|  |
| --- |
| WEEK -6 PROGRAMS |
| Done by:- |
|  |
| *19131A05P4 19131A05P8*  *19131A05P5 19131A05P9*  *19131A05P6 19131A05Q0*  *19131A05P7 19131A05Q1* |
|  |
|  |

**AIM:**

Write a program that implement factorial in the stack applications

**ALGORITHM:**

Start

Step 1: declare

functions push (val), pop(), display ().

Step 2:read a number 'a'.

Step 3: if (a==0) print 1

               Otherwise go to step 4.

Step 4: initialise i=a.

               Repeat the following steps until i<=0.

                4.1: push() ;

                4.2: i--;

Step 5: initialise i=a.

               Repeat the following steps until i<=1.

               5.1: x= pop ();

               5.2: y= pop ();

               5.3: val= x\*y;

               5.4: push(val);

               5.5: i--;

Step 6: call the function display();

Stop.

Push function: - void push (int val)

Start

Step1: increment top i.e

top++;

Step 2:  stack [top]= val;

Stop.

Pop function: -  int pop()

Start

Step 1: a = stack[top];

Step 2: decrement top i.e top--;

Step 3: return a.

Stop.

Display function: - void display()

Start

Step 1: initialise i=top.

Step 2: repeat the following steps until i<0.

              2.1: print stack[i].

              2.2: i--;

Stop.

**THEORY:**

Stacks have push and pop operations. Push adds a new item to the top of the stack and pop removes the item from the top of the stack and returns it.

The most fundamental use of stacks relates to how functions are called. One function can call another function which can call a third and so on to any depth until there is no more stack left to support further calls.

When a call is made one method of passing arguments to a function is to push the data onto the stack along with the address of the program statement execution must return to when the function has completed its work. This stack is referred to as the call stack or sometimes the machine stack. A specific CPU register, the stack pointer (SP), may be

provided to manage it. In this case a call function instruction says: push the return onto the stack and jump to the function address

making that part of the pushing automatic.

**PROGRAM:**

#include <iostream>

using namespace std;

int stack[100], n=100, top=-1;

void push(int val) {

      top++;

      stack[top]=val;

}

int pop() {

   int a;

      a = stack[top];

      top--;

   return a;

}

void display()

{

      cout<<stack[top];

}

int main()

{

   int a,i,val,x,y;

   cout<<"enter a number"<<endl;

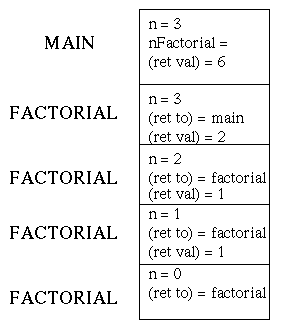
   cin>>a; //no whose factorial is to be found

   if(a==0)

   cout<<"1"<<endl;

   else

   {



|  |  |  |
| --- | --- | --- |
| 1 |  |  |
| 2 | 2 |  |
| 3 | 3 | 6 |

x = pop(); = 1 2

   y = pop(); = 2 3

   val = x\*y; = 2 6

 push(val); = 2 6

   for(i=a;i>0;i--)

   push(i);

   for(i=a;i>1;i--)

   {

   x = pop();

   y = pop();

   val = x\*y;

   push(val);

   }

   display();

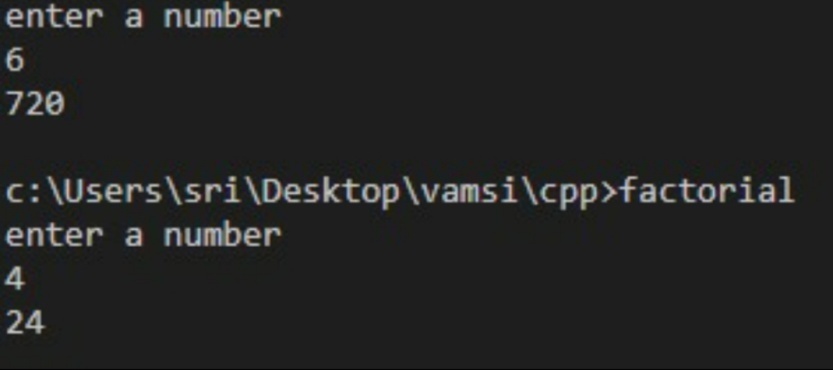
}

   return 0;

