SISTER NIVEDITA UNIVERSITY

COMPILER DESIGN

SUBMITTED BY: PIYUSH CHANDRA CHANDRA

DEPT- BTECH (CSE), ROLL- 1027

SUBMITTED TO: SOUMYAJIT PAL

ASSIGNMENT 1

19/12/2020 – 31/12/2020

1. **WAPC** to read the contents of a text file called my_file and display it on the terminal

➢
```c
#include <stdio.h>

int main()
{
    char in_name[80];
    FILE *in_file;
    int ch;

    printf("Enter file name:\n");
    scanf("%s", in_name);

    in_file = fopen(in_name, "r");

    if (in_file == NULL)
    {
        printf("Can't open %s for reading.\n", in_name);
    }
    else
    {
        while ((ch = fgetc(in_file)) != EOF)
        {
            printf("%c", ch);
        }
        fclose(in_file);
    }
    return 0;
}
```

```
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc file.c -o file } ; if ($?) { .\file }
Enter file name:
file.txt
Hello, I am Piyush Chandra
B.Tech CSE
```

2. **WAPC** to read the contents of a text file called my_file and display the count of vowels in the file

```c
#include<stdio.h>
int main()
{
   FILE *fptr;
   char c;
   int count=0;
   fptr=fopen("myfile.txt","r");
   while ((c=getc(fptr))!=EOF)
   {
      //putchar(c);
      printf("%c",c);
      if(c=='a'||c=='e'||c=='i'||c=='o'||c=='u')
      {
         count++;
      }

   }
   printf("\n The number of vowels are:%d",count);
   fclose(fptr);
   return 0;

}
```

```
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc fileHandling1.c -o fileHandling1 } ; if ($?) { .\fileHandling1 }
Hello I am Piyush Chandra Chandra!
Studying B.Tech CSE
Compiler Design


 The number of vowels are:17
```

3. **WAPC** to accept a token from user and check whether it is a keyword.

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

int main()
{
    int flag=0;
    char* kw [] = {"auto", "break", "case", "char", "continue", "
do", "default", "const", "double", "else", "enum", "extern", "for
", "if", "goto", "float", "int", "long", "register", "return", "s
igned", "static", "sizeof", "short", "struct", "switch", "typedef
", "union", "void", "while", "volatile", "unsigned"};
    char* ckw []={"if","else","default","switch","case"};
    char* ikw []={"for","while","do","goto","break","continue"};
    char* dkw []={"int","char","float","double","long","signed","
unsigned"};
    char item[100];
    int i;
    printf("\nEnter a token: ");
    scanf("%s",item);
    for(i = 0; i < 32; ++i)
    {
        if(strcmp(item,kw[i]) == 0)
        {

            flag=1;
           break;
        }




    }
    if(flag==1)
    {
        for(int j=0;j<10;j++)
        {
            if(strcmp(item,ckw[j])== 0)
            {
                printf("Conditional Keyword Entered");
```

```
                break;
            }
            else if(strcmp(item,ikw[j])==0)
            {
                printf("\n Iteration keyword Entered");
                break;
            }
            else
            {
                printf("Data Type Keyword Entered");
                break;
            }
        }
    }
    else{
        printf("\n Not a keyword");
    }

    return 0;
}
```

```
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc type_of_keyword.c -o type_of_keyword } ; if ($?) { .\type_of_keyword }

Enter a token: break else continue
Data Type Keyword Entered
PS G:\CODER\C>
```

4. **WAPC** to check If it is a keyword, report the type of keyword: conditional kw (if, else, default, switch, case) iteration kw (for, while, do, goto, break, continue) data type kw (int, char, float, double, long, signed, unsigned, short), others

➢
```c
#include<stdio.h>
#include<string.h>

int main()
{
    char* kw [] = {"auto", "break", "case", "char", "continue", "
do", "default", "const", "double", "else", "enum", "extern", "for
", "if", "goto", "float", "int", "long", "register", "return", "s
igned", "static", "sizeof", "short", "struct", "switch", "typedef
", "union", "void", "while", "volatile", "unsigned"};
    char item[100];
    int i;
    printf("\nEnter a token: ");
    scanf("%s",item);
    for(i = 0; i < 32; ++i)
    {
        if(strcmp(item, kw[i]) == 0)
        {
            printf("\nKeyword entered");
            break;
        }
    }
    return 0;
}
```

```
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc keyword_or_not.c -o keyword_or_not } ; if ($?) { .\keyword_or_not }

Enter a token: break else continue

Keyword entered
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc keyword_or_not.c -o keyword_or_not } ; if ($?) { .\keyword_or_not }

Enter a token: hello i am piyush
PS G:\CODER\C>
```

5. **WAPC** to accept tokens from user and count the number of keywords

