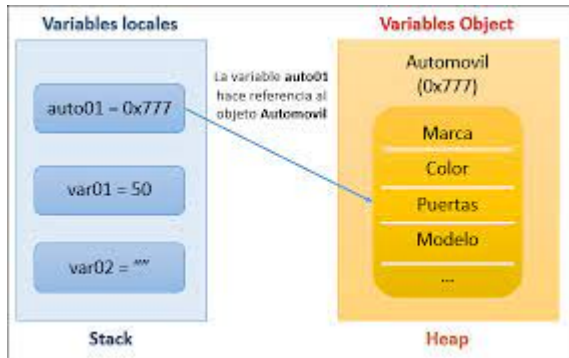


Punteros

¿Qué son los punteros?

Un puntero es una variable que almacena la dirección de memoria de un objeto. Los punteros se usan ampliamente en C y C++ para tres propósitos principales: para asignar nuevos objetos en el montón, para pasar funciones a otras funciones.



Los punteros y el aspersand

El ampersand es un operador de C/C++ y es comúnmente utilizado para los punteros, este operador nos permite obtener la dirección de memoria de una variable cualquiera y es justo esto (la dirección en memoria) lo que utilizan los punteros para referenciar valores.

Los punetors y el asterisco

El asterisco es, por decirlo de alguna forma, el operador por excelencia de los punteros su utilidad radica en que si el valor de dicho apuntador corresponde a una dirección de memoria el asterisco nos permite resolverla y acceder al valor almacenado allí. Viéndolo desde otro enfoque, un apuntador es únicamente una dirección de memoria (un número) el asterisco es el que hace la magia de obtener el valor referenciado por dicha dirección.

¿Cómo funcionan?

Como podemos observar en la imagen anterior, existen dos espacios en los cuales se guardan los datos, funciones o estructuras, en el stack se guardaran los datos llamados punteros los cuales pueden apuntar a espacios de memoria y modificarlos sin necesidad de crear una copia y almacenarla en el heap, a diferencia de los datos que se guardan en el heap si no se usa un puntero los datos que se vayan modificando se creara una copia y se guardara en el mismo ocupando mucha mas memoria.

```
#include <iostream>

using namespace std;

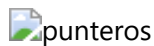
int main(){
```

```
int num;
num = 100;

int *apuntador_num ;
apuntador_num = &num;

cout << "num: " << num << endl;
cout << "&num: " << &num << endl;
cout << "apuntador_num: " << apuntador_num << endl;
cout << "*apuntador_num: " << *apuntador_num << endl;
cout << "&apuntador_num: " << &apuntador_num << endl;
}
```

En el código anterior podemos observar un código que usa punteros y nos permite observar como funcionan dichos punteros y como este recupera las direcciones del dato, como lo vemos a continuación:



Funcionalidades de los punteros

Las funcionalidades de los punteros pueden ser infinitas, pero en general en este bloque pudimos revisar las siguientes:

- Definir datos, estructuras, procedimientos y funciones.
- Colas
- Pilas
- Listas
- Árboles Binarios

¿Que son las colas?

Las colas se utilizan en sistemas informáticos, transportes y operaciones de investigación (entre otros), donde los objetos, personas o eventos son tomados como datos que se almacenan y se guardan mediante colas para su posterior procesamiento.



