Outline

Stacks
- Application of Stacks
  - Reversing a String
  - Palindrome Example
  - Infix, Postfix and Prefix Notations

APPLICATION OF STACKS

Stacks can be used to reverse a sequence. For example, if a string "Computers" is entered by the user the stack can be used to create and display the reverse string "sretupmoC" as follows.

The program simply pushes all of the characters of the string into the stack. Then it pops and display until the stack is empty.

```c
#include<stdio.h>
#include<stdlib.h>
#define STACKSIZE 50
typedef struct{
    int count;
    int top;
    char items[STACKSIZE];/*stack can contain up to 50 characters*/
}STACK;

void push(STACK *, char);
char pop(STACK *);

int main()
{
    int i;
    STACK s;
    char p, A[20];
    s.top = -1; /*indicates that the stack is empty at the beginning*/
    s.count=0;
    printf("Input the string please:
");
    gets(A);  /* alternatively you can use scanf("%s",A); */
/*pushing the characters into the stack*/
    for(i=0;A[i] != '\0';i++){
        p = A[i];
        push(&s,p);
    }
    printf("The string in reverse order is:");
/*popping and printing the string in reverse order*/
    while(s.count >= 0){
        p=pop(&s);
        printf("%c",p);
    }
    return 0;
}

void push(STACK *Sptr, char ps) /*pushes ps into stack*/
{
    if(Sptr->top == STACKSIZE-1){
        printf("Stack is full\n");
    }
    Sptr->items[++Sptr->top]=ps;
    Sptr->count++;
}
```

PALINDROME EXAMPLE

The strings where the reading from the reverse and forward directions give the same word are called a palindrome. For example, the string “radar” is an example for palindrome. Among many other techniques stack can be used to determine if a string is a palindrome or not. This is achieved by pushing all the letters of a given word into stack and checking if the letters popped are the same as the letter of the string.

The following program determines if a given string is a palindrome or not?

```c
#include<stdio.h>
#include<stdlib.h>
#define STACKSIZE 25
typedef struct{
    int count;
    int top;
    char items[STACKSIZE];/*stack can contain up to 20 characters*/
}STACK;
void push(STACK *, int);
char pop(STACK *);
int main()
{
    int i, check=0, size;
    STACK s;
    char p, A[20];
    s.count=0;
    s.top = -1;    /*indicates that the stack is empty at the beginning*/
    printf("Enter the string\n");
    gets(A);    /* alternatively you can use scanf("%s",A); */
    /*pushing the characters of the string into the stack*/
    for(i=0;A[i]!='\0';i++){
        p = A[i];
        push(&s,p);
```
size = s.count;
/*popping and checking if the letters are equal in rev. & forw. direction*/
for(i=0;i<=size-1;i++){
    p = pop(&s);
    if(p != A[i])
        check ++;
} 
if(check == 0)
    printf("\nThe string %s is a palindrome\n",A);
else
    printf("\nThe string %s is NOT a palindrome\n",A);
return 0;
}

void push(STACK *Sptr, int ps) /*pushes ps into stack*/
{
    if(Sptr->top == STACKSIZE-1){
        printf("Stack is full\n");
        exit(1); /*exit from the function*/
    }
    else {
        Sptr->top ++;
        Sptr->count ++;
        Sptr->items[Sptr->top] = ps;
    }
}

char pop(STACK *Sptr)
{
    char pp;
    if(Sptr->top == -1){
        printf("Stack is empty\n");
        exit(1); /*exit from the function*/
    }
    else {
        pp = Sptr->items[Sptr->top];
        Sptr->top --;
        Sptr->count --;
    }
    return pp;
}
**INFIX, POSTFIX AND PREFIX NOTATIONS**

<table>
<thead>
<tr>
<th>Infix</th>
<th>Postfix</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>A+B</td>
<td>AB+</td>
<td>+AB</td>
</tr>
<tr>
<td>A+B-C</td>
<td>AB+C-</td>
<td>-+ABC</td>
</tr>
<tr>
<td>(A+B)*(C-D)</td>
<td>AB+CD-*</td>
<td>*+AB-CD</td>
</tr>
</tbody>
</table>

Infix, Postfix and Prefix notations are used in many calculators. The easiest way to implement the Postfix and Prefix operations is to use stack. Infix and prefix notations can be converted to postfix notation using stack.

