofstack;

}

char topandpop(stack s)

{

if(!isempty(s))

return s->array[s->topofstack--];

else

return s->topofstack;

}

**intstack header file:**

typedef struct int\_stackrecord\* int\_stack;

struct int\_stackrecord

{

int capacity;

int topofstack;

int \*array;

};

int\_stack create\_int\_stack(int maxelements)

{

int\_stack s=malloc(sizeof(struct int\_stackrecord));

if(s==NULL)

printf("Out of space!");

s->array=malloc(sizeof(int) \* maxelements);

if(s->array == NULL)

printf("Out of space!");

s->capacity = maxelements;

s->topofstack = -1;

return s;

}

void int\_disposestack(int\_stack s)

{

if(s!=NULL)

{

free(s->array);

free(s);

}

}

int int\_isempty(int\_stack s)

{

return s->topofstack == -1;

}

int int\_isfull(int\_stack s)

{

return s->topofstack==s->capacity-1;

}

void intpush(int x,int\_stack s)

{

if(int\_isfull(s))

printf("Stack is full");

else

s->array[++s->topofstack] = x;

}

void intpop(int\_stack s)

{

if(int\_isempty(s))

printf("Stack is empty");

else

s->topofstack--;

}

void int\_display(int\_stack s)

{

int i;

for(i=0;i<s->topofstack;i++)

printf("%c",s->array[i]);

}

int int\_return\_top(int\_stack s)

{

if(!isempty(s))

return s->array[s->topofstack];

else

return s->topofstack;

}

int int\_topandpop(int\_stack s)

{

if(!int\_isempty(s))

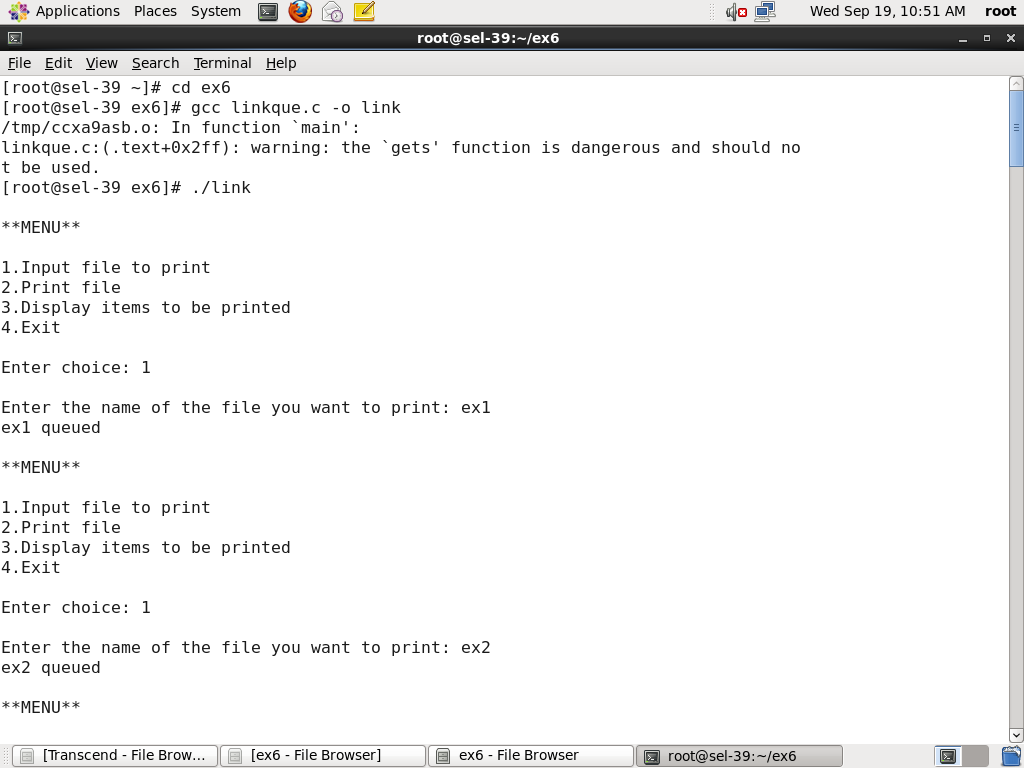
return s->array[s->topofstack--];

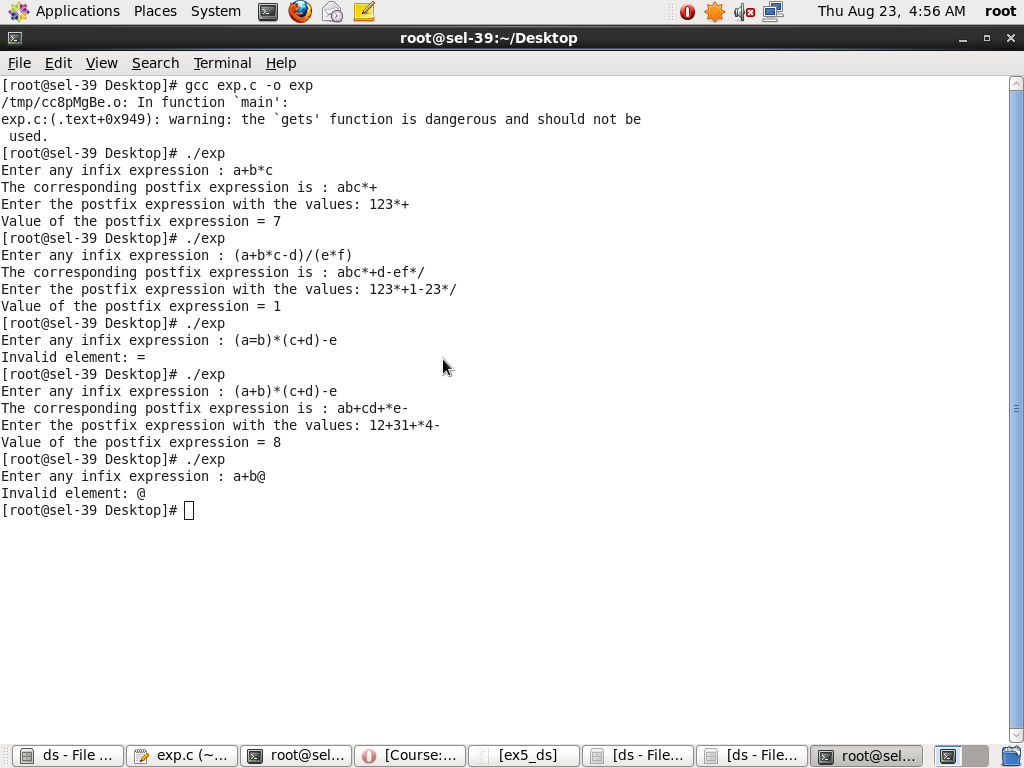
else

return s->topofstack;

}

**OUTPUT:**





**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include "queuelist.h"

void main()

{

queue q =createqueue();

int choice;

char ele[20];

do

{

printf("\n\*\*MENU\*\*\n\n1.Input file to print\n2.Print file\n3.Display items to be printed\n4.Exit\n\nEnter choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnter the name of the file you want to print: ");

getchar();

gets(ele);

printf("%s queued\n",ele);

enqueue(ele,q);

break;

case 2:

if(!isempty(q))

{

printf("%s printed \n",dequeue(q));