Código de ejemplo

```
#include <iostream>
using namespace std;

struct nodo          // [ # ]>-->
{
    int nro;
    struct nodo *sgte;
};

struct cola          // <--< >-->
```

```
{
    nodo *delante;
    nodo *atras ;
};

void encolar( struct cola &q, int valor )
{
    struct nodo *aux = new(struct nodo);

    aux->nro = valor;
    aux->sgte = NULL;

    if( q.delante == NULL)
        q.delante = aux;    // encola el primero elemento
    else
        (q.atras)->sgte = aux;
    q.atras = aux;          // puntero que siempre apunta al ultimo elemento
}

int desencolar( struct cola &q )
{
    int num ;
    struct nodo *aux ;

    aux = q.delante;        // aux apunta al inicio de la cola
    num = aux->nro;
    q.delante = (q.delante)->sgte;
    delete(aux);           // libera memoria a donde apuntaba aux

    return num;
}

void muestraCola( struct cola q )
{
    struct nodo *aux;
    aux = q.delante;

    while( aux != NULL )
    {
        cout<<"    "<< aux->nro ;
        aux = aux->sgte;
    }
}

void vaciaCola( struct cola &q)
{
    struct nodo *aux;

    while( q.delante != NULL)
    {
        aux = q.delante;
        q.delante = aux->sgte;
        delete(aux);
    }
}
```

```

    }
    q.delante = NULL;
    q.atras   = NULL;
}

int menu()
{
    int op=0;
    system("cls");
    cout<< endl <<"[...] COLAS          "
        << endl <<"  0.  SALIR          "
        << endl <<"  1.  ENCOLAR        "
        << endl <<"  2.  DESENCOLAR     "
        << endl <<"  3.  MOSTRAR COLA   "
        << endl <<"  4.  VACIAR COLA    "
        << endl <<"  5.  SALIR          "
        << endl <<"\n INGRESE OPCION:  ";
    cin>> op;
    return op;
}

int main()
{
    struct cola q;
    q.delante = NULL;
    q.atras   = NULL;

    int dato; // numero a encolar
    int x ;   // numero que devuelve la funcon pop

    system("color 0b");
    do
    {
        switch( menu() )
        {
            case 0: exit(0);
            case 1:
                cout<< "\n NUMERO A ENCOLAR: "; cin>> dato;
                encolar( q, dato );
                cout<<"\n\n\t\tNumero " << dato << " encolado...\n\n";
                break;
            case 2:
                x = desencolar( q );
                cout<<"\n\n\t\tNumero " << x << " desencolado...\n\n";
                break;
            case 3:
                cout << "\n\n MOSTRANDO COLA\n\n";
                if(q.delante!=NULL) muestraCola( q );
                else cout<<"\n\n\tCola vacia...!"<<endl;
                break;
            case 4:
                vaciaCola( q );
                cout<<"\n\n\t\tHecho...\n\n";
                break;
        }
    }
    while(1);
}

```

```

    }
    cout<<endl<<endl;
    system("pause");
}while(true);
return 0;
}

```

Salida del codigo

 Salida codigo parte 1  Salida codigo parte 2

A continuacion explicaremos brevemente las opciones que se presentan en el menu colas:

- Salir: Finaliza el bucle en el cual se desarrollan las demas funciones.
- Encolar: Permite añadir un nuevo nodo en el que se va a guardar un dato como observamos en la imagen.
- Desencolar: Elimina el primer dato que se haya ingresado.
- Mostrar cola: Como podemos observar en la imagen anterior permite mostrar los elementos que se añadieron a la cola.
- Vaciar cola: Elimina todos los datos de la cola.

Para mas informacion [presione aquí](#)

¿Qué son las pilas?

Una pila (stack en inglés) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out último en entrar, primero en salir) que permite almacenar y recuperar datos.

 Pilas

Codigo de ejemplo

```

#include <iostream>
using namespace std;

struct nodo{
    int nro;
    struct nodo *sgte;
};

typedef nodo *ptrPila;    // creando nodo tipo puntero( tipo de valor )
//struct nodo1 ptrPila1;

void push( ptrPila &p, int valor )    // Apilar
{
    ptrPila aux = new(struct nodo);    // apuntamos al nuevo nodo creado
    aux->nro = valor;
}