The reason why postfix notation is preferred is that you don’t need any parenthesis and there is no prescience problem.

- In **Postfix** notation the expression is scanned from left to right. When a number is seen it is pushed onto the stack; when an operator is seen the operator is applied to the two numbers popped from the stack and the result is pushed back to the stack.

**Ex:** 6 5 2 3 + 8 * 3 + * is evaluated as follows:

- The first 4 numbers are pushed onto the stack:

  - Next + is read, so 3 and 2 are popped and their sum (3+2=5) is pushed onto the stack:

  - Next 8 is pushed onto the stack:

  - Next * is read, so 8 and 5 are popped and their product (8*5=40) is pushed onto the stack:

  - Next + is read, so 40 and 5 are popped and their product (40+5=45) is pushed onto the stack:

  - Next 3 is pushed onto the stack:

  - Next + is read, so 3 and 45 are popped and their sum (3+45=48) is pushed onto the stack:
- Finally * is read, so 48 and 6 are popped and their product (6+48=288) is pushed onto the stack:

\[ \begin{array}{c}
\text{top} \\
288 \\
\end{array} \]

Note that \( 6 \ 5 \ 2 \ 3 + 8 * + 3 + * \) = 288 where the corresponding infix notation is:((3+2)*8+5+3)*6

The following program is an example of the evaluation of a given Postfix expression. Note that the user may enter any postfix expression as a string. The program reads the characters in the string one by one and if the character is a digit (number) than it converts the character into an integer otherwise uses the related operation.

```c
#include<stdio.h>
#include<stdlib.h>
#define STACKSIZE 20
typedef struct{
    int top;
    int items[STACKSIZE]; /*stack can contain up to 20 integers*/
}STACK;
void push(STACK *, int);
int pop(STACK *);
int calculate(char []);
int main()
{ int result;
  char E[50];
  printf("Enter your Postfix expression (don't use space character): \n");
  scanf("%s",E);
  result = calculate(E);
  printf("The result of the Postfix expression %s=%d \n",E,result);
  return 0;
}
int calculate(char exp[])
{ STACK s;
  s.top =-1; /*indicates that the stack is empty at the beginning*/
  int i,num1,num2,value;
  for(i=0; exp[i]!='\0';i++)
  { if(exp[i] >='0' && exp[i] <='9')  /*checks if exp[i] has a digit*/
    push(&s,(int)(exp[i] -'0')); /*converts digit into integer*/
    else{
      num1=pop(&s);
      num2=pop(&s);
      switch(exp[i]){case '+': value=num2+num1;break;
        case '-': value=num2-num1;break;
        case '*': value=num2*num1;break;
        case '/': value=num2/num1;break;
        default : printf("Illegal Operator \n");
        exit(1);
      }
      push(&s,value);
    }
  }
  return 0;
}
```

288
return pop(&s);
}

void push(STACK *Sptr, int ps) /*pushes ps into stack*/
{
    if(Sptr->top == STACKSIZE-1){
        printf("Stack is full\n");
        exit(1); /*exit from the function*/
    }
    else {
        Sptr->top++;
        Sptr->items[Sptr->top]= ps;
    }
}

int pop(STACK *Sptr)
{
    int pp;
    if(Sptr->top == -1){
        printf("Stack is empty\n");
        exit(1); /*exit from the function*/
    }
    else {
        pp = Sptr->items[Sptr->top];
        Sptr->top--;
    }
    return pp;
}