**GANPAT UNIVERSITY**

**U.V. PATEL COLLEGE OF ENGINEERING**



**Lab File**

2CEIT304: Data Structures

B.Tech Semester: III

Computer Engineering/ Information Technology

Academic Year: 2020 (Odd Sem.)

**Prepared By:**

**Name**

**Er NO**

**Department of Computer Science & Engineering**

**UVPCE Ganpat University**

**LISTOF EXPERIMENTS**

|  |  |  |
| --- | --- | --- |
| Sr. No | List | Date |
| 1 | Write a C Program to Implement following :   1. To traverse elements of an array. 2. To calculate the factorial of a number using recursion. 3. To find the address of a variable using a pointer. 4. To count the length of the string. (Do not use strlen( ) ) 5. To reverse the string. (Do not use strrev( ) ). 6. To count the number of a particular character entered by a user. 7. To count & display all the vowels. 8. To change the case of the string. (Lower to Upper & Upper to Lower). 9. To concatenate two strings. 10. To compare two strings. |  |
| 2 | Implement function of stack with following operations:   1. Push( ) 2. Pop( ) 3. Peek( ) 4. Display( ) 5. Isempty( ) 6. Isfull( ) |  |
| 3 | Implement  Applications of Stack.   1. Write a program to recognize the string with language L={wcwR / w takes multiple occurrences of {a,b}}. 2. Write a program to check the validity of expressions, which contains multiple opening and closing brackets. (i.e., [{(a+b) \* c} – d]). 3. Write a program to convert unparenthized and parenthized infix expressions to postfix. 4. Write a program to evaluate the given postfix expression. |  |

**Practical 1**

**Aim**: Implement  Applications of Stack.

1. **Write a program to recognize the string with language L={wcwR / w takes multiple occurrences of {a,b}}.**

**Pseudo code/Algorithm:**

|  |
| --- |
| 1. **[ get the string from the user and initialize the index variable to get character one by one from the string]**   READ(str <-0  PUSH(S,top,’#’)   1. **[ fetch one character at a time and push on the stack until separator C occur in string] . repeat while str[i] not ‘C’**   Push(S,tp, str[i]) i <- i+1   1. **[scan characters followingthe ‘C’ one at a time and pop character from the stack and compare]**   i < i +1  repeat while s[i] notNULL x <- POP (S , top)  if X NOT EQUAL to S[i]  then  Write ( “invalid String”) goto step 5   1. **[ compare the top and end of string simulteneously if step 3 is successful] . if S[i]= NULL and S[top] = ‘#’**   then  write ( “String is valid”) else write ( “invalid String”)   1. **[finished the algorithm]**   Exit |

**C Program:**

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  #include<string.h>  void push(char s[],int n,int \*top,char x)  {      if(\*top>=n-1)      {          printf("stack is overflow.\n");      }      else      {          \*top=\*top+1;          s[\*top]=x;      }  }  char pop(char s[],int \*top)  {      if(\*top==-1)      {      return '~';      }      else      {        \*top=\*top-1;      return(s[\*top+1]);        }  }  int chk\_lang(char a[])  {      int i,top=-1,n;      char s[80],c;      n=strlen(a);      for(i=0;a[i]!='C'&&a[i]!='c';i++)      {          if((a[i]>='a'&&a[i]<='z')||(a[i]>='A'&&a[i]<='Z'))            push(s,n,&top,a[i]);          else          return 0;      }      i++;      while(a[i]!='\0')      {          c=pop(s,&top);          if(c==a[i])          i++;          else          return 0;      }      if(top==-1)      return 1;      else      return 0;  }  void main()  {      char a[80];      clrscr();      printf("enter a string : ");      gets(a);      if(chk\_lang(a))          printf("String is valid");      else          printf("Unvalid String ");      getch();  } |

**Output:**

|  |
| --- |
|  |

1. **Write a program to check the validity of expressions, which contains multiple opening and closing brackets. (i.e., [{(a+b) \* c} – d]).**

**Pseudo code/Algorithm:**

|  |
| --- |
| **1.[Read the Expression from the user. check is it valid or not ?]**  READ (S )  I <- 0  SPUSH (S,TOP, ’#’)  X <-CHK\_EXP(S)  If X=0  Then  Write(“invalid Expresssion ”);  Exit  **2. [ Read one character from the string and do accordingly]**  Repeat while s[i] not NULL  if s[i] = ‘(‘ Then SPUSH(S ,top,s[i]) Else if s[i]= “) ”  Then  C <- SPOP(S, TOP)  if C NOT EQUAL to ‘(‘  Then  write ( “string is not valid”) EXIT  Else continue   1. **[ Check the termination conditions]**   if s[i] = NULL and s[Top] = “#” then  Write (“Valid string”)  else Write ( “Invalid String”)   1. **finish algorithms]**   Exit |