}

else

printf("\nNo file to print. Could not print.\n");

break;

case 3:

if(!isempty(q))

displayQueue(q);

else

printf("\nQueue is empty.No items to be printed.\n");

break;

case 4:

disposeQueue(q);

printf("\nAll files in queue have been discarded.\n");

break;

default:

printf("Incorrect choice!");

}//switch

}while(choice!=4);

}

**queuelist header file**

typedef struct node\* ptrtonode;

typedef struct queuelist\* queue;

struct node

{

char element[20];

ptrtonode next;

};

struct queuelist

{

ptrtonode front;

ptrtonode rear;

};

void initqueue(queue q)

{ q->front = q->rear = NULL; }

int isempty(queue q)

{ return q->front==NULL; }

queue createqueue()

{

queue q=(queue)malloc(sizeof(struct queuelist));

if(q==NULL)

{ printf("Out of space");

exit(0);

}

initqueue(q);

return q;

}

void enqueue(char x[], queue q)

{

ptrtonode tmpcell=(ptrtonode)malloc(sizeof(struct node));

if(tmpcell==NULL)

{

printf("Out of space");

exit(0);

}

else

{

strcpy(tmpcell->element, x);

tmpcell->next = NULL;

}

if(q->front==NULL)

q->rear=q->front=tmpcell;

else

{

q->rear->next=tmpcell;

q->rear=q->rear->next;

}

}

char\* dequeue(queue q)

{

ptrtonode temp;

static char data[20];

if(isempty(q))

{

printf("Empty Queue");

return NULL;

}

else

{

strcpy(data,q->front->element);

temp=q->front;

q->front=q->front->next;

free(temp);

return data;

}

}

void displayQueue(queue q)

{

ptrtonode tmp=q->front;

printf("\nFiles to be printed: \n");

while(tmp!=NULL)

{

printf("\n%s",tmp->element);

tmp=tmp->next;

}

printf("\n");

}

void disposeQueue(queue q)

{

ptrtonode temp;

while(!isempty(q))

{

temp=q->front;

q->front=q->front->next;

if(isempty(q))

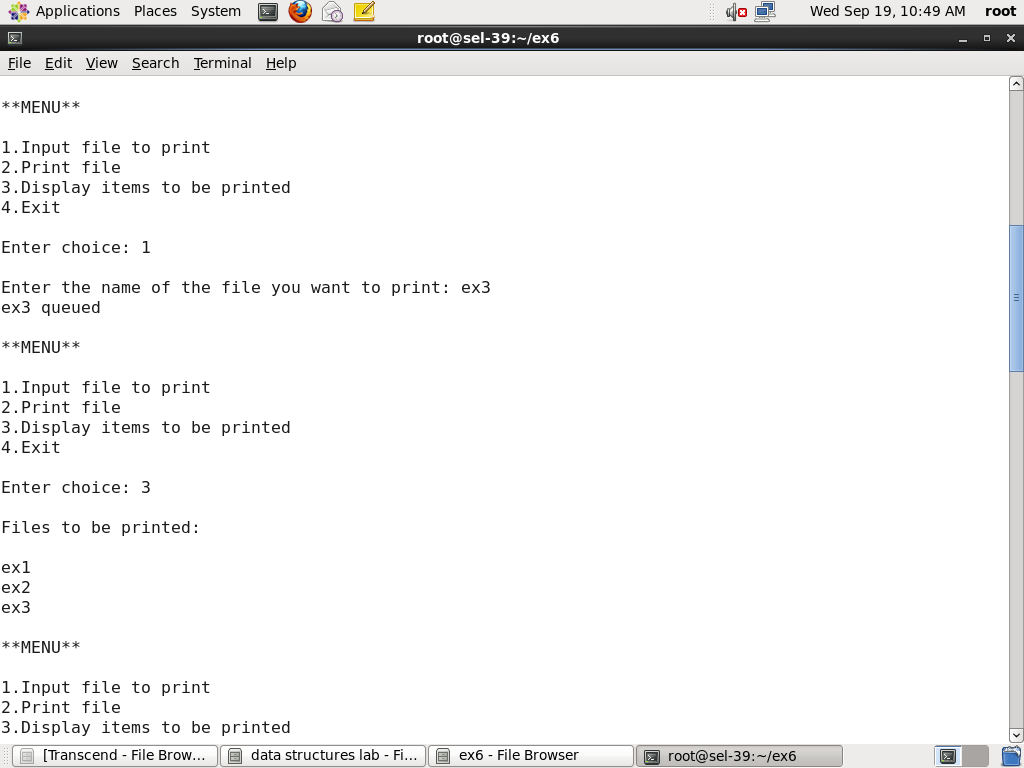
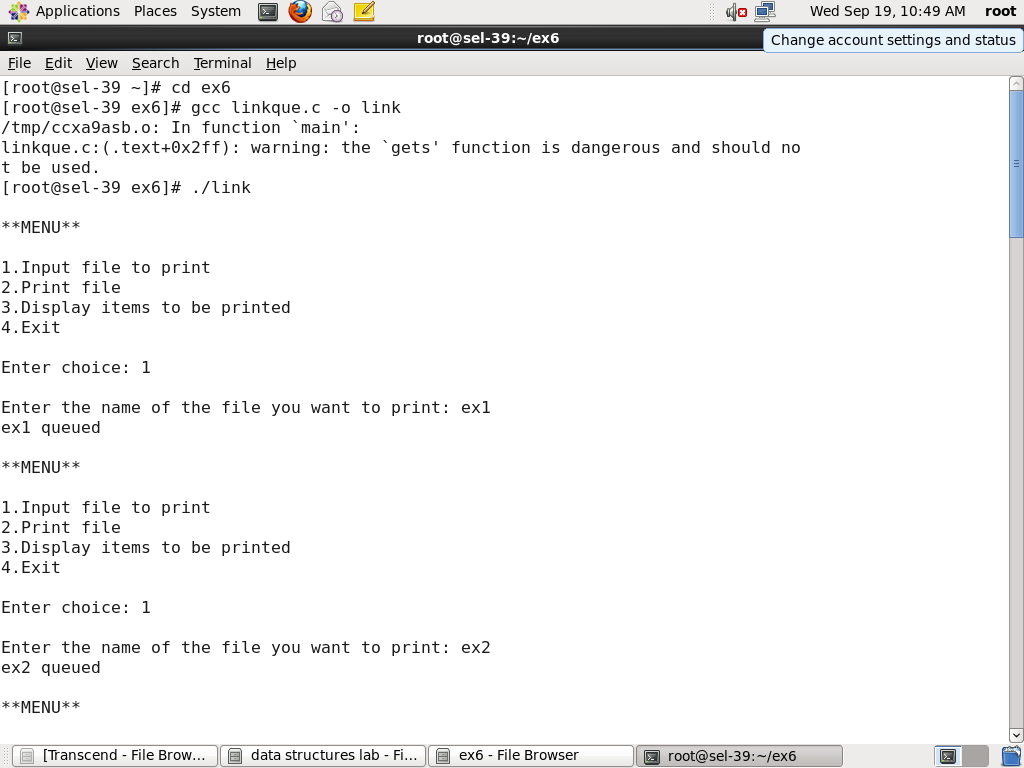
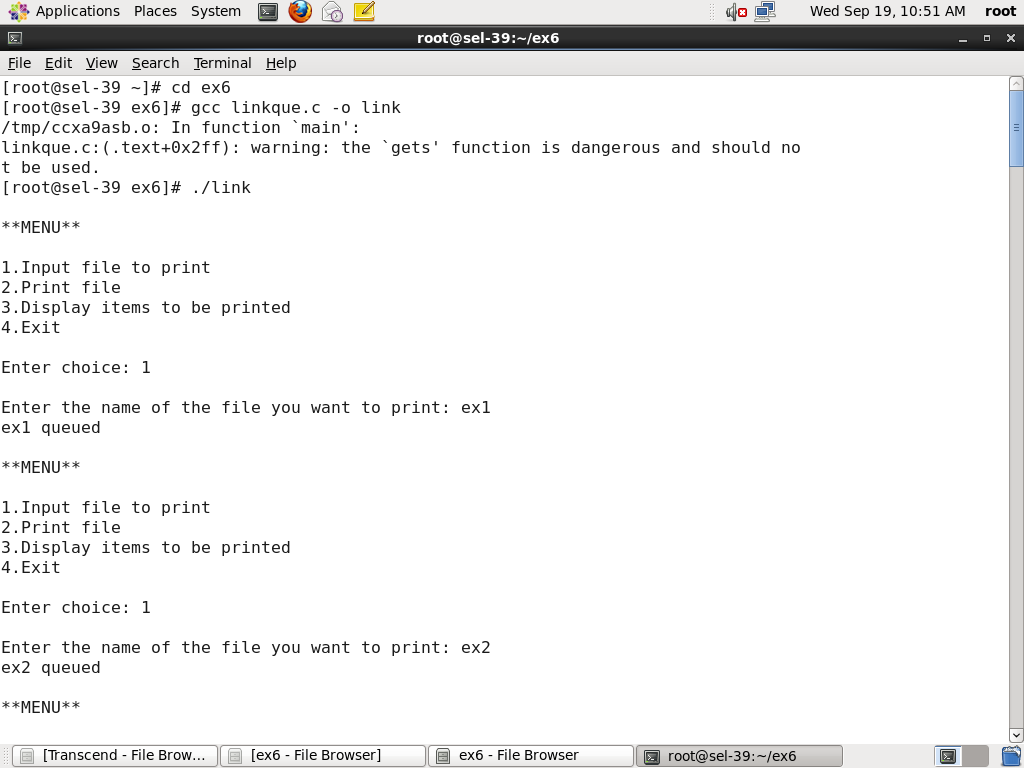
initqueue(q);

