UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

Algoritmos y estructura de datos

Practica 8. Pilas dinámicas

Alumno: Caudillo Sánchez Diego

Matricula: 1249199

Grupo: 551

Docente: Alma Leticia Palacios Guerrero

Fecha de entrega: 03/Mayo/2019

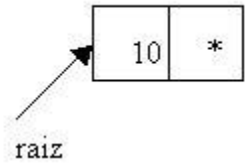
Introducción

Una lista se comporta como una pila si las inserciones y extracciones las hacemos por un mismo lado de la lista. También se las llama listas LIFO (Last In First Out - último en entrar primero en salir).

Importante: Una pila al ser una lista puede almacenar en el campo de información cualquier tipo de valor (int, char, float, vector de caracteres, un objeto, etc). Para estudiar el mecanismo de utilización de una pila supondremos que en el campo de información almacena un entero (para una fácil interpretación y codificación)

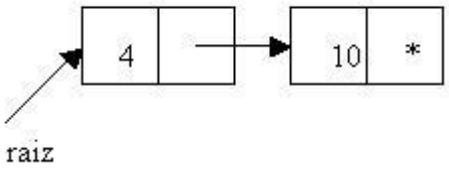
Inicialmente la PILA está vacía y decimos que el puntero raíz apunta a **null** (Si apunta a null decimos que no tiene una dirección de memoria):

Insertamos un valor entero en la pila: insertar(10)



Luego de realizar la inserción la lista tipo pila queda de esta manera: un nodo con el valor 10 y raíz apunta a dicho nodo. El puntero del nodo apunta a **null** ya que no hay otro nodo después de este.

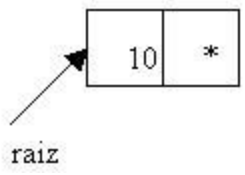
Insertamos luego el valor 4: insertar(4)



Ahora el primer nodo de la pila es el que almacena el valor cuatro. raíz apunta a dicho nodo. Recordemos que raíz es el puntero externo a la lista que almacena la dirección del primer nodo. El nodo que acabamos de insertar en el campo puntero guarda la dirección del nodo que almacena el valor 10.

Ahora qué sucede si extraemos un nodo de la pila. ¿Cuál se extrae? Como sabemos en una pila se extrae el último en entrar.

Al extraer de la pila tenemos: extraer()



La pila ha quedado con un nodo.

Hay que tener cuidado que si se extrae un nuevo nodo la pila quedará vacía y no se podrá extraer otros valores (avisar que la pila está vacía) La pila ha quedado con un nodo. Hay que tener cuidado que si se extrae un nuevo nodo la pila quedará vacía y no se podrá extraer otros valores (avisar que la pila está vacía)

Competencia. Aplicar eficientemente el principio LIFO para generar soluciones creativas a problemas de ingeniería.

Problema

Se pide diseñar e implementar un algoritmo basado en el paradigma LIFO que evalúe una expresión postfija dada por el usuario.

- El programa debe mostrar el estado de la pila durante el proceso de evaluación de la expresión.
- Las expresiones ya deben estar en notación postfija.
- El tamaño máximo de la expresión es indeterminado.
- La pila debe ser evaluada después de haberse capturado completa; es decir, la función que captura. no debe evaluar la expresión.
- La expresión solo debe contener números enteros y operadores matemáticos.
- Evite utilizar variables globales.
- El programa debe organizado en funciones o métodos.
- El programa debe repetirse N veces, hasta que el usuario elija la opción de salida.