```

```
    aux->sgte = p ;
    p = aux ;
    cout <<" << apilado >> " <<endl;
}

void pop( ptrPila &p )    // Desapilar
{
    ptrPila aux;

    aux = p ;
    //num = aux->nro;    // asignamos el primer valor de la pila
    cout <<" << desapilado >> " << aux->nro <<endl;

    p = aux->sgte ;
    delete(aux);
}

void mostrar_pila( ptrPila p )
{
    ptrPila aux;
    aux = p;    // apunta al inicio de la lista

    while( aux !=NULL )
    {
        cout<<"\t"<< aux->nro <<endl;
        aux = aux->sgte;
    }
}

void destruir_pila( ptrPila &p)
{
    ptrPila aux;

    while( p != NULL)
    {
        aux = p;
        p = aux->sgte;
        cout<<"despachando: " << aux->nro <<"\t";
        delete(aux);
    }
    cout<<"\n\n\t\t Pila despachada...\n\n";
}

int menu()
{
    int op;
    cout<<endl;
    cout<<" 1. APILAR                "<<endl;
    cout<<" 2. DESAPILAR           "<<endl;
    cout<<" 3. ELIMINAR PILA         "<<endl;
    cout<<" 4. SALIR                  "<<endl;
    cout<<"\n INGRESE OPCION: ";
    cin>> op;
    if (op==4) exit(0);
}
```

```

        return op;
    }

    int main()
    {
        ptrPila p = NULL; // creando pila
        int valor;
        int op;

        do
        {
            cout<<"\n\n    FUNCIONALIDAD PILA : \n";
            if(p==NULL)
                cout<<"\t << vacia >> ";
            else
                mostrar_pila( p );
            switch(menu())
            {
                case 1: cout<< "\n NUMERO A APILAR: "; cin>> valor;
                        push( p, valor );
                        break;
                case 2: pop( p );
                        break;
                case 3: destruir_pila( p );
                        break;
            }
        }while(op!=5);
        return 0;
    }

```

Salida del programa

 Salida codigo parte 1  Salida codigo parte 2

Como podemos observar en las pilas funcionan apilado un dato o caracter uno sobre otro los mismos se apilaran en orden y se desapilaran de la misma manera de arriba hacia abajo, si se escoje la funcion eliminar pila se borran todos.

Para mas ejemplos Para mas ejemplos [presione aquí](#)

¿Qué son las listas?

La clase list de la Biblioteca estándar de C++ es una plantilla de clase de contenedores de secuencias que mantienen sus elementos en disposición lineal y permiten realizar inserciones y eliminaciones de manera eficiente en cualquier ubicación de la secuencia.