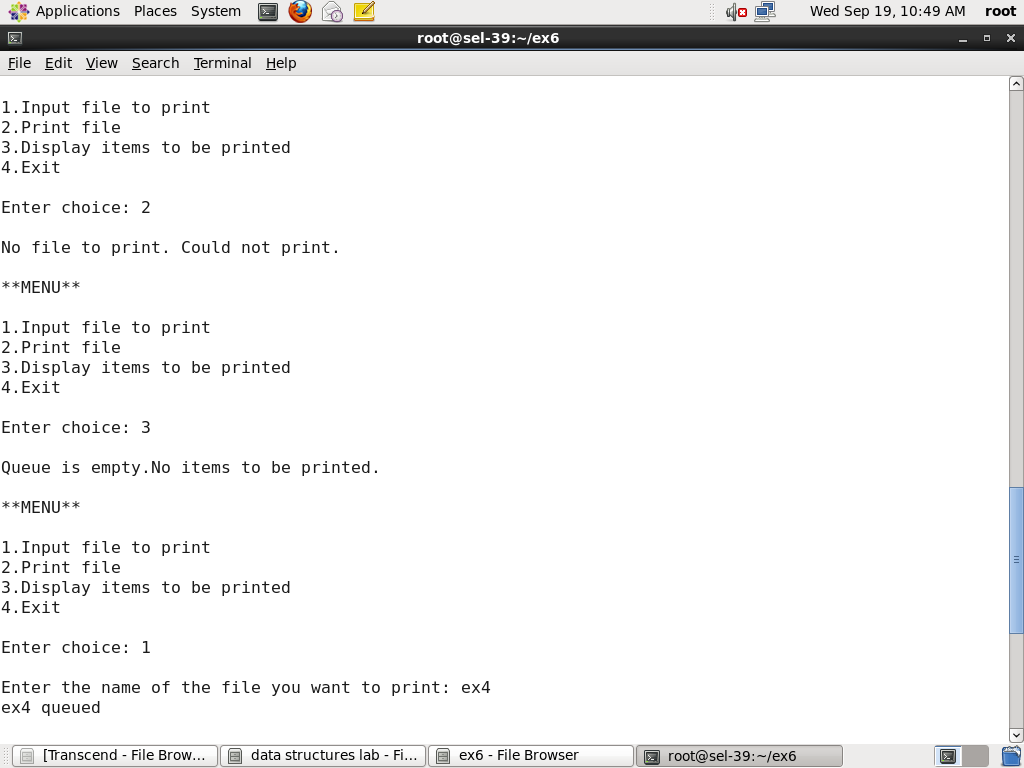
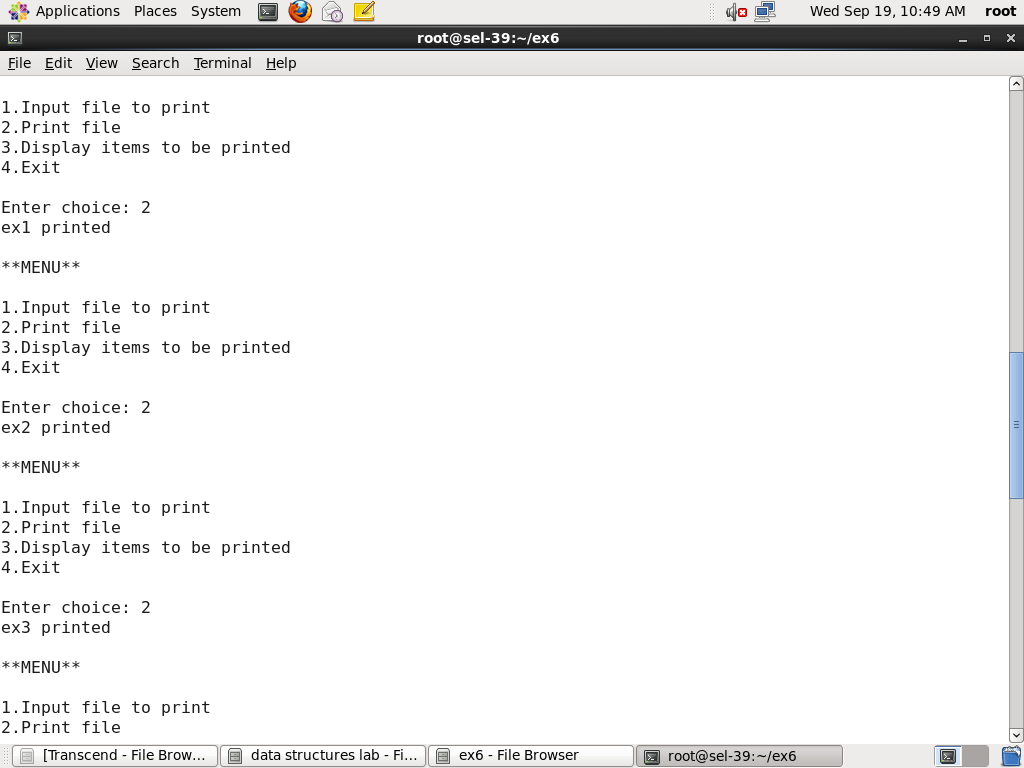
free(temp);