**C Program:**

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  #include<string.h>  void push(char s[],int n, int \*top,char x)  {      if(\*top>=(n-1))      {          printf("stack overflow");          return;      }      \*top=\*top+1;      s[\*top]=x;  }  char pop(char s[],int \*top)  {      if(\*top==-1)          return 'n';      \*top=\*top-1;      return(s[\*top+1]);  }  int isopd(char c)  {      if((c>='a'&&c<='z')||(c>='A'&&c<='Z')||(c>='0'&&c<='9'))          return 1;      return 0;  }  int isopr(char c)  {      switch(c)      {          case '+':case '-':case '\*':case '/':case '%':              return 1;      }      return 0;  }  int iscbrac(char c)  {      switch(c)      {          case')':case'}':case']':              return 1;      }      return 0;  }  int chk\_exp\_mxd(char a[])  {      int i,n,top=-1;      char s[20],c;      n=strlen(a);      for(i=0;i<(n-1);i++)      {          if(isopd(a[i])&&isopd(a[i+1])||isopr(a[i])&&isopr(a[i+1])||  isopr(a[i])&&a[i+1]==')'||isopr(a[n-1]))              return 0;      }      for(i=0;i<n;i++)      {          if(a[i]=='('||a[i]=='{'||a[i]=='[')              push(s,n,&top,a[i]);          else if(a[i]==')'||a[i]=='}'||a[i]==']')              c=pop(s,&top);      }      if(top==-1)          return 1;      else          return 0;  }  void main()  {      char a[80];      clrscr();      printf("\n enter expression :\n");      gets(a);      if(chk\_exp\_mxd(a))          printf("expression is vaild");      else          printf("expression isn't valid");      getch();  } |

**3**

**Output:**

|  |
| --- |
|  |

1. **Write a program to convert unparenthized and parenthized infix expressions to postfix.**

**Pseudo code/Algorithm:**

|  |
| --- |
| 1. **Operand**   Append to end of output expression.   1. **Operator ^**   Push ^ onto stack .   1. **Operators +, -, \*, /**   Pop from stack, append to output.  Until stack empty or top operator has lower precedence than new operator. Then push new operator onto stack.  If (current operator < upcoming operator) Push(upcoming operator)  Else if(current operator >= upcoming operator) Repeatative Pop(current operator)  Else  Print(invalid) |

**C Program:**

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  #include<string.h>  void push(char s[],int n,int \*top,char x)  {      if(\*top>=(n-1))      {          printf("stack overflow");          return;      }      \*top=\*top+1;      s[\*top]=x;  }  char pop(char s[],int \*top)  {      if(\*top==-1)          return '~';      \*top=\*top-1;      return(s[\*top+1]);  }  int isopd(char c)  {      if((c>='a'&&c<='z')||(c>='A'&&c<='Z')||(c>='0'&&c<='9'))          return 1;      return 0;  }  int isopr(char c)  {      switch(c)      {      case '+':case '-':case '\*':case '/':case '%':          return 1;      }      return 0;  }  int iprec(char c)  {      switch(c)      {      case '+': case '-':          return 2;      case '\*': case '/': case '%':          return 4;      case '(':          return 8;      case '#':          return 0;      default :          return -1;      }  }  int sprec(char c)  {      switch(c)      {      case  '+': case '-':          return 2;      case '\*': case '/': case '%':           return 4;      case '(':          return 1;      case '#':          return 0;      default :          return -1;      }  }  void infix2postfix(char a[])  {      char s[80],c,pstfx[80];      int i,n,k=0,top=0;      n=strlen(a);      s[top]='#';      for(i=0;i<n;i++)      {          if(isopd(a[i]))              pstfx[k++]=a[i];          else if(isopr(a[i])||a[i]=='(')          {              while(iprec(a[i])<=sprec(s[top]))                      pstfx[k++]=pop(s,&top);              push(s,n,&top,a[i]);          }          else if(a[i]==')')              while((c=pop(s,&top))!='(')                    pstfx[k++]=c;      }      while(s[top]!='#')      pstfx[k++]=pop(s,&top);      pstfx[k]='\0';      printf("\ninfix   :%s\n",a);      printf("postfix   :%s",pstfx);  }  void main()  {      char a[80];      clrscr();      printf("Enter expression:\n");      scanf("%s",a);      infix2postfix(a);      getch();  } |

**Output:**

|  |
| --- |
|  |

1. **Write a program to evaluate the given postfix expression.**

**Pseudo code/Algorithm:**

|  |
| --- |
| 1. **Operand**   Append to end of output expression.   1. **Operator ^**   Push ^ onto stack .   1. **Operators +, -, \*, /**   Pop from stack, append to output.  Until stack empty or top operator has lower precedence than new operator. Then push new operator onto stack.   1. **Open parenthesis**   Push (onto stack)   1. **Close parenthesis**   Pop operators from stack and append to output.  Until open parenthesis is popped.  Discard both parentheses. FOR EVALUATE THE POSTFIX EXPRESSION. While ( not end of input)symbol = input symbol  if (symbol is operand)  push(operand\_stack, symbol)  else  op2 = pop(operand\_stack) op1 = pop(operand\_stack)  result = apply symbol to operand1 and operand2 push(operand\_stack, result)  return  pop(operand\_stack) |

