PART A

5. Write a C program to implement the following contiguous memory allocation techniques :
a) Worst-fit
b) Best-fit
c) First-fit

```
#include <stdio.h>
#include <stdlib.h>
int processMemory[100],tempMemory[100],memory[100];
int noMemoryBlock,noProcesses;
void fnFirstFit(int memory[])
{
	int i,j;
	printf("\nFirstFit\nProcess\t\tMemoryBlock");
	for(i=0;i<noProcesses;i++)
	{
		int flag=0;
		for(j=0;j<noMemoryBlock;j++)
		{
			if(processMemory[i]<=memory[j])
			{
				flag=1;
				memory[j]-=processMemory[i];
				printf("\n%d\t\t%d",i+1,j+1);
				break;
			}
		}
		if(flag==0)
			return;
	}
}

void fnWorstFit(int memory[100])
{
	int i,j;
	printf("\nWorstFit\nProcess\t\tMemoryBlock");
```

```
	for(i=0;i<noProcesses;i++)
	{
		int high=-1;
		for(j=0;j<noMemoryBlock;j++)
		{
			if(processMemory[i]<=memory[j])
			{
				if(memory[high]<memory[j] || high==-1)
					high=j;
			}
		}
		if(high!=-1)
		{
			memory[high]-=processMemory[i];
			printf("\n%d\t\t%d",i+1,high+1);
		}
		else
		{
			printf("Cant allocate further");
			return;
		}
	}
}

void fnBestFit(int memory[100])
{
	int i,j;
	printf("\nBestFit\nProcess\t\tMemoryBlock");
	for(i=0;i<noProcesses;i++)
	{
		int low=-1;
		for(j=0;j<noMemoryBlock;j++)
		{
			if(processMemory[i]<=memory[j])
			{
				if(memory[low]>memory[j] || low==-1)
```

```
                                        low=j;
                        }
                }
                if(low!=-1)
                {
                        memory[low]-=processMemory[i];
                        printf("\n%d\t\t%d",i+1,low+1);
                }
                else
                {
                        printf("Cant allocate further");
                        return;
                }
        }
}

void restore()
{
        int i;
        for(i=0;i<noMemoryBlock;i++)
                memory[i]=tempMemory[i];
}

int main()
{
        int i,choice;
        printf("\nEnter the total number of memory blocks and number of requested processes: ");
        scanf("%d%d",&noMemoryBlock,&noProcesses);
        printf("\nEnter the size of memory block: \n");
        for(i=0;i<noMemoryBlock;i++)
                scanf("%d",&tempMemory[i]);
        printf("\nEnter the size of memory requested by process: \n");
        for(i=0;i<noProcesses;i++)
                scanf("%d",&processMemory[i]);
        while(1)
        {
```

```
                printf("\n\n1.FirstFit\n2.BestFit\n3.WorstFit\n4.Exit\nEnter your choice: ");
                scanf("%d",&choice);
                restore();
                switch(choice)
                {
                        case 1: fnFirstFit(memory);
                                break;

                        case 2: fnBestFit(memory);
                                break;

                        case 3: fnWorstFit(memory);
                                break;

                        case 4: exit(0);
                }
        }
}
```

OUTPUT:

```
Enter the total number of memory blocks and number of requested processes: 5 5

Enter the size of memory block:
200 300 400 500 600

Enter the size of memory requested by process:
400 300 120 80 212


1.FirstFit
2.BestFit
3.WorstFit
4.Exit
Enter your choice: 1

FirstFit
Process         MemoryBlock
1               3
2               2
3               1
4               1
5               4

1.FirstFit
2.BestFit
3.WorstFit
4.Exit
Enter your choice: 2

BestFit
Process         MemoryBlock
1               3
2               2
3               1
4               1
5               4
```