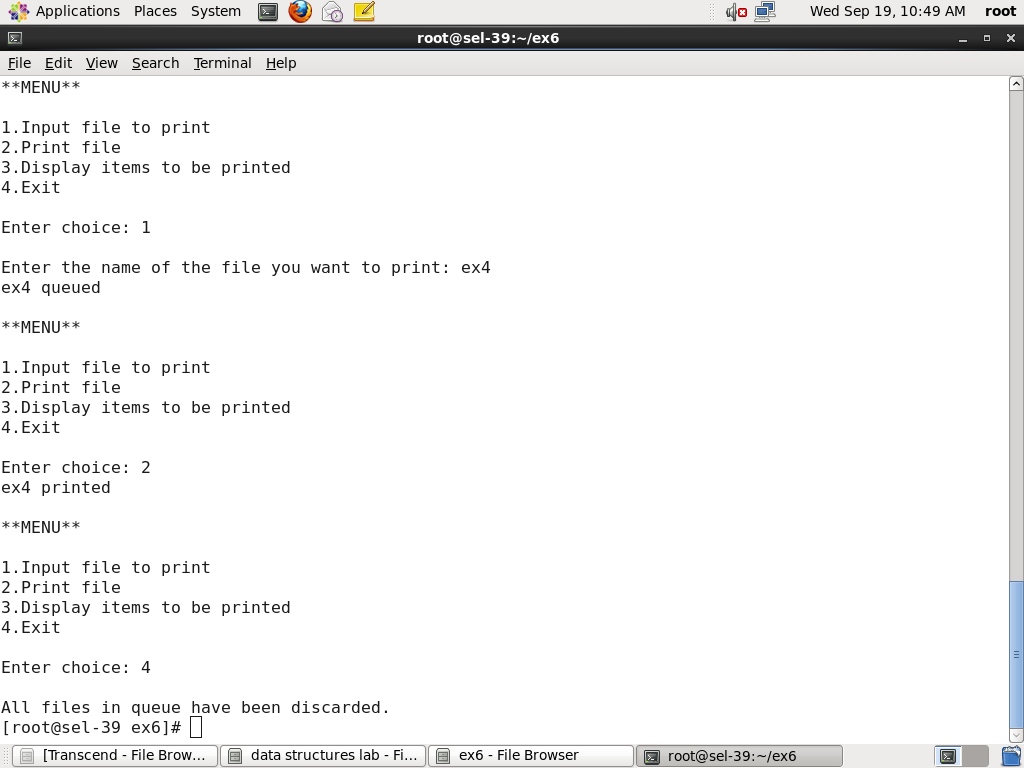
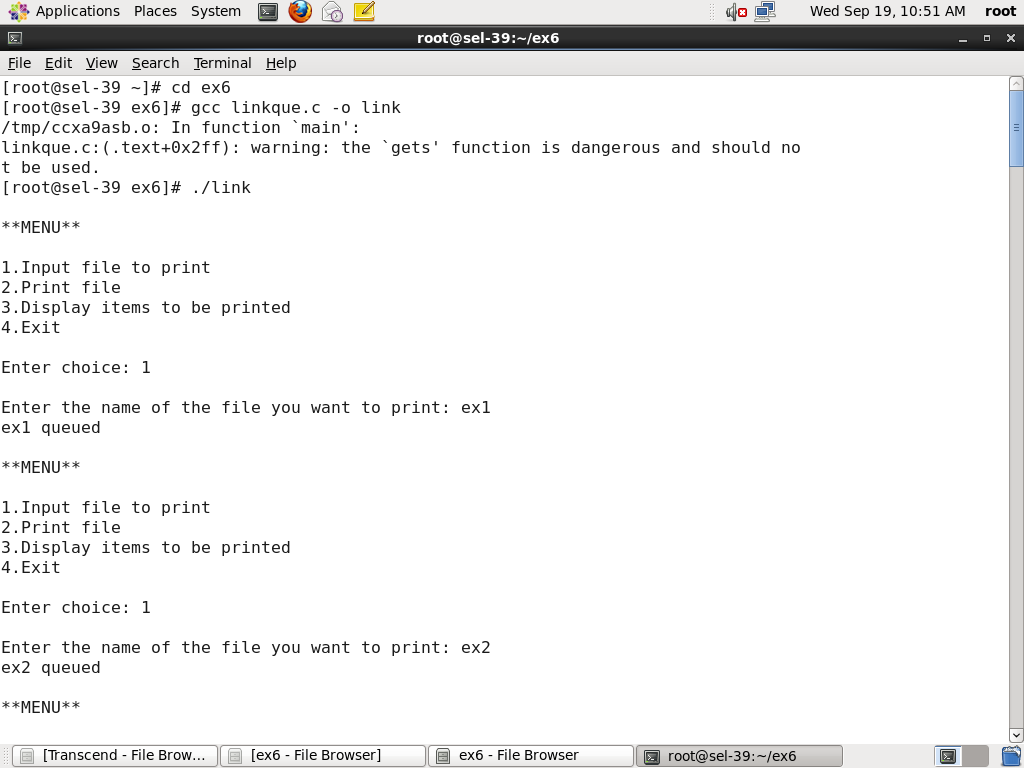
}

}

**OUTPUT:**







**PROGRAM CODE:**

#include <stdio.h>

#include <string.h>

#include<stdlib.h>

#include "queuearr.h"

void main()

{

int max,choice;

char ele[15];

char\*x;

printf("Enter maximum no of items that can be stored in the container: ");

scanf("%d",&max);

queue q =createqueue(max);

do

{

printf("\n\*\*MENU\*\*\n1.Producer\n2.Consumer\n3.Display items in the container\n4.Exit\n\nEnter choice:");

scanf("%d",&choice);

switch(choice)

{

case 1: if(!isfull(q))

{ printf("\nEnter item that is to be stored in the container: ");

scanf("%s",ele);

enqueue(q,ele); }

else

printf("\nContainer is full! Could not store\n");

break;

case 2: x=dequeue(q);

if(x!=NULL)

printf("Item %s consumed.\n",x);

else

printf("Container is empty.Could not consume!\n");

break;

case 3: display(q);

break;

case 4: disposequeue(q);

break;

default: printf("Incorrect choice!");

}//switch

}while(choice!=4);

}

**queuearr adt file:**

typedef struct queuerecord\* queue;

typedef char elementtype[30]; //elementtype can hold 30 characters -a string

struct queuerecord

{

int capacity;

int front;

int rear;

int size;

elementtype \*arr; //arr is a pointer to string

};

void makeempty(queue q)

{

q->size = 0;

q->front = -1;

q->rear = -1;

}

queue createqueue(int maxelements)

{

int i;

queue q=(queue)malloc(sizeof(struct queuerecord));

if(q==NULL)

printf("Out of space!");

q->arr=(elementtype\*)malloc(sizeof(elementtype) \* maxelements);

if(q->arr == NULL)

printf("Out of space!");

q->capacity = maxelements;

makeempty(q);

return q;

}

int isempty (queue q)

{ return q->size == -1; }

void display(queue q)