Código

```
/*
Alumno: Caudillo Sánchez Diego
Materia: Algoritmos y estructura de datos
Grupo:551

>>>Descripción del programa<<<

Una de las aplicaciones de las pilas es la evaluación de expresiones postfijas.
Cada operador en una expresión escrita en notación postfija utiliza como
operadores
```

```
los dos valores previos en la cadena o un valor de la cadena y el resultado de la
evaluación de un operador anterior. Por ejemplo, Suponga que el usuario introduce
la expresión postfija 723+- El proceso sería: Los operandos 7,2 y 3 entran a la
pila.
*El operador + se evalúa extrayendo 7 y 2 de la pila.
*El resultado de la operación anterior entra a la pila (quedando 9 y 3 en la
pila)
*El operador – se evalúa con los dos datos 9 y 3.
*El resultado de la expresión es 6.
*Se pide diseñar e implementar un algoritmo basado en el paradigma LIFO que
evalúe una expresión postfija
dada por el usuario.

*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

struct nodo
{
    int tope;
    int capacidad;
    int* array;
};

/*Cabeceras de función*/
struct nodo* createStack(int lenght);
int isEmpty(struct nodo* stack);
char pop(struct nodo* stack);
void push(struct nodo* stack, char op);
int evalutePostfix(char* exp);
void printStr(char* str);
char *getStr(void);

int main(int argc, char const *argv[])
{
    char opc;
    printf("Captura la funcion: ");
    char *str = getStr();
    puts("Funcion a evaluar");
    printStr(str);
    printf("\nResultado postfijo: %d", evalutePostfix(str));
    setbuf(stdin, 0);
    do
```

```
    {
        setbuf(stdin, 0);
        printf("\nOtra funci%cn s/n?: ",162); scanf(" %c", &opc);
        setbuf(stdin, 0);
        switch (opc)
        {
            case 's':printf("Captura la funcion: ");
                char *str = getStr();
                puts("Funcion a evaluar");
                printStr(str);
                printf("\nResultado postfijo: %d\n", evalutePostfix(str));
                break;

            default: printf("No existe!");
                setbuf(stdin, 0);
                getchar();
                setbuf(stdin, 0);
                break;
        }setbuf(stdin, 0);
    } while (opc != 'n');
    return 0;
}


/*
DESCRIPCION
    Funcion la cual crea una pila dinamica de acuerdo a la captura del
    usuario. La funcion devuelve un apuntador de tipo struc nodo.
PARAMETROS
    lenght: Variable de tipo entero la cual es el tamaño de la cadena
    capturada por el usuario.
*/
struct nodo* createStack(int lenght)
{
    struct nodo* stack = (struct nodo*) malloc(sizeof(struct nodo));

    if(!stack) return NULL;

    stack->tope = -1;
    stack->capacidad = lenght;
    stack->array = (int*) malloc(stack->capacidad * sizeof(int));

    if(!stack->array) return NULL;

    return stack;
```

```
}


/*
DESCRIPCION
    Funcion que revisa si la pila se encuentra vacia.
PARAMETROS
    stack: apuntador del tipo de dato struct nodo el cual manda la pila para revisar
    si el tope es -1 indicando que esta vacio.
*/
int isEmpty(struct nodo* stack)
{ return stack->tope == -1; }


/*
DESCRIPCION
    Funcion que quita un dato de la pila y lo retorna mediante la funcion. El tipo
    de dato que devuelve es de tipo char.
PARAMETROS
    stack: apuntador de tipo struct nodo. Se refiere a la pila que mediante la funcion
    isEmpty verifica que no este vacia, para asi remover y posteriormente mandar el dato.
*/
char pop(struct nodo* stack)
{
    if(!isEmpty(stack)) return stack->array[stack->tope--];
    return 0;
}

/*
DESCRIPCION
    Funcion que inserta un dato en la pila. La funcion es del tipo void asi que no
    retorna ningun valor.
PARAMETROS
    stack: apuntador del tipo struct nodo el cual simula la pila, asi pudiendo insertar el
dato en ella.
    op: del tipo char, y es el dato el cual sera insertado en el tope de la pila.
*/
void push(struct nodo* stack, char op)
{ stack->array[++stack->tope] = op; }

/*
DESCRIPCION
    Funcion la cual evalua una funcion posfija. La funcion recibe como parametro un apuntador
a cadena
    el cual contiene los datos de la funcion a evaluar.
```

```
PARAMETROS
    exp: apuntador del tipo char el cual contiene el apuntador a la funcion que se va evaluar.
*/
int evalutePostfix(char* exp)
{
    struct nodo* stack =  createStack(strlen(exp)); // crea la pila.
    int i;

    if(!stack) return -1;   // Retorna un -1 si no fue posible la creacion de la pila.

    for(i = 0; exp[i]; i++)
    {
        if(isdigit(exp[i])) push(stack, exp[i] - '0');  // se insertan los datos de la cadena
en la pila
        //mientras detecta que los valores que se insertan sean numeros.

        else
        {
            int val1 = pop(stack);  // Si no son numero entonces remueve dos datos de la pila
para evaluarlos.
            int val2 = pop(stack);
            switch (exp[i]) // realiza la operacion dependiendo del del signo que se haya
encontrado.
            // y el resultado lo guarda en la pila.
            {
                case '+': push(stack, val2 + val1); break;
                case '-': push(stack, val2 - val1); break;
                case '*': push(stack, val2 * val1); break;
                case '/': push(stack, val2 / val1); break;
            }
        }
    }return pop(stack); //retorna el valor del resultaodo.
}
/*
    DESCRIPCION
        Función que imprime una cadena. La función no devuelve un valor.
    PARAMETROS
        str: apuntador de tipo char, la cual contiene la dirección de una
        cadena capturada.
*/
void printStr(char* str)
{ for(int i = 0; str[i]; i++) printf("%c", str[i]); }

/*
    DESCRIPCION
```

```
        Función que retorna el apuntador a la cadena la cual se ha capturado
        los dato de la funcion. La cadena es dinamica.
    PARAMETROS
        No recibe parametros.
*/
char *getStr(void)
{
    char *line = NULL, *tmp = NULL;
    size_t size = 0, index = 0;
    int ch = EOF;

    while (ch)                                  // Mientras
    {
        ch = getc(stdin);
        if (ch == EOF || ch == '\n') ch = 0;    // Verifica si necesitamos parar.
        if (size <= index)                      // Verificamos si necesitamos expandir memoria
        {
            size += 2;                          // Expandemos memoria con 2 locaciones más.
            tmp = realloc(line, size);          // Expandemos el bloque de memoria reservado
anteriormente.
            if (!tmp)                           // En caso de que no se pueda reservar
memoria.
            {
                free(line);                     // Liberamos la memoria reservada.
                line = NULL;                    // Terminamos en null la cadena.
                break;
            }line = tmp;                        // Le pasamos la dirección expandida a nuestra
cadena para
                                                // seguir escribiendo.
        }
        /* Actually store the thing. */
        line[index++] = ch;                     // Almacenamiento del dato capturado.
    }return line;                               // Regresa la direccion donde se ha guardada
la cadena.
}
```