 Listas

Codigo de ejemplo

```
#include <iostream>
#include <stdlib.h>
using namespace std;

struct nodo{
    int nro;          // en este caso es un numero entero
    struct nodo *sgte;
};

typedef struct nodo *Tlista;

void insertarInicio(Tlista &lista, int valor)
{
    Tlista q;
    q = new(struct nodo);
    q->nro = valor;
    q->sgte = lista;
    lista = q;
}

void insertarFinal(Tlista &lista, int valor)
{
    Tlista t, q = new(struct nodo);
    q->nro = valor;
    q->sgte = NULL;
    if(lista==NULL)
        lista = q;
    else
    {
        t = lista;
        while(t->sgte!=NULL)
        {
            t = t->sgte;
        }
        t->sgte = q;
    }
}

int insertarAntesDespues()
{
    int _op, band;
    cout<<endl;
    cout<<"\t 1. Antes de la posicion          "<<endl;
    cout<<"\t 2. Despues de la posicion        "<<endl;

    cout<<"\n\t Opcion : "; cin>> _op;

    if(_op==1)
        band = -1;
    else
        band = 0;
}
```



```
        return band;
    }

void insertarElemento(Tlista &lista, int valor, int pos)
{
    Tlista q, t;
    int i;
    q = new(struct nodo);
    q->nro = valor;

    if(pos==1)
    {
        q->sgte = lista;
        lista = q;
    }
    else
    {
        int x = insertarAntesDespues();
        t = lista;
        for(i=1; t!=NULL; i++)
        {
            if(i==pos+x)
            {
                q->sgte = t->sgte;
                t->sgte = q;
                return;
            }
            t = t->sgte;
        }
    }
    cout<<"    Error...Posicion no encontrada..!"<<endl;
}

void buscarElemento(Tlista lista, int valor)
{
    Tlista q = lista;
    int i = 1, band = 0;

    while(q!=NULL)
    {
        if(q->nro==valor)
        {
            cout<<endl<<" Encontrada en posicion "<< i <<endl;
            band = 1;
        }
        q = q->sgte;
        i++;
    }

    if(band==0)
        cout<<"\n\n Numero no encontrado..!"<< endl;
}

void reportarLista(Tlista lista)
```

```
{  
    int i = 0;  
  
    while(lista != NULL)  
    {  
        cout << ' ' << i+1 << " " << lista->nro << endl;  
        lista = lista->sgte;  
        i++;  
    }  
}
```

```
void eliminarElemento(Tlista &lista, int valor)
```

```
{  
    Tlista p, ant;  
    p = lista;  
  
    if(lista!=NULL)  
    {  
        while(p!=NULL)  
        {  
            if(p->nro==valor)  
            {  
                if(p==lista)  
                    lista = lista->sgte;  
                else  
                    ant->sgte = p->sgte;  
  
                delete(p);  
                return;  
            }  
            ant = p;  
            p = p->sgte;  
        }  
    }  
    else  
        cout<<" Lista vacia..!";  
}
```

```
void eliminaRepetidos(Tlista &lista, int valor)
```

```
{  
    Tlista q, ant;  
    q = lista;  
    ant = lista;  
  
    while(q!=NULL)  
    {  
        if(q->nro==valor)  
        {  
            if(q==lista) // primero elemento  
            {  
                lista = lista->sgte;  
                delete(q);  
                q = lista;  
            }  
        }  
    }  
}
```

```
        }
        else
        {
            ant->sgte = q->sgte;
            delete(q);
            q = ant->sgte;
        }
    }
    else
    {
        ant = q;
        q = q->sgte;
    }

} // fin del while

cout<<"\n\n Valores eliminados..!"<<endl;
}

void menu1()
{
    cout<<"\n\t\tLISTA ENLAZADA SIMPLE\n\n";
    cout<<" 1. INSERTAR AL INICIO           "<<endl;
    cout<<" 2. INSERTAR AL FINAL             "<<endl;
    cout<<" 3. INSERTAR EN UNA POSICION          "<<endl;
    cout<<" 4. REPORTAR LISTA                    "<<endl;
    cout<<" 5. BUSCAR ELEMENTO                   "<<endl;
    cout<<" 6. ELIMINAR ELEMENTO 'V'             "<<endl;
    cout<<" 7. ELIMINAR ELEMENTOS CON VALOR 'V' "<<endl;
    cout<<" 8. SALIR                             "<<endl;

    cout<<"\n INGRESE OPCION: ";
}

int main()
{
    Tlista lista = NULL;
    int op;      // opcion del menu
    int _dato;   // elemento a ingresar
    int pos;     // posicion a insertar

    do
    {
        menu1(); cin>> op;
        switch(op)
        {
            case 1:
                cout<<"\n NUMERO A INSERTAR: "; cin>> _dato;
                insertarInicio(lista, _dato);
                break;
            case 2:
                cout<<"\n NUMERO A INSERTAR: "; cin>> _dato;
                insertarFinal(lista, _dato );
        }
    }
}
```

```

        break;
    case 3:
        cout<< "\n NUMERO A INSERTAR: "; cin>> _dato;
        cout<< " Posicion : "; cin>> pos;
        insertarElemento(lista, _dato, pos);
        break;
    case 4:
        cout << "\n\n MOSTRANDO LISTA\n\n";
        reportarLista(lista);
        break;
    case 5:
        cout<<"\n Valor a buscar: "; cin>> _dato;
        buscarElemento(lista, _dato);
        break;
    case 6:
        cout<<"\n Valor a eliminar: "; cin>> _dato;
        eliminarElemento(lista, _dato);
        break;
    case 7:
        cout<<"\n Valor repetido a eliminar: "; cin>> _dato;
        eliminaRepetidos(lista, _dato);
        break;
    }
    cout<<endl<<endl;
}while(op!=8);
return 0;
}