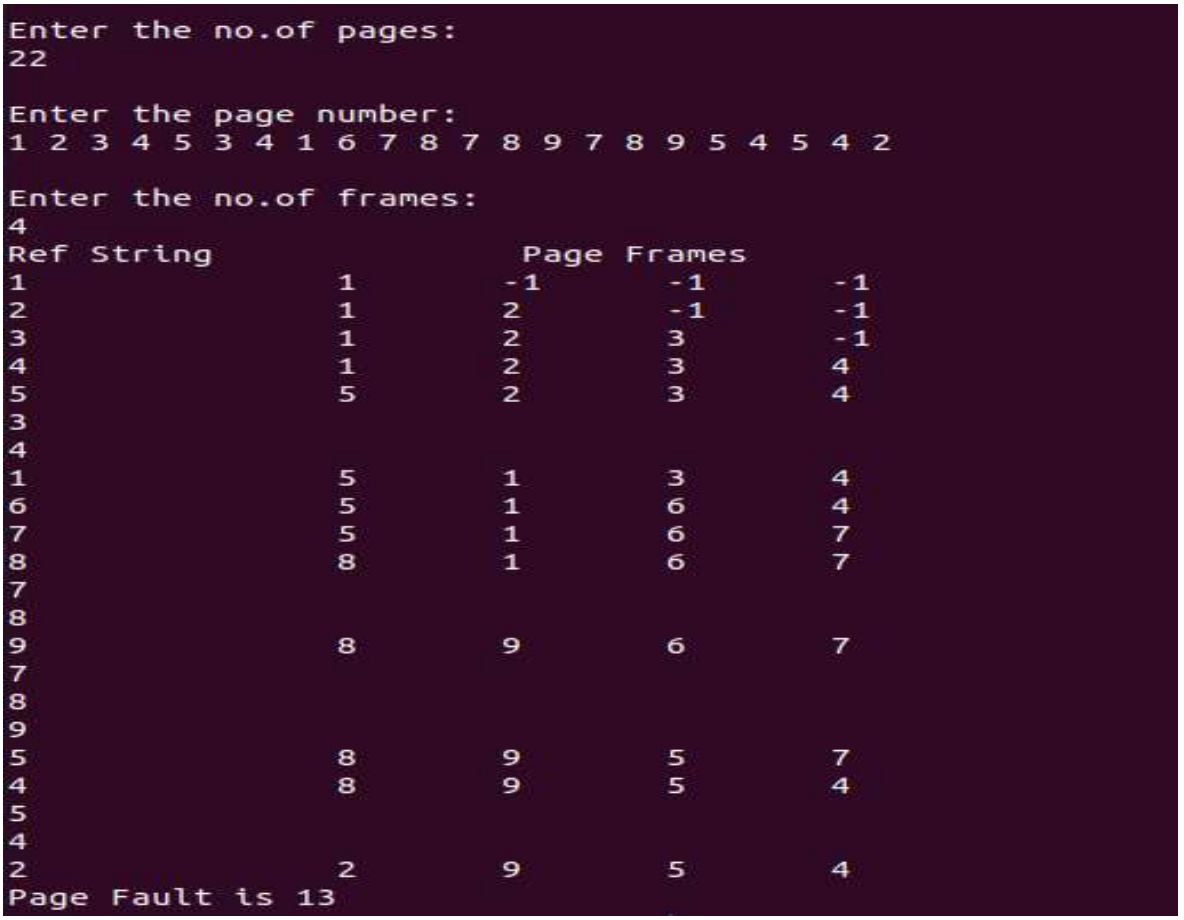
```
1.FirstFit
2.BestFit
3.WorstFit
4.Exit
Enter your choice: 3

WorstFit
Process         MemoryBlock
1               5
2               4
3               3
4               2
5               3

1.FirstFit
2.BestFit
3.WorstFit
4.Exit
Enter your choice: 4
```

6. Write a C program to implement the following page replacement algorithms:
a)FIFO

```
#include <stdio.h>
int main()
{
	int i,j,n,a[50],frame[10],no,k,avail,count=0;
	printf("\nEnter the no.of pages:\n");
	scanf("%d",&n);
	printf("\nEnter the page number:\n");
	for(i=1;i<=n;i++)
		scanf("%d",&a[i]);
	printf("\nEnter the no.of frames:\n");
	scanf("%d",&no);
	for(i=0;i<no;i++)
		frame[i]= -1;
	j=0;
	printf("Ref String\t\t Page Frames\n");
	for(i=1;i<=n;i++)
	{
		printf("%d\t\t",a[i]);
		avail=0;
		for(k=0;k<no;k++)
			if(frame[k]==a[i])
				avail=1;
		if(avail==0)
		{
			frame[j]=a[i];
			j=(j+1)%no;
			count++;
			for(k=0;k<no;k++)
				printf("%d\t",frame[k]);
		}
	printf("\n");
	}
	printf("\nPage Fault is %d\n",count);
```

```
    return 0;
}
```