```c
#include<stdio.h>
#include<string.h>
int main()
{
    char* kw [] = {"auto", "break", "case", "char", "continue"
, "do", "default", "const", "double", "else", "enum", "extern"
, "for", "if", "goto", "float", "int", "long", "register", "re
turn", "signed", "static", "sizeof", "short", "struct", "switc
h", "typedef", "union", "void", "while", "volatile", "unsigned
"};
    char item[200];
    char temp[50];
    int i = 0, index = 0, countkw = 0, j;
    printf("\nEnter tokens with a space at the end: ");
    gets(item);
    strcat(item," ");
    while(item[i] != '\0')
    { if(item[i] == ' '){
            temp[index] = '\0';
            for(j = 0; j < 32; ++j){
                if(strcmp(temp, kw[j]) == 0)
     {
                    ++countkw;
                    break;} }
            index = 0;
            temp[0] = '\0'; }
        else
     {
            temp[index] = item[i];
            ++index; }
        ++i;  }
    printf("\nNumber of keywords  = %d",countkw);
    return 0;
}
```

```
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc count_keywords.c -o count_keywords } ; if ($?) { .\count_keywords }

Enter tokens with a space at the end: Hey I am Piyush

Number of keywords  = 0
PS G:\CODER\C>
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc count_keywords.c -o count_keywords } ; if ($?) { .\count_keywords }

Enter tokens with a space at the end: break else continue

Number of keywords  = 3
```

6. **WAPC** to check whether an identifier is legal or not

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

int main()
{
    char* kw [] = {"auto", "break", "case", "char", "continue"
, "do", "default", "const", "double", "else", "enum", "extern"
, "for", "if", "goto", "float", "int", "long", "register", "re
turn", "signed", "static", "sizeof", "short", "struct", "switc
h", "typedef", "union", "void", "while", "volatile", "unsigned
"};
    char item[100];
    int i;
    printf("\nEnter an identifier: ");
    gets(item);

    // if it starts with a number
    if(item[0] >= 65 && item[0] <= 90 || item[0] >= 97 && item
[0] <= 122 || item[0] == '_')
    {
        if(strlen(item) <= 32)    // checking for max length
        {
            for(i = 0; item[i]!= '\0'; ++i)         // checkin
g for whitespace
            {
                if(item[i]==' ')
                {
                    printf("\nCannot contain whitespace");
                    return 0;
                }
            }

            for(i = 0; i < 32; ++i)             // checking fo
r keywords
            {
                if(strcmp(item, kw[i]) == 0)
                {
                    printf("\nCannot be a keyword");
                    return 0;
                }
            }
```

```c
            for(i = 0; item[i] != '\0'; ++i)            // ch
ecking for special chars
            {
                if(item[i] >= 65 && item[i] <= 90 || item[i] >
= 97 && item[i] <= 122 || item[i] == '_' || item[i] >= 48 && i
tem[i] <= 57)
                    continue;
                else
                {
                    printf("\nContains special character which
 is not allowed");
                    return 0;
                }
            }
            printf("\nIdentifier is legal");
        }
        else
            printf("\nIdentifier is too long");
    }
    else
        printf("\nStarts with illegal symbol");
    return 0;
}
```

```
Enter an identifier: The avarage total marks

Cannot contain whitespace
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc identifier_legal_or_not.c -o identifier_legal_or_not } ; if ($?) { .\identifier_legal_or_not }


Enter an identifier: TotalAvarageMarks

Identifier is legal
PS G:\CODER\C>
```

7. **WAPC** to accept a single line of code. Verify whether it is a comment or not. If it is a comment, report the type of comment

```c
#include<stdio.h>
#include<string.h>
int main()
{
    char item[100];
    int length;
    printf("\n Enter a sigle line of Code:");
    gets(item);
    length=strlen(item);
    if(item[0]=='/')
    {
            if(item[1]=='/')
            {
                printf("\n Single Line comment found"); }
            else if(item[1]=='*')
            {
                if(item[length-2]=='*'&&item[length-1]=='/') {
                    printf("\n Multi-line comment");  }
                else {
                   printf("\n Invalid statement");  }
           }
            else
            {
                printf("\n Invalid Statement");

            }

    }
    else
    {
        printf("Not a comment");
    }
    return 0;

}
```

```
 Enter a sigle line of Code://Hello I am Piyush Chandra Chandra

 Single Line comment found
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc comments.c -o comments } ; if ($?) { .\comments }

 Enter a sigle line of Code:Hello I am Piyush Chandra Chandra
Not a comment
PS G:\CODER\C>
```

8. **WAPC** to accept a single line of code. Verify whether it contains a comment or not

```c
#include<stdio.h>
#include<string.h>
int main()
{
    char item[100];
    int length;
    printf("\n Enter a sigle line of Code:");
    gets(item);
    length=strlen(item);
    int i=0;
    int flag=0;
    for(i=0;i<length;i++)
{
    if(item[i]=='/') {
            if(item[i+1]=='/') {
                flag=1;
                break; }
            else if(item[i+1]=='*') {
                for(int j=i+1;j<length;j++) {
                    if(item[j]=='*') {
                        if(item[j+1]=='/')
                        {
                            flag=2;
                            break;
                        } }  } } }
}
    if(flag==1) {
           printf("\nContains Single Line Comment");
}
    else if(flag==2) {
        printf("\nContains a Multiline Comment");
}
    else{
        printf("Doesnt Contain a comment");}
    return 0;
}
```

```
 Enter a sigle line of Code://Hello this is Piyush!

Contains Single Line Comment
PS G:\CODER\C> cd "g:\CODER\C\" ; if ($?) { gcc comments2.c -o comments2 } ; if ($?) { .\comments2 }

 Enter a sigle line of Code:Hello this is Piyush!
Doesnt Contain a comment
PS G:\CODER\C>
```

9. **WAPC** to accept an infix expression. Convert it into a postfix expression and print it.