{

int i;

if(isempty(q))

printf("\nContainer is empty");

else

{ printf("\nThe products in the container are: ");

i=q->front;

while(i!=q->rear)

{ printf("\n%s",q->arr[i]); i=(i+1)%q->capacity; } //end while

printf("\n%s\n",q->arr[q->rear]);

} //end else

}

void disposequeue(queue q)

{

if(q!=NULL)

{

free(q->arr);

free(q);

}

}

int isfull (queue q)

{ return q->size == q->capacity; }

void enqueue(queue q, elementtype item)

{

if(q->rear+1==q->capacity)

q->rear=0;

else

q->rear++;

strcpy(q->arr[q->rear],item);

if(q->front==-1)

q->front=0;

++q->size;

}

char\* dequeue(queue q)

{

static char data[15];

if(isempty(q))

{ return NULL; }

else

{

strcpy(data,q->arr[q->front]);

if(q->front==q->rear) //only one element is present

makeempty(q);

else if(q->front+1 == q->capacity)

q->front=0; //front is pointing to end of arr

else

q->front++;

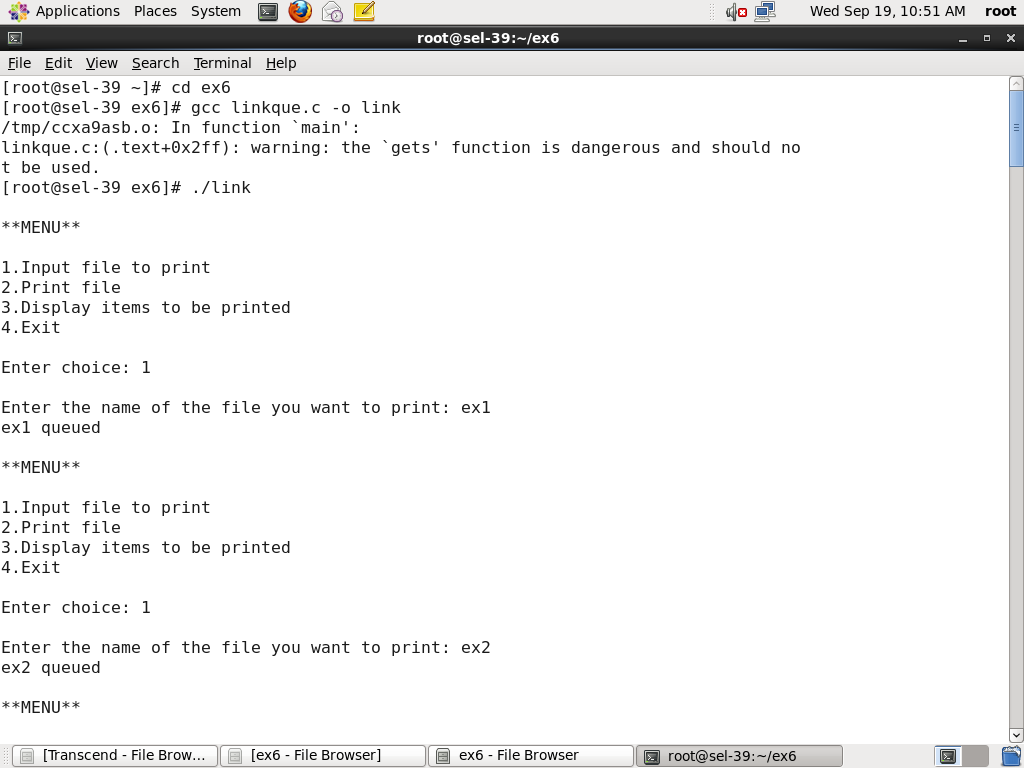
q->size--;

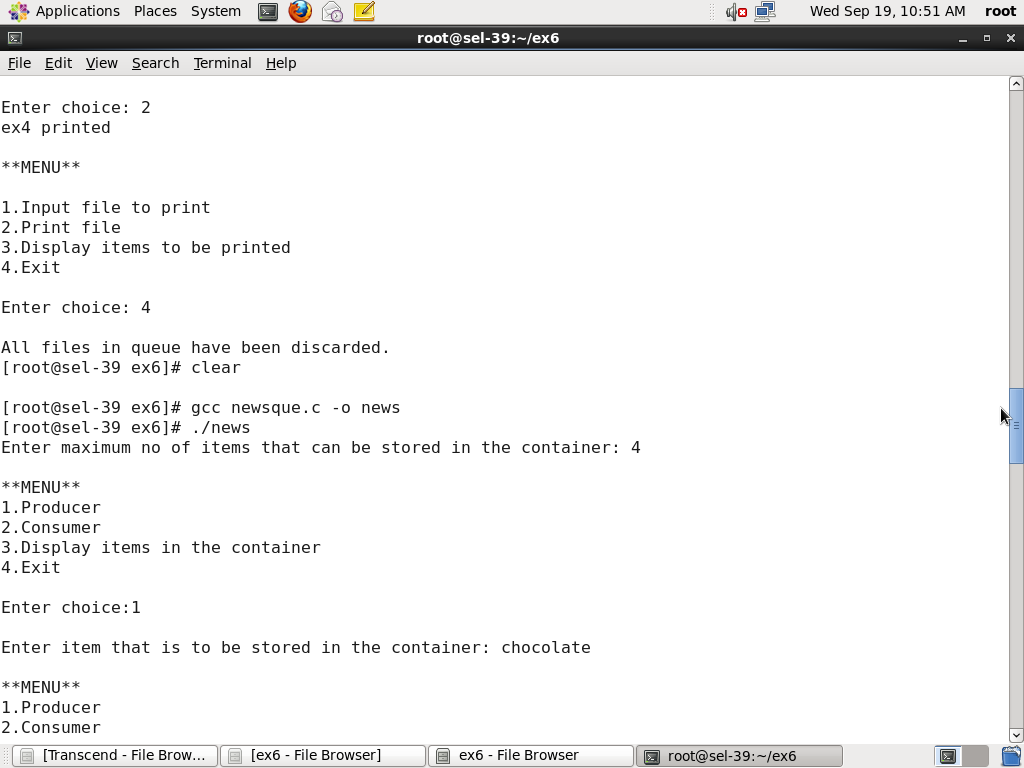
return data;