}

**OBSERVATIONS:**

|  |  |  |
| --- | --- | --- |
| INPUT | EXPECTED OUTPUT | FINAL OUTPUT |
| 6 | 720 | 720 |
| 4 | 24 | 24 |



AIM:

*Write a program to implement Evaluation of postfix expression in Stack applications*

ALGORITHM:

*Start*

STEP 1:*initialize and declare the required variables of respective*

*data types and formats.*

STEP 2: *give the postfix expression as an input.*

STEP 3:*the given string or the line from the input stream is read and*

*the given steps are followed:*

STEP 4:*check whether each element of the given string is an*

*operand or an operator.*

STEP 5:*If(element == operand),Goto step7.*

STEP 6:*Else if(element==operator)Goto step 8.*

STEP 7:*Push the value onto the stack*

STEP 8:*Pop the top 2 values from the stack using pop().*

*Perform the operation.*

*Push the result into the stack using push().*

*Exit.*

*Stop*

THEORY:

*Stack is a linear data structure, that stores elements in an ordered manner by following “LIFO” (last in first out) order. The elements are added and removed from the top.*

*Operations that can be performed on stack:*

*Push(): adds an element at the top of the stack.*

*Pop() : removes or deletes an element from the top of the stack.*

*Peek(): returns the value of the topmost element in the stack.*

*An****expression****is called the****postfix expression****if the operator appears in the****expression****after the operands*

*In postfix evaluation, every character of the postfix expression is scanned from left to right. If the character encountered is an operand, it is pushed onto the stack. If an operator is encountered then the top 2 values are popped from the stack and the operator is applied on these values. The result is pushed onto the stack.*

PROGRAM:

*#include <iostream>*

*#include<string>*

*using namespace std;*

*int main()*

*{*

*int a,x1,x2,x3,x4,i=0;*

*stack <int> s1;*

*string str,t;*

*char e;*

*cout<<"Enter a postfix Expression "<<endl;*

*getline(cin,str);*

*cout<<str<<endl;*

*stringstream X(str);*

*while(getline(X,t,' '))*

*{*

*stringstream Y(t);*

*Y>>a;*

*stringstream Z(t);*

*Z>>e;*

*if(a==0 && e=='0')*

*{*

*s1.push(a);*

*cout<<"push value"<<a<<endl;*

*}*

*else if(a)*

*{*

*s1.push(a);*

*cout<<"push value"<<a<<endl;*

*}*

*else*

*{*

*x1=s1.top();*

*cout<<"Value on top"<<x1<<"going to pop"<<endl;*

*s1.pop();*

*x2=s1.top();*

*cout<<"value on top"<<x2<<"going to pop"<<endl;*

*s1.pop();*

*switch(e)*

*{*

*case '+': x3=x1+x2;*

*cout<<"push value"<<x3<<endl;*

*s1.push(x3);*

*break;*

*case '-': x3=x2-x1;*

*s1.push(x3);*

*cout<<"push value"<<x3<<endl;*

*break;*

*case '\*': x3=x2\*x1;*

*s1.push(x3);*

*cout<<"push value"<<x3<<endl;*

*break;*

*case '/': x3=x2/x1;*

*s1.push(x3);*

*cout<<"push value"<<x3<<endl;*

*break;*

*}*

*}*

*}*

*cout<<"ans is"<<s1.top()<<endl;*

*}*

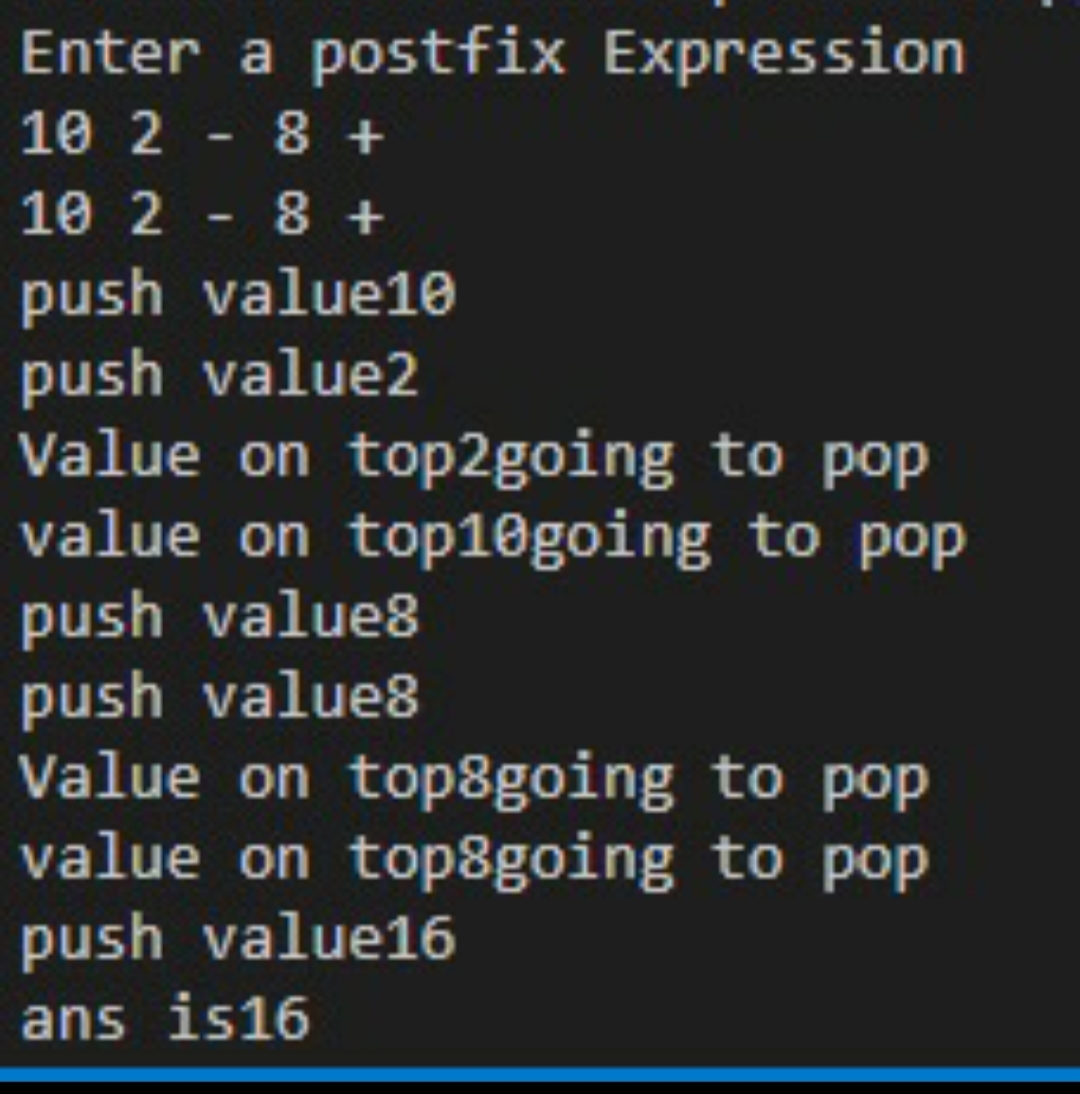
THEORETICAL OUTPUT:

INPUT: 10 2 – 8 +

|  |  |  |
| --- | --- | --- |
| CHARCTER | STACK | OPERATION |
| 10 | 10 |  |
| 2 | 10 , 2 |  |
| - | 8 | 10-2=8 |
| 8 | 8 , 8 |  |
| + | 16 | 8+8 = 16 |

OUTPUT: 16

PRACTICAL OUTPUT:



OBSERVATIONS:

|  |  |  |
| --- | --- | --- |
| Input postfix expression | Expected output | Final output |
| *10 2 – 8 +* | *16* | *16* |

AIM:

*Write a program to implement number conversion in Stack applications.*

ALGORITHM:

***Start***

STEP 1:*Declare stack <int> s and Read a number n.*

STEP 2:*Print the choices as follows.*

*1. Decimal to binary*

*2. Decimal to ternary*

*3. Decimal to octa*

STEP 3:*Read the choice number 'a'.*

STEP 4*: if(a==1) then assign b=2 and break.*

*else if(a==2) then assign b=3 & break.*

*else if(a==3) then assign b=8 & break.*

STEP 5:*repeat the following steps while (n>0)*

*5.1: r=n%b ;*

*5.2: n=n/b ;*

*5.3: s.push(r) ;*

*5.4: c++;*

STEP 6:*repeat the following steps while (c>0)*

*6.1: print the value of s.top()*

*6.2: s.pop() ;*

*6.3: c-- ;*

***Stop.***

THEORY:

*Step 1:*

*Initially Top = -1 .*

*It is used to create empty Stack.*

*Step 2 :*

*Accept Decimal Number*

*Step 3:*

*Now we are dividing the original number by 2 and remainder is pushed onto stack , In the Second Iteration again we are dividing num by 2 and remainder is pushed onto stack . This action is carried out till number becomes 0.*

*Step 4 :*

*Push Remainder Onto Stack if Stack is not Full and number != 0*

*When number becomes 0 i.e Do not push elements onto stack.*

PROGRAM:

*#include<bits/stdc++.h>*

*using namespace std;*

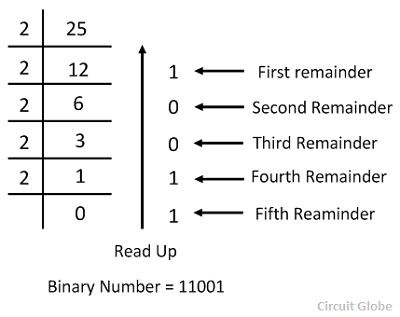
*int main()*

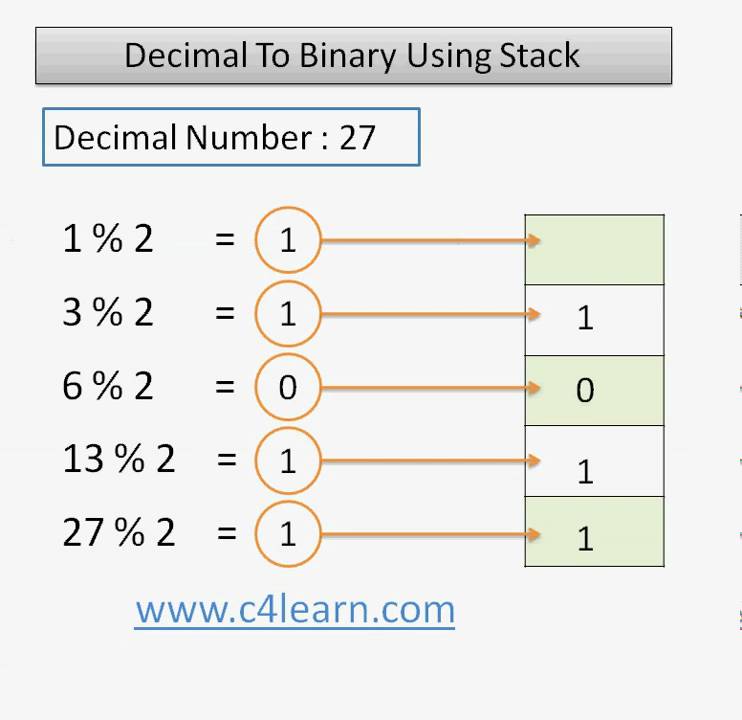
*{ int n,a,b,r,c=0,i=0;*

*stack <int> s;*

|  |
| --- |
| *1* |
| *1* |
| *0* |
| *0* |
| *1* |

Pop operation gives 11001





*cout<<"enter a number"<<endl;*

*cin>>n;*

*cout<<"1.decimal to Binary"<<endl;*

*cout<<"2.decimal to Ternary"<<endl;*

*cout<<"3.decimal to Octa"<<endl;*

*cout<<"Enter your choice"<<endl;*

*cin>>a;*

*switch(a){*

*case 1 : b =2;*

*break;*

*case 2 : b =3;*

*break;*

*case 3 : b =8;*

*}*

*while(n>0)*

*{*

*r = n%b;*

*n = n/b;*

*s.push(r);*

*c++;*

*}*

*while(c>0)*

*{*

*cout<<s.top();*

*s.pop();*

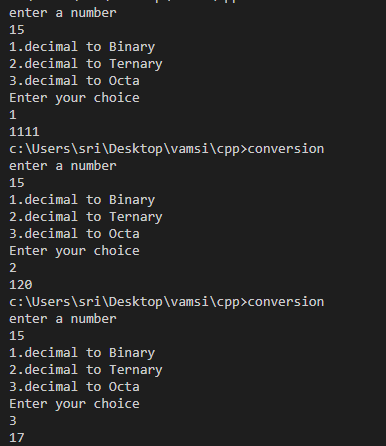
*c--;*

*}*

*}*

OBSERVATION:

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | INPUT | EXPECTED OUTPUT | ACTUAL OUTPUT |
| *1* | *15*  *1* | *1111* | *1111* |
| *2* | *15*  *2* | *120* | *120* |
| *3* | *15*  *3* | *17* | *17* |

****