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

#define SIZE 100
char stack[SIZE];
int top=-1;

void push(char item)
{
   if(top>=SIZE)
   {
      printf("\n Stack Overflow");
   }
   else
   {

      stack[++top]=item;
   }

}
char pop()
{
   char item;
   if(top==-1)
   {
      printf("\n Stack is empty");
      exit(1);
   }
   else
   {
      item=stack[top];
      --top;
      return item;
   }

}

int is_operator(char symb)
{
   if(symb=='^'||symb=='*'||symb=='/'||symb=='+'||symb=='-')
   {
      return 1;
   }
   else{
```

```
        return 0;
    }
}
int precendence(char symb)
{
    if(symb=='^')
    {
        return 5;
    }
    else if(symb=='*')
    {
        return 4;
    }
   else if(symb=='^')
    {
        return 3;
    }
    else if(symb=='+')
    {
        return 2;
    }
    else if(symb=='-')
    {
        return 1;
    }
    else{
        return 0;
    }
}
void infix_to_postfix(char infix[],char postfix[])
{
    char item;int x;
    int i=0,j=0;
    push('(');
    strcat(infix,")");
    item=infix[0];
    while(item!='\0')
    {
        if(item=='(')
        {
            push(item);
        }
        else if(isdigit(item)||isalpha(item))
        {
            postfix[j]=item;
            ++j;
        }
        else if(is_operator(item)==1)
```

```
        {
           x=pop();
           while(precendence(x)>precendence(item))
           {
              postfix[j]=x;
              ++j;
              x=pop();
           }
           push(x);
           push(item);
        }
        else if(item==')')
        {
           x=pop();
           while(x!='(')
           {
              postfix[j]=x;
              ++j;
              x=pop();
           }
        }
        else{
           printf("\n Inavlid infix Expression");
           exit(1);
        }
     ++i;
     item=infix[i];
     }
     postfix[j]='\0';
}
int main()
{

     char infix[SIZE],postfix[SIZE];
     printf("\n Enter the infix expression:");
     gets(infix);
     infix_to_postfix(infix,postfix);
     puts(postfix);
     return 0;
}
```

```
 Enter the infix expression:AB54HG/6*5E
AB54HG65E*/
PS G:\CODER\C>
```

10. **WAPC** to convert infix to prefix

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

#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char item)
{
   if(top >= SIZE)
   {
      printf("\nStack Overflow");
   }
   else
   {
      ++top;
      stack[top] = item;  // stack[++top] = item;
   }
}

char pop()
{
   char item;
   if(top == -1)
   {
      printf("\nStack underflow");
      exit(1);
   }
   else
   {
      item = stack[top];
      --top;
      return item;
   }
}

int is_operator(char symb)
{
   if(symb == '^' || symb == '*' || symb == '/' || symb == '+' || symb == '-')
```

```
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int precedence(char symb)
{
    if(symb == '^')
    {
        return 5;
    }

    else if(symb == '*')
    {
        return 4;
    }

    else if(symb == '/')
    {
        return 3;
    }

    else if(symb == '+')
    {
        return 2;
    }

    else if(symb == '-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void infix_to_postfix(char infix[], char postfix[])
{
```

```
char item, x;
int i = 0, j = 0;
push('(');
strcat(infix, ")");
item=infix[0];
while(item != '\0')
{
   if(item == '(')
   {
      push(item);
   }
   else if(isdigit(item) || isalpha(item))   // alnum()
   {
      postfix[j] = item;
      ++j;
   }
   else if(is_operator(item) == 1)
   {
      x = pop();
      while(precedence(x) > precedence(item))
      {
         postfix[j] = x;
         ++j;
         x = pop();
      }
      push(x);
      push(item);
   }
   else if(item == ')')
   {
      x = pop();
      while(x != '(')
      {
         postfix[j] = x;
         ++j;
         x = pop();
      }
   }
   else
   {
      printf("\nInvalid infix expression");
      exit(1);
   }
```

```
        ++i;
        item = infix[i];
    }
    postfix[j] = '\0';
}

int main()
{
    char infix[SIZE], postfix[SIZE];
    printf("\nEnter the infix expression: ");
    gets(infix);
    infix_to_postfix(infix, postfix);
    puts(postfix);
    return 0;
}
```

```
Enter the infix expression: AB76*/SG67+
AB76*SG67/+
PS G:\CODER\C>
```