**C Program:**

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  #include<string.h>  #define size 30  void spush(char s[], int \*top, char ch)  {      \*top=\*top+1;      s[\*top]=ch;  }  char spop(char s[], int \*top)  {      char ch='\0';      ch=s[\*top];      \*top=\*top-1;      return ch;  }  void push(int s[], int \*top, int x)  {      \*top=\*top+1;      s[\*top]=x;  }  int pop(int s[], int \*top)  {      int x;      x=s[\*top];      \*top=\*top-1;      return x;  }  int precedence(char ch)  {      if(ch=='/'||ch=='\*'||ch=='%')          return 8;      else if(ch=='+'||ch=='-')          return 4;      else if(ch=='(')          return 2;      else          return 0;  }  int isopd(char ch)  {      if( (ch>=65 &&ch<=90) || (ch>=97 &&ch<=122) || (ch>=48 &&ch<=57) )          return 1;      else          return 0;  }  int isopt(char ch)  {      if( ch=='+' || ch=='-' || ch=='\*' || ch=='/' || ch=='%' )          return 1;      else          return 0;  }  int checkExp(char ch[])  {      int i,l;      l=strlen(ch);      for(i=0;i<l;i++)      {          if( (isopd(ch[i]) &&isopd(ch[i+1])) ||              (isopt(ch[i]) &&isopt(ch[i+1])) ||                  (isopd(ch[i]) &&ch[i+1]=='(') ||                      (isopt(ch[i]) &&ch[i+1]==')') ||              isopt(ch[l-1]) )              return 0;      }      return 1;  }  int checkPar(char ch[])  {      char s[size],temp;      int i,\*top;      \*top=-1;      spush(s,top,'#');      for(i=0;ch[i]!='\0';i++)      {          if( ch[i]=='(' || ch[i]=='[' || ch[i]=='{' )              spush(s,top,ch[i]);          else if(ch[i]==')')          {              temp=spop(s,top);              if(temp!='(')                  return 0;              else                  continue;          }          else              continue;      }      if(s[\*top]=='#' &&ch[i]=='\0')          return 1;      else          return 0;  }  void convert(char infix[],char postfix[])  {      int i,x,j=0;      int \*top;      char s[size];      \*top=-1;      spush(s,top,'#');      for(i=0;infix[i]!='\0';i++)      {          x=infix[i];          if(isopd(x))          {              postfix[j]=x;              j++;          printf("\nPostfix currently is %s",postfix);          }          else          {              if(x=='(')                  spush(s,top,x);              else if(x==')')              {                  while(s[\*top]!='(')                  {                      postfix[j]=spop(s,top);                      j++;                  printf("\nPostfix currently is %s",postfix);                  }                  spop(s,top);              }              else              {                  if(precedence(x)>precedence(s[\*top]))                      spush(s,top,x);                  else                  {                      while(precedence(x)<=precedence(s[\*top]))                      {                          postfix[j]=spop(s,top);                          j++;                      printf("\nPostfix currently is %s",postfix);                      }                      spush(s,top,x);                  }              }          }      }      while(s[\*top]!='#')      {          postfix[j]=spop(s,top);          j++;      }      postfix[j]='\0';  }  int evaluate(char expr[])  {      int opd1,opd2,result,i;      char current;      int s[size];      int \*top;      \*top=-1;      for(i=0;expr[i]!='\0';i++)      {          current=expr[i];          if(isopd(current))          {              current=current-'0';              push(s,top,current);          }          else          {              opd2=pop(s,top);              opd1=pop(s,top);          printf("\n%d %d",opd1,opd2);              switch(current)              {                  case '+':                      result=opd1+opd2;                      break;                  case '-':                      result=opd1-opd2;                      break;                  case '\*':                      result=opd1\*opd2;                      break;                  case '/':                      result=opd1/opd2;                      break;                  case '%':                      result=opd1%opd2;                      break;              }              push(s,top,result);          }      }      result=pop(s,top);      return result;  }  void main()  {      int result;      char infix[size],postfix[size]="";      clrscr();      printf("Enter the expression:\n");      gets(infix);      if( checkExp(infix) &&checkPar(infix) )      {          printf("\nEntered expression is valid");          convert(infix,postfix);          printf("\nPostfix expression for corresponding infix expression is: ");          puts(postfix);          result=evaluate(postfix);          printf("\nFinal result after evaluation of postfix: %d", result);      }      else          printf("\nEntered expression is not valid");      getch();  } |

**Output:**

1. **Write a program to show the moving steps of the hanoi.**

**Pseudo code/Algorithm:**

|  |
| --- |
| MOVE(N,ORG,INT,DES)  If N >0  then  MOVE(N-1,ORG,DES,INT)  ORG-> DES (MOVE from ORG to DES)  MOVE(N-1,INT,ORG,DES)  end if |

**C Program:**

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  void hanoi(int n, char s, char m, char d)  {      if(n!=0)      {          hanoi(n-1,s,d,m);          printf("\nMove %d from %c to %c",n,s,d);          hanoi(n-1,m,s,d);      }  }  void main()  {      int n=3;      char s='a';      char d='c';      char m='b';      clrscr();      printf("\ntower of hanoi : \n");      hanoi(n,s,m,d);      getch();  } |

**Output:**

|  |
| --- |
|  |