```

Salida del codigos

 Salida codigo parte 1  Salida codigo parte 2  Salida codigo parte 3

En este caso podemos observar que podemos crear una lista que vaya en orden, esto nos puede ayudar a organizar de manera ordenada nombres de personas u objetos, como podemos observar en el codigo tiene 8 opciones que funcionan de esta manera:


- Insertar al inicio o final: permite ingresar un valor o una cadena de caracteres al principio o al final de la lista.
- Reportar lista: Imprime la lista en el terminal.
- Buscar elemento: Busca el valor u cadena de caracteres que deseamos encontrar.
- Eliminar elemento 'V': Elimina un elemento especifico de cierta posicion.
- Eliminar elementos con valor 'V': Permite eliminar todos los datos que esten repetidos dentro de la lista.
- Salir: Esta opcion permite salir del bucle en el que se desarrollan las otras funciones.

Para mas ejemplos [presione aquí](#)

¿Qué es un arbol binario?

Es la manera recursiva como pasaremos por cada nodo del árbol, existes tres formas

- **Enorden:** Si visitamos primero hijo izquierdo, luego el padre y finalmente el hijo derecho
- **Preorden:** Primero el padre, luego el hijo izquierdo y finalmente el hijo derecho.
- **Postorden:** Primero hijo izquierdo, luego el hijo derecho y finalmente el padre

 arbol binario

Codigo de ejemplo

```
#include <iostream>
#include <cstdlib>
using namespace std;

struct nodo{
    int nro;
    struct nodo *izq, *der;
};

typedef struct nodo *ABB;

ABB crearNodo(int x)
{
    ABB nuevoNodo = new(struct nodo);
    nuevoNodo->nro = x;
    nuevoNodo->izq = NULL;
    nuevoNodo->der = NULL;

    return nuevoNodo;
}

void insertar(ABB &arbol, int x)
{
    if(arbol==NULL)
    {
        arbol = crearNodo(x);
    }
    else if(x < arbol->nro)
        insertar(arbol->izq, x);
    else if(x > arbol->nro)
        insertar(arbol->der, x);
}

void preOrden(ABB arbol)
{
    if(arbol!=NULL)
    {
        cout << arbol->nro << " ";
        preOrden(arbol->izq);
        preOrden(arbol->der);
    }
}

void enOrden(ABB arbol)
{

```

```
        if(arbol!=NULL)
        {
            enOrden(arbol->izq);
            cout << arbol->nro << " ";
            enOrden(arbol->der);
        }
    }

void postOrden(ABB arbol)
{
    if(arbol!=NULL)
    {
        postOrden(arbol->izq);
        postOrden(arbol->der);
        cout << arbol->nro << " ";
    }
}

void verArbol(ABB arbol, int n)
{
    if(arbol==NULL)
        return;
    verArbol(arbol->der, n+1);

    for(int i=0; i<n; i++)
        cout<<" ";

    cout<< arbol->nro <<endl;

    verArbol(arbol->izq, n+1);
}

int main()
{
    ABB arbol = NULL;    // creado Arbol

    int n; // numero de nodos del arbol
    int x; // elemento a insertar en cada nodo

    cout << "\n\t\t ..[ ARBOL BINARIO DE BUSQUEDA ].. \n\n";

    cout << " Numero de nodos del arbol: ";
    cin >> n;
    cout << endl;

    for(int i=0; i<n; i++)
    {
        cout << " Numero del nodo " << i+1 << ": ";
        cin >> x;
        insertar( arbol, x);
    }


    cout << "\n Mostrando ABB \n\n"; verArbol( arbol, 0);
    cout << "\n Recorridos del ABB";
```

```

    cout << "\n\n En orden    : ";    enOrden(arbol);
    cout << "\n\n Pre Orden  : ";    preOrden(arbol);
    cout << "\n\n Post Orden : ";    postOrden(arbol);
    cout << endl << endl;
    return 0;
}

```

Salida del programa

 arbol binario

Como podemos observar en este ejemplos no pide primero el numero de nodos que deseamos crear para que funcione el arbol, luego nos pide uno a uno los valores que seran ingresados en cada nodo, al final presenta el arbol, y las funciones en orden, pre orden y post orden presentaran los valores respectivamente.