OUTPUT:



b) LRU

```
#include <stdio.h>
#include <stdlib.h>

int findLRU(int time[],int n)
{
    int i,minimum=time[0],pos=0;
    for(i=1;i<n;++i)
    {
        if(time[i]<minimum)
        {
            minimum=time[i];
            pos=i;
        }
```

```
        }
        return pos;
}

int main()
{
        int no_of_frames,no_of_pages,frames[10],pages[30],counter=0,time[10],flag1,flag2,i,j,pos,faults=0;
        printf("Enter number of frames: ");
        scanf("%d",&no_of_frames);
        printf("Enter number of pages: ");
        scanf("%d",&no_of_pages);
        printf("Enter reference string: ");
        for(i=0;i<no_of_pages;++i)
        {
                scanf("%d",&pages[i]);
        }
        for(i=0;i<no_of_frames;++i)
        {
                frames[i]=-1;
        }
        for(i=0;i<no_of_pages;++i)
        {
                flag1=flag2=0;
                for(j=0;j<no_of_frames;++j)
                {
                        if(frames[j]==pages[i])
                        {
                                counter++;
                                time[j]=counter;
                                flag1=flag2=1;
                                break;
                        }
                }
                if(flag1==0)
                {
                        for(j=0;j<no_of_frames;++j)
```

```
                {
                        if(frames[j]==-1)
                        {
                                counter++;
                                faults++;
                                frames[j]=pages[i];
                                time[j]=counter;
                                flag2=1;
                                break;
                        }
                }
        }
        if(flag2==0)
        {
                pos=findLRU(time,no_of_frames);
                counter++;
                faults++;
                frames[pos]=pages[i];
                time[pos]=counter;
        }
        printf("\n");
        for(j=0;j<no_of_frames;++j)
        {
                printf("%d\t",frames[j]);
        }
    }
    printf("\n\nTotal Page Faults = %d\n",faults);
    return 0;
}
```

OUTPUT:

```
Enter number of frames: 3
Enter number of pages: 20
Enter reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7       -1      -1
7       0       -1
7       0       1
2       0       1
2       0       1
2       0       3
2       0       3
4       0       3
4       0       2
4       3       2
0       3       2
0       3       2
0       3       2
1       3       2
1       3       2
1       0       2
1       0       2
1       0       7
1       0       7
1       0       7

Total Page Faults = 12
```

c) LFU

```
#include <stdio.h>

#include <stdlib.h>


int main()

{

        int total_frames,total_pages,hit=0;

        int pages[25],frame[10],arr[25],time[25];

        int m,n,page,flag,k,minimum_time,temp;

        printf("Enter Total Number of Pages: ");

        scanf("%d",&total_pages);

        printf("Enter Total Number of Frames: ");

        scanf("%d",&total_frames);

        for(m=0;m<total_frames;m++)

        {

                frame[m]=-1;

        }
```

```
for(m=0;m<25;m++)
{
	arr[m]=0;
}
printf("Enter Values of Reference String: \n");
for(m=0;m<total_pages;m++)
{
	printf("Enter Value No.[%d]: ",m+1);
	scanf("%d",&pages[m]);
}
printf("\n");
for(m=0;m<total_pages;m++)
{
	arr[pages[m]]++;
	time[pages[m]]=m;
	flag=1;
	k=frame[0];
	for(n=0;n<total_frames;n++)
	{
		if(frame[n]==-1||frame[n]==pages[m])
		{
			if(frame[n]!=-1)
			{
				hit++;
			}
			flag=0;
			frame[n]=pages[m];
			break;
		}
		if(arr[k]>arr[frame[n]])
		{
			k=frame[n];
		}
	}
	if(flag)
	{
```

```
                        minimum_time=25;
                        for(n=0;n<total_frames;n++)
                        {
                                if(arr[frame[n]]==arr[k] && time[frame[n]]<minimum_time)
                                {
                                        temp=n;
                                        minimum_time=time[frame[n]];
                                }
                        }
                        arr[frame[temp]]=0;
                        frame[temp]=pages[m];
                }
                for(n=0;n<total_frames;n++)
                {
                        printf("%d\t",frame[n]);
                }
                printf("\n");
        }
        printf("Page Hit: %d\n",hit);
        return 0;
}
```