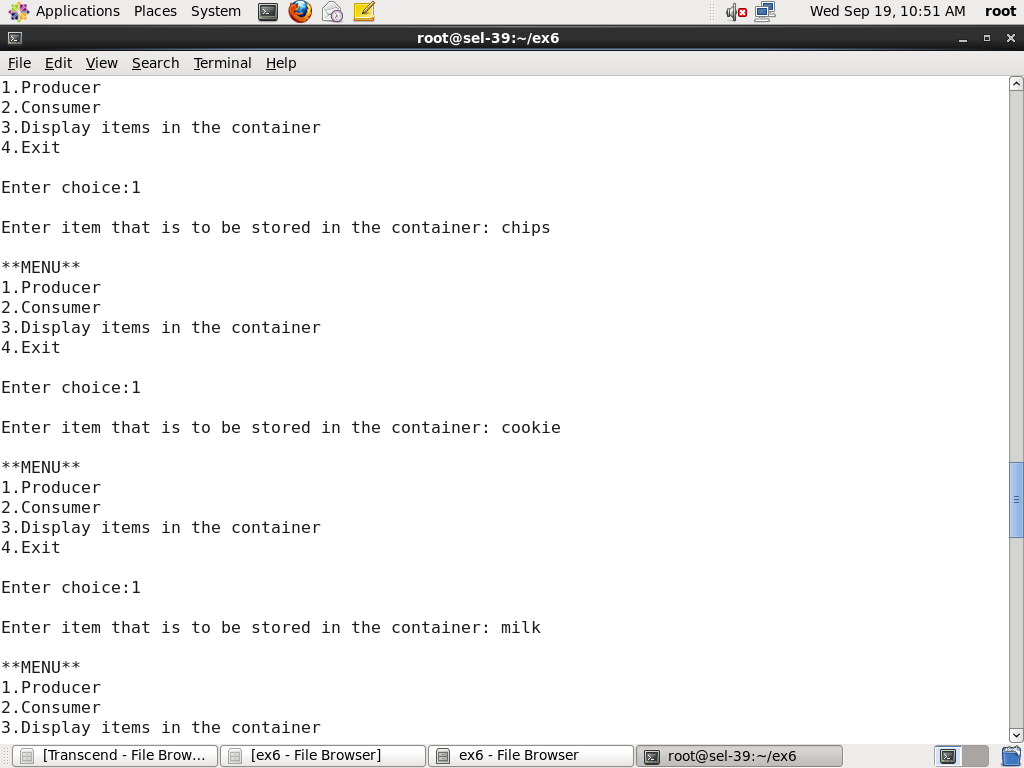
}

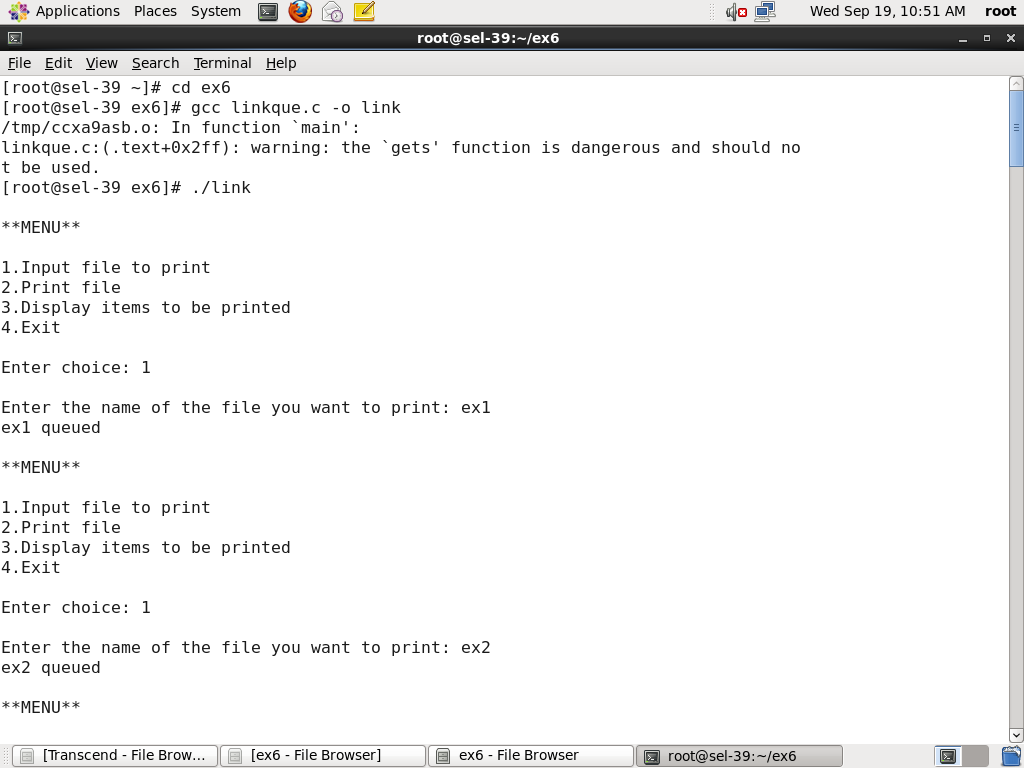
}

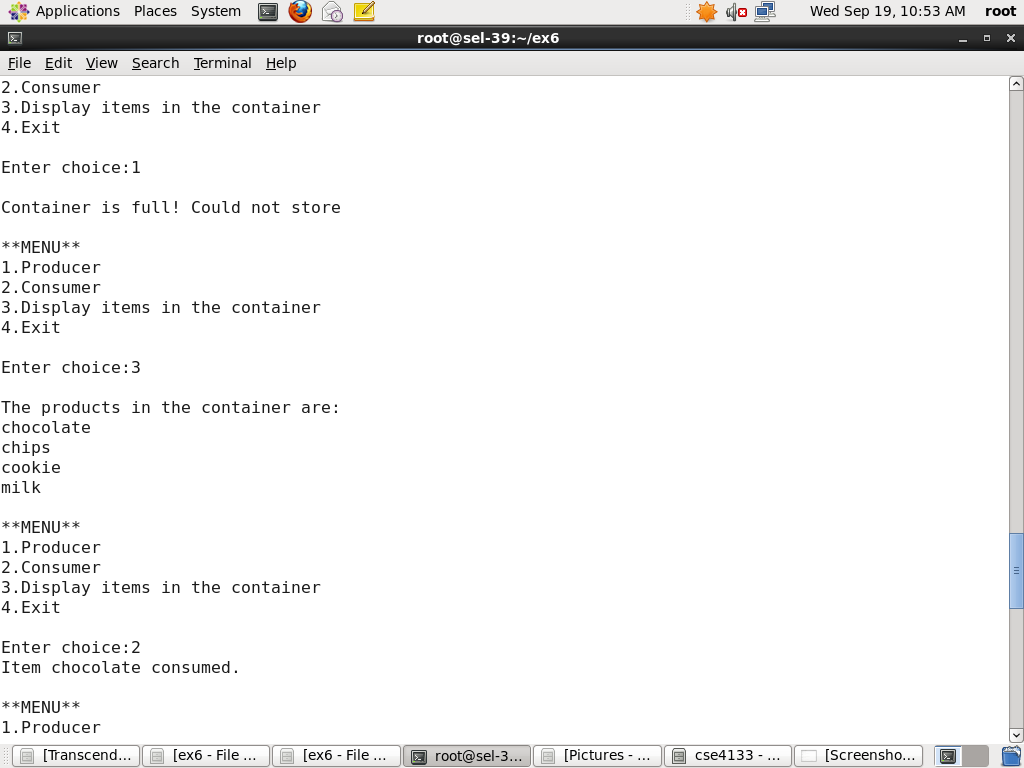
**OUTPUT:**

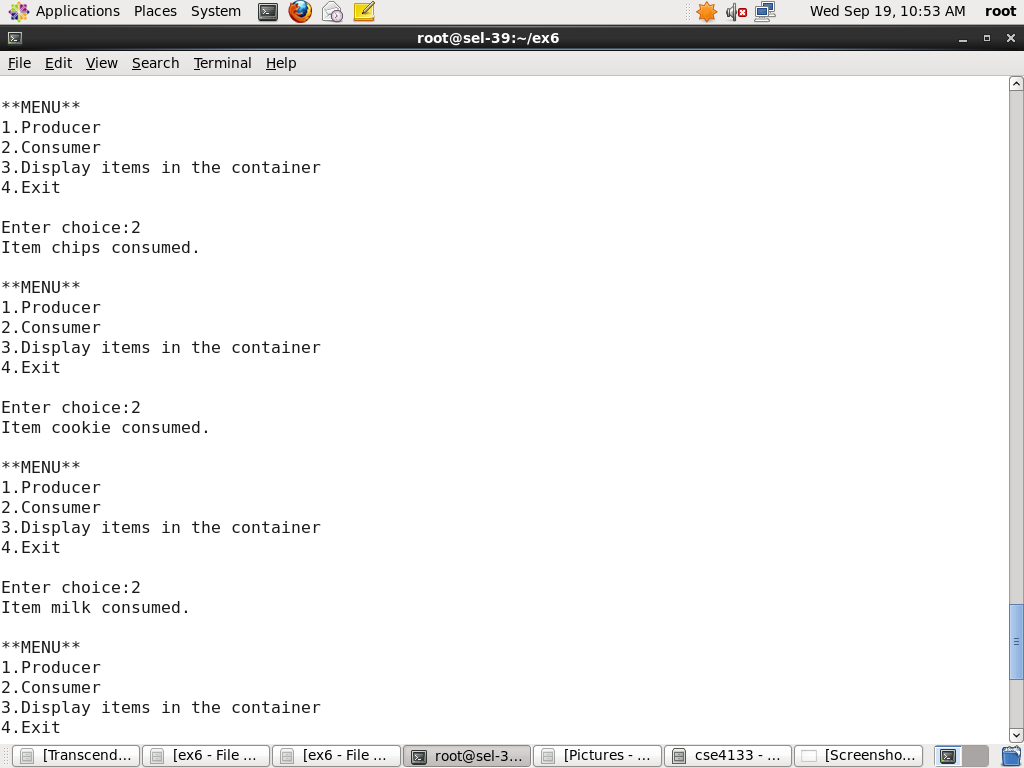


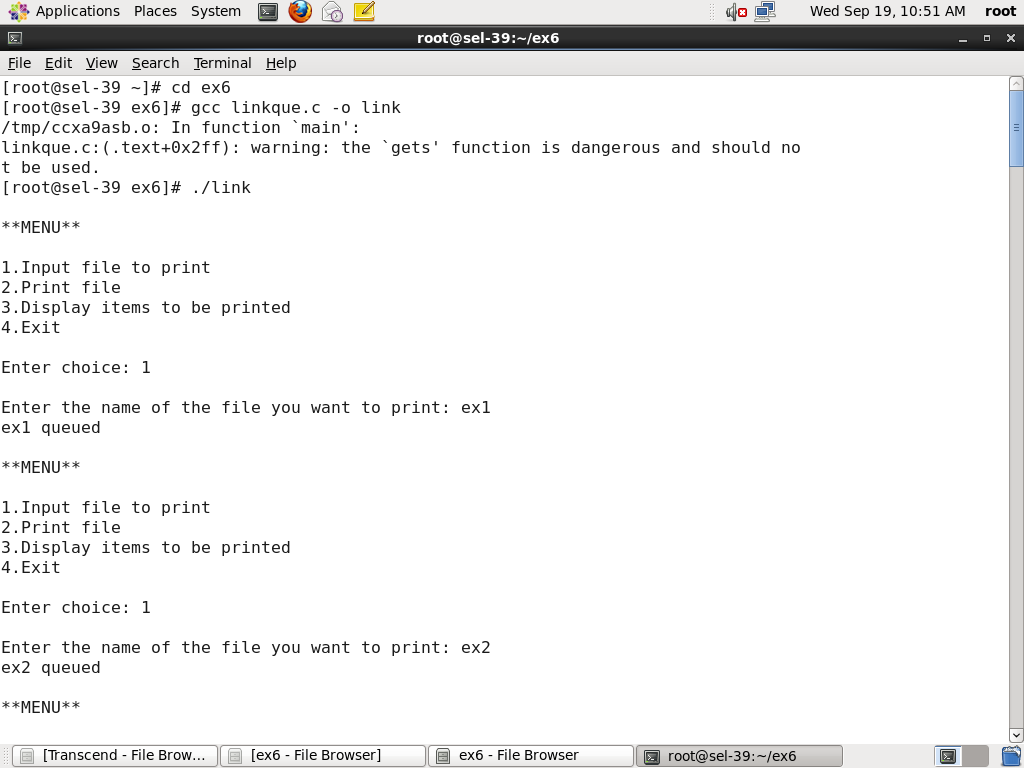
****

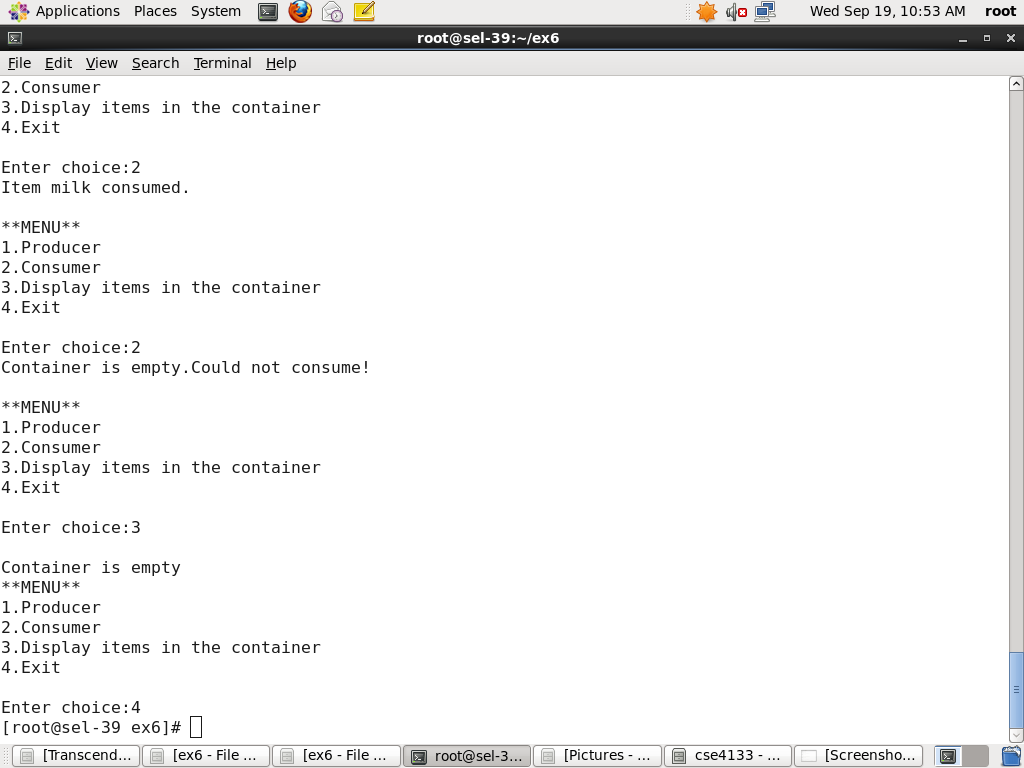
****



****

****



****

**PROGRAM CODE:**

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#include "exptree.h"

void main()

{

char postfix[100];

int i=0;

tree t;

stack s=createstack();

printf("Enter the postfix expression: ");

gets(postfix);

while(postfix[i]!='\0')

{

if(isalpha(postfix[i]) || isdigit(postfix[i]))