Para mas ejemplos [presione aquí](#)

AUTOMATAS FINITOS DETERMINISTAS

¿Qué es un AUTOMATA FINITO DETERMINISTA?

Un autómata finito determinista (abreviado AFD) es un autómata finito que además es un sistema determinista; es decir, para cada estado en que se encuentre el autómata, y con cualquier símbolo del alfabeto leído, existe siempre no más de una transición posible desde ese estado y con ese símbolo.

Definición formal

Formalmente, se define como una 5-tupla $(Q, \Sigma, q_0, \delta, F)$ donde:

- Q : Es un conjunto de estados.
- Σ : Es un alfabeto.
- q_0 : Pertenece a Q es el estado inicial.
- δ : Es una función de transición.
- F : Es un subconjunto de Q es un conjunto de estados finales o de aceptación.

```

graph TD
    A((Estado inicial)) --> B[Estado intermedio]
    B --> C((Estado final))
    C --> C[Estado final]
    A --> A[Estado inicial]
    B --> A[Estado inicial]
    C --> B[Estado intermedio]
    A --> B[Estado intermedio]
    B --> B[Estado intermedio]
    C --> A[Estado inicial]
    A --> C[Estado final]
    B --> C[Estado final]

```

En el anterior esquema podemos ver como funciona un automata finito determinista en general puede pasar del estado inicial a un intermedio y de ahi a un final, o a su vez puede repetir varias veces el estado inicial e inmediatamente pasar al estado final.

Codigo ejemplo

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <unordered_map>

using namespace std;

struct Automata {
    vector<int> states;
    vector<char> symbols;
    unordered_map<int, unordered_map<char, int>> transitions;
    int initial_state;
    vector<int> final_states;
};

Automata read_automata(string filename) {
    ifstream fin(filename);
    Automata automata;
    int n, m, k;
    fin >> n >> m >> k;
    automata.states.resize(n);
    automata.symbols.resize(m);
    for (int i = 0; i < n; i++) {
        automata.states[i] = i;
    }
    for (int i = 0; i < m; i++) {
        fin >> automata.symbols[i];
    }
    for (int i = 0; i < k; i++) {
        int state;
        fin >> state;
        automata.final_states.push_back(state);
    }
    fin >> automata.initial_state;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            int next_state;
            fin >> next_state;
            automata.transitions[i][automata.symbols[j]] = next_state;
        }
    }
    return automata;
}

bool process_string(Automata automata, string s) {
```



```
int current_state = automata.initial_state;
for (char c : s) {
    if (automata.transitions[current_state].count(c) == 0) {
        return false;
    }
    current_state = automata.transitions[current_state][c];
}
return find(automata.final_states.begin(), automata.final_states.end(),
current_state) != automata.final_states.end();
}

int main() {
    Automata automata = read_automata("automata.txt");
    string s;
    cin >> s;
    if (process_string(automata, s)) {
        cout << "Accepted" << endl;
    } else {
        cout << "Rejected" << endl;
    }
    return 0;
}
```

Para mas ejemplos de AFD [presione aquí](#)

Los ordenadores son inútiles. Sólo pueden darte respuestas. -Pablo Picasso

Como lo menciona esta frase celebre de Pablo Picasso es cierto, los ordenadores nos dan solo respuestas, estas respuestas dependeran de nuestras preguntas así que lo mejor para seguir innovando dentro de la tecnología es seguir siendo creativos y cuestionarnos a nosotros mismos hasta que punto llegaremos.