OUTPUT:

```
Enter Total Number of Pages:     20
Enter Total Number of Frames:    3
Enter Values of Reference String
Enter Value No.[1]:       7
Enter Value No.[2]:       0
Enter Value No.[3]:       1
Enter Value No.[4]:       2
Enter Value No.[5]:       0
Enter Value No.[6]:       3
Enter Value No.[7]:       0
Enter Value No.[8]:       4
Enter Value No.[9]:       2
Enter Value No.[10]:      3
Enter Value No.[11]:      0
Enter Value No.[12]:      3
Enter Value No.[13]:      2
Enter Value No.[14]:      1
Enter Value No.[15]:      2
Enter Value No.[16]:      0
Enter Value No.[17]:      1
Enter Value No.[18]:      7
Enter Value No.[19]:      0
Enter Value No.[20]:      1
```

```
7          -1         -1
7          0          -1
7          0          1
2          0          1
2          0          1
2          0          3
2          0          3
4          0          3
4          0          2
3          0          2
3          0          2
3          0          2
3          0          2
1          0          2
1          0          2
1          0          2
1          0          2
7          0          2
7          0          2
1          0          2
Page Hit:            9
```

PART B

9. Write a C program to compute FIRST of all Non Terminals of a given grammar.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
int n,m=0,p,i=0,j=0;
char a[10][10],f[10];
void first(char c);

int main()
{
        int i,z;
        char c,ch;
        printf("Enter the no of productions:\n");
        scanf("%d",&n);
        printf("Enter the productions:\n");
        for(i=0;i<n;i++)
                scanf("%s%c",a[i],&ch);
        do
        {
                m=0;
                printf("Enter the element whose first is to be found: ");
                scanf("%c",&c);
                first(c);
                printf("First(%c)={",c);
                for(i=0;i<m;i++)
                        printf("%c",f[i]);
                printf("}\n");
                strcpy(f," ");
                printf("Continue(0/1)?\n");
                scanf("%d%c",&z,&ch);
        }while(z==1);
        return(0);
```

```
}


void first(char c)
{
        int k;
        if(!isupper(c))
        f[m++]=c;
        for(k=0;k<n;k++)
        {
                if(a[k][0]==c)
                {
                        if(islower(a[k][2]))
                                f[m++]=a[k][2];
                        else
                                first(a[k][2]);
                }
        }
}
```

OUTPUT:

```
Enter the no of productions:
8
Enter the productions:
E=TD
D=+TD
D=#
T=FS
S=*FS
S=#
F=(E)
F=a
Enter the element whose first is to be found: E
First(E)={(a}
Continue(0/1)?
1
Enter the element whose first is to be found: T
First(T)={(a}
Continue(0/1)?
1
Enter the element whose first is to be found: D
First(D)={+#}
Continue(0/1)?
1
Enter the element whose first is to be found: F
First(F)={(a}
Continue(0/1)?
1
Enter the element whose first is to be found: S
First(S)={*#}
Continue(0/1)?
0
```

10. Write a C program to construct predictive parsing table for the given grammar.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

void addtononT(char);
void addtoter(char);
int nop,ppt[10][10];
char productions[10][10],ter[10],nonT[10],first[10][10],follow[10][10];

int main()
{
	int i,j,k,m,pos=0;
	for(i=0;i<10;i++)
		for(j=0;j<10;j++)
			ppt[i][j]=-1;
	printf("Enter the number of productions:");
	scanf("%d",&nop);
	printf("\nEnter productions in the form like E->E+T (Enter # for epsilon): \n");
	for(i=0;i<nop;i++)
	{
		printf("Enter production number %d:",i+1);
		scanf("%s",productions[i]);
	}
	for(i=0;i<nop;i++)
		addtononT(productions[i][0]);
	for(i=0;i<nop;i++)
		for(j=3;productions[i][j]!='\0';j++)
			if(islower(productions[i][j])||(!isalpha(productions[i][j])))
				addtoter(productions[i][j]);
	for(j=0;ter[j]!='\0';j++);
	ter[j]='$';
	ter[++j]='\0';
	printf("\nEnter first of all non terminals without any space b/w the symbols like abc# (#for epsilon): \n" );
	for(i=0;i<nop;i++)
```

```
{
        printf("Enter first of:");
        for(k=3;k<productions[i][k]!='\0';k++)
                printf("%c",productions[i][k]);
        printf("=");
        scanf("%s",first[i]);
        for(j=strlen(first[i]);j>=0;j--)
                first[i][j+1]=first[i][j];
        first[i][0]=productions[i][0];
}
printf("\nEnter follow of all non terminals without any space b/w symbols like abc# (# for epsilon): \n");
for(i=0;nont[i]!='\0';i++)
{
        printf("Enter follow of %c=",nont[i]);
        scanf("%s",follow[i]);
        for(j=strlen(follow[i]);j>=0;j--)
                follow[i][j+1]=follow[i][j];
        follow[i][0]=nont[i];
}
for(i=0;i<nop;i++)
{
        for(m=0;follow[m][0]!=first[i][0];m++);
        for(j=1;first[i][j]!='\0';j++)
        {
                if(first[i][j]!='#')
                {
                        for(k=0;ter[k]!='\0';k++)
                                if(ter[k]==first[i][j])
                                        break;
                        ppt[m][k]=i;
                }
                else
                {
                        for(m=0;follow[m][0]!=first[i][0];m++);
                        for(j=1;follow[m][j]!='\0';j++)
                        {
```

```
                                        for(k=0;ter[k]!='\0';k++)
                                                if(ter[k]==follow[m][j])
                                                        break;
                                        ppt[m][k]=i;
                                }
                        }
                }
                first[i][0]='0';
        }
        printf("\n\nPredictive parsing table\n");
        printf(".....Terminals.....\n");
        printf("Non Terminals |\t\t");
        for(i=0;ter[i]!='\0';i++)
                printf("%c\t",ter[i]);
        printf("\n");
        for(i=0;follow[i][0]!='\0';i++)
        {
                m=0;
                printf("%c\t\t",nont[i]);
                for(j=0;ter[j]!='\0';j++)
                {
                        pos=ppt[i][j];
                        for(;m<=j;m++)
                                printf("\t");
                        if(pos!=-1)
                                printf("%s",productions[pos]);
                }
                printf("\n");
        }
        return 0;
}

void addtonont(char c)
{
        int j;
        for(j=0;nont[j]!='\0';j++)
```

```
                if(nont[j]==c)
                        return;
        nont[j]=c;
        nont[j+1]='\0';
}

void addtoter(char c)
{
        int j;
        for(j=0;ter[j]!='\0';j++)
                if(ter[j]==c)
                        return;
        if(c!='#')
        {
                ter[j]=c;
                ter[j+1]='\0';
        }
}
```

OUTPUT:

```
Enter the number of productions:8

Enter productions in the form like E->E+T (Enter # for epsilon):
Enter production number 1:E->TX
Enter production number 2:X->+TX
Enter production number 3:X->#
Enter production number 4:T->FY
Enter production number 5:Y->*FY
Enter production number 6:Y->#
Enter production number 7:F->i
Enter production number 8:F->(E)

Enter first of all non terminals without any space b/w the symbols like abc# (#for epsilon):
Enter first of:TX=+(
Enter first of:+TX=+
Enter first of:#=#
Enter first of:FY=i(
Enter first of:*FY=*
Enter first of:#=#
Enter first of:i=i
Enter first of:(E)=(

Enter follow of all non terminals without any space b/w symbols like abc# (# for epsilon):
Enter follow of E=)$
Enter follow of X=)$
Enter follow of T=+)$
Enter follow of Y=+$)
Enter follow of F=+$)*


Predictive parsing table
.....................Terminals.....................
Non Terminals |        +       *       i       (       )       $
E                      E->TX                   E->TX
X                      X->+TX                          X->#    X->#
T                                      T->FY   T->FY
Y                      Y->#    Y->*FY                  Y->#    Y->#
F                                      F->i    F->(E)
```

11. Write a C program to implement recursive descent parsing for the given grammar.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

void nonterminal(char);
int noofproduction,k,temp;
char productionset[20][20];
char str[20];
int result;

int main()
{
	int i,ch;
	printf("\nEnter the no of productions: \n");
	scanf("%d",&noofproduction);
	printf("\nEnter the productions in form like E->E+T (Enter # for the epsilon): \n");
	for(i=0;i<noofproduction;i++)
	{
		printf("Enter the production number %d:",i+1);
		scanf("%s",productionset[i]);
	}
	do
	{
		k=0;
		printf("\nEnter the string\n");
		scanf("%s",str);
		nonterminal(productionset[0][0]);
		if(k==strlen(str))
			printf("\nInput string is valid.\n");
		else
			printf("\nInput string is invalid.\n");
		printf("\nDo you want to continue(0/1)?\n");
		scanf("%d",&ch);
	}while(ch==1);
```