Evidencia de ejecución

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 8> ./d
Captura la funcion: 729-+
Funcion a evaluar
729-+
Resultado postfijo: 0
Otra función s/n?: s
Captura la funcion: 999-+
Funcion a evaluar
999-+
Resultado postfijo: 9

Otra función s/n?: n
No existe!
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 8>
```

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 8> ./d
Captura la funcion: 48-3
Funcion a evaluar
48-3
Resultado postfijo: 3
Otra función s/n?: s
Captura la funcion: 99+1
Funcion a evaluar
99+1
Resultado postfijo: 1

Otra función s/n?: n
No existe!
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 8>
```

Conclusión

Con la realización de esta práctica se aplicó lo visto en clase. El uso de memoria dinámica aplicándola en las pilas de esa manera no hay gasto de memoria ya que podemos ir expandiendo el bloque de memoria de acuerdo a las necesidades del usuario. Una de las partes las cuales se complicó más, fue la parte de hacer los *pop* y los *push* ya que tenia problemas con el desplazamiento de la memoria para obtener los datos. Pero al dar un repaso del tema quedo solucionado.

Bibliografía

Joyanes, L. Zahonero, I. (2005). Programación en C. Metodología, algoritmos y estructura de datos. Madrid: McGraw Hill.

http://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?codigo=115&punto=&inicio=40