{

t=createtree();

t->element=postfix[i];

t->left=t->right=NULL;

push(t,s);

}

else if(postfix[i]=='+' || postfix[i]=='-' || postfix[i]=='\*' || postfix[i]=='/' || postfix[i]=='%')

{

t=createtree();

t->element=postfix[i];

t->right=pop(s);

t->left=pop(s);

push(t,s);

}

i++;

}

printf("\nPRE ORDER TRAVERSAL\n");

preorder(t);

printf("\nPOST ORDER TRAVERSAL\n");

postorder(t);

printf("\nIN ORDER TRAVERSAL\n");

inorder(t);

printf("\n");

}

***exptree header file*:**

typedef struct treenode \*tree;

typedef struct node \*ptrtonode;

typedef ptrtonode stack;

struct treenode

{

char element;

tree left;

tree right;

};

struct node

{

tree element;

ptrtonode next;

};

int isempty(stack s)

{

return s->next==NULL;

}

stack createstack( )

{

stack s=(stack)malloc(sizeof(struct node));

if(s==NULL)

printf("Out of space");

return s;

}

tree pop(stack s)

{

ptrtonode tmpcell;

tree x;

if(isempty(s))

printf("\nStack is empty");

else

{

tmpcell = s->next;

x=tmpcell->element;

s->next = s->next->next;

free(tmpcell);

}

return x;

}

void push(tree x, stack s)

{

ptrtonode tmpcell=(stack)malloc(sizeof(struct node));

if(tmpcell==NULL)

printf("Out of space");

else

{

tmpcell->element = x;

tmpcell->next = s->next;

s->next = tmpcell;

}

}

tree createtree( )

{

tree t=(tree)malloc(sizeof(struct treenode));

if(t==NULL)

printf("Out of space");

return t;

}

void postorder(tree t)

{

if ( t == NULL)

return;

postorder( t->left);

postorder( t->right);

printf("%c ", t->element);

}

void preorder(tree t)

{

if ( t == NULL)

return;

printf("%c ", t->element);

preorder( t->left);

preorder( t->right);

}

void inorder(tree t)

{

if ( t == NULL)

return;

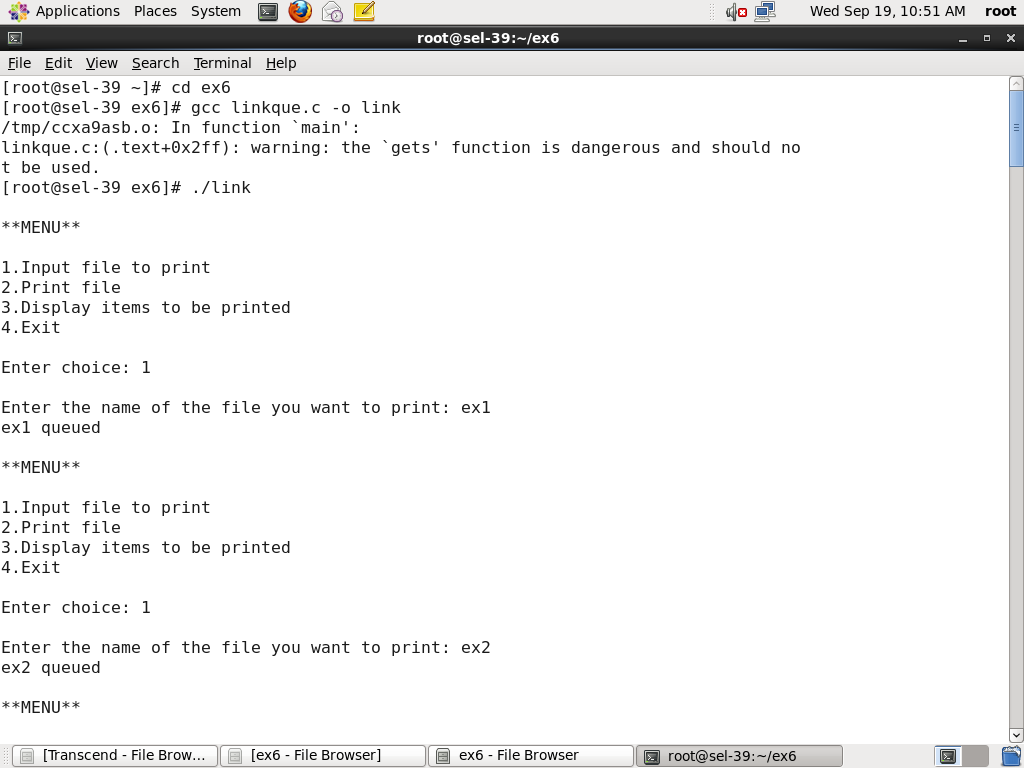
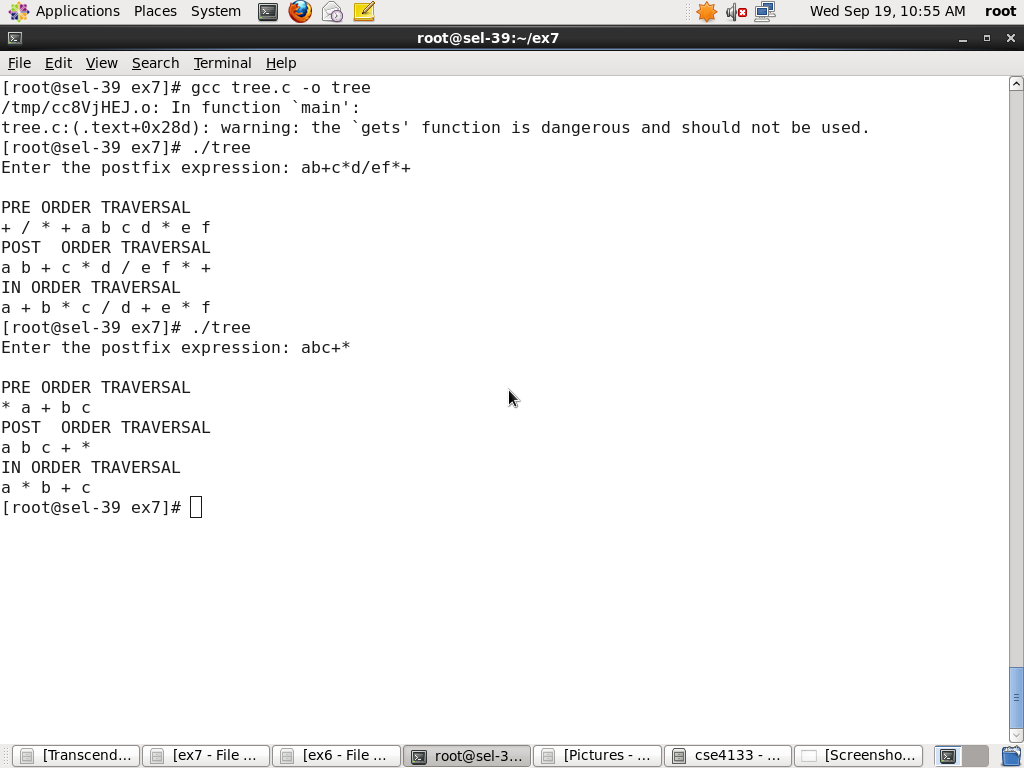
inorder( t->left);

printf("%c ", t->element);

inorder( t->right);

}

**OUTPUT:**

****