```
	return 0;
}

void nonterminal(char p)
{
	int i,j,found=0;
	for(i=0;i<noofproduction;i++)
	{
		temp=k;
		if(productionset[i][0]==p)
		{
			for(j=3;productionset[i][j]!='\0';j++)
			{
				if(isupper(productionset[i][j]))
				{
					found=1;
					nonterminal(productionset[i][j]);
				}
				else if(productionset[i][j]==str[k])
				{
					k++;
					found=1;
				}
				else if(productionset[i][j]=='#')
				{
					found=1;
					return;
				}
				else
				{
					k=temp;
					break;
				}
			}
		}
	}
```

```
        if(i>=noofproduction && found==0 && k!=strlen(str))
        {
                printf("Invalid Input.\n");
                exit(0);
        }
}
```

OUTPUT:

```
Enter the no of productions:
3

Enter the productions in form like E->E+T (Enter # for the epsilon):
Enter the production number 1:S->cAd
Enter the production number 2:A->ab
Enter the production number 3:A->a

Enter the string
cad

Input string is valid.

Do you want to continue(0/1)?
1

Enter the string
cabd

Input string is valid.

Do you want to continue(0/1)?
1

Enter the string
abcd
Invalid Input.
```

12. Write a C program to construct the closure of an LR(0) item with respect to the given grammar.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
void closure(char[]);;
char citem[10][10],gram[10][10];
int nop,noi;
int main()
{
   int i,ch;
   char agram[10][10],item[10];
   printf("how many no of production:\n");
   scanf("%d",&nop);
   printf("\nenter production like this eg:E->E+T enter # for epsilon\n");
   for(i=0;i<nop;i++)
   {
      printf("enter production no %d:",i+1);
      scanf("%s",gram[i]);
   }
   agram[0][0]=gram[0][0];
   agram[0][1]='!';
   agram[0][2]='-';
   agram[0][3]='>';
   agram[0][4]=gram[0][0];
   agram[0][5]='\0';
   for(i=1;i<=nop;i++)
   strcpy(agram[i],gram[i-1]);
   printf("\naugmented is:\n");
   for(i=0;i<=nop;i++)
   printf("%s",agram[i]);
   do
   {
      printf("\nenter the item to find the closure\n");
      scanf("%s",item);
      printf("closure of %s={",item);
```

```
        closure(item);
        for(i=0;i<noi;i++)
        printf("%s",citem[i]);
        printf("}\n");
        printf("\nenter 1:to continue 2.stop\n");
        scanf("%d",&ch);
    }while(ch==1);
    return 0;
}
void closure(char it[20])
{
    int i,j,k=0,found;
    char temp[10];
    noi=0;
    strcpy(citem[k],it);
    noi++;
    while(k<noi)
    {
        i=0;
        while(it[i]!='\0'&&it[i]!='.')
        i++;
        if(i<(strlen(it)-1))
        {
            for(j=0;j<nop;j++)
            {
                found=0;
                if(it[i+1]==gram[j][0]&&isupper(it[i+1]))
                {
                    strcpy(temp,gram[j]);
                    for(i=strlen(temp);i>=3;i--)
                    temp[i+1]=temp[i];
                    temp[i+1]='.';
                    for(i=0;i<noi;i++)
                    if(strcmp(citem[i],temp)==0)
                    found=1;
                    if(found==0)
```

```
            {
               strcpy(citem[i],temp);
               noi++;
            }
         }
      }
      else
      {
         printf("%s }",it);
         exit(0);
      }
      k++;
      strcpy(it,citem[k]);
   }
   return;
}
```

OUTPUT:

```
Enter the no.of productions:
3

Enter production in the form of E->E+T( Enter # for epsilon):
Enter production no 1:S->AA
Enter production no 2:
A->aA
Enter production no 3:A->b

Augmented grammar is:
S!->S S->AA A->aA A->b
Enter the item to find the closure:
S->.S
closure of S->.S={  S->.S  S->.AA  A->.aA  }

Enter 1 to continue and 2 to stop:
1

Enter the item to find the closure:
S->.AA
closure of S->.AA={  S->.AA  A->.aA  }

Enter 1 to continue and 2 to stop:
2
```

****