# Notas segundo bimestre

***Nombre:*** Gabriel Del Valle.

***Fecha:*** 10/03/2023

## *Punteros:*

Los punteros se utilizan para manejar y manipular datos en la memoria del programa, lo que les da una gran flexibilidad y potencia a los programadores.

Para inicializar punteros usamos " * ".

```
int *puntero = arreglo;
char *c;
```

Para devolver una direccion de memoria de un puntero usamos el " & ";

```
int *puntero;
ptr = puntero;
```

A un puntero se le puede designar un espacio en la memoria concreta o pueden apuntar a otro puntero o al contenido de otro puntero.

```
int *ptr;
ptr = 0x1F3CE00A;
ptr = NULL;
```

```
char c;
char    *ptr;
ptr = &c;
```

```
char c;
char *ptr1;
char *ptr2;

ptr1 = &c;
ptr2 = ptr1;
```

# *Listas y Pilas*

Las listas y pilas son estructuras de datos fundamentales en la programación, utilizadas para almacenar y manipular datos de manera eficiente. Las listas son estructuras de datos lineales que permiten insertar, eliminar y acceder a elementos de manera eficiente. Las listas se implementan como una secuencia de nodos, cada uno de los cuales almacena un elemento y un puntero al siguiente nodo en la secuencia. Las listas son útiles cuando necesitamos agregar y eliminar elementos en el medio de la secuencia, ya que esto se puede hacer en tiempo constante, O(1), sin tener que mover otros elementos. Las listas también permiten iterar sobre los elementos en la secuencia de manera eficiente.

## Ejemplo lista

```
#include <iostream>
#include <list>

int main() {

  list<int> mi_lista;

  mi_lista.push_back(1);
  mi_lista.push_back(2);
  mi_lista.push_back(3);

  cout << "Elementos de la lista: ";
  for (auto it = mi_lista.begin(); it != mi_lista.end(); it++) {
    cout << *it << " ";
  }
  cout << endl;

  auto it = mi_lista.begin();
  it++;
  mi_lista.erase(it);

  cout << "Elementos de la lista actualizada: ";
  for (auto it = mi_lista.begin(); it != mi_lista.end(); it++) {
    cout << *it << " ";
  }
  cout << sendl;

  return 0;
}
```

En este ejemplo creamos una lista vacía, agregamos elementos a la lista, iteramos sobre los elementos e imprimimos su contenido, eliminamos el segundo elemento de la lista y finalmente imprimimos la lista actualizada.

## Ejemplo Pilas

```
#include <iostream>
#include <stack>

int main() {

  stack<int> mi_pila;

  mi_pila.push(1);
  mi_pila.push(2);
  mi_pila.push(3);

cout << "Elemento en la cima de la pila: " << mi_pila.top() << endl;

  mi_pila.pop();

  cout << "Nuevo elemento en la cima de la pila: " << mi_pila.top() << endl;

  return 0;
}
```

En este ejemplo creamos una pila vacía, agregamos elementos a la pila, imprimimos el elemento más reciente en la cima de la pila, eliminamos el elemento en la cima de la pila y finalmente imprimimos el nuevo elemento en la cima de la pila.

---

## *Funciones y Módulos*

### Estructuras:

las estructuras son tipos de datos definidos por el usuario que permiten agrupar diferentes tipos de datos relacionados en una sola entidad. Una estructura se puede considerar como una clase simple sin métodos ni funciones miembro, aunque en C++ las estructuras también pueden contener funciones miembro y ser heredadas.

Su sintaxis es:

```
struct nombre_de_la_estructura {
  tipo_de_dato_1 variable_1;
  tipo_de_dato_2 variable_2;
  tipo_de_dato_3 variable_3;
};
```

Ejemplo de estructura con una función incluida:

```
#include <iostream>

struct Persona {
```

```
  std::string nombre;
  int edad;
};

void imprimir_persona(Persona p) {
  std::cout << "Nombre: " << p.nombre << std::endl;
  std::cout << "Edad: " << p.edad << std::endl;
}

int main() {
  Persona mi_persona;
  mi_persona.nombre = "Juan";
  mi_persona.edad = 30;

  imprimir_persona(mi_persona);

  return 0;
}
```

### Estructuras de control repetitivas:

Las estructuras de control repetitivas (también conocidas como bucles o ciclos) son una herramienta fundamental en la programación que permiten repetir un bloque de código varias veces, mientras se cumpla una determinada condición. En C++, existen tres tipos principales de estructuras de control repetitivas: el bucle while, el bucle do-while y el bucle for.

### Modulos y registros

Los módulos se refieren a la capacidad de dividir el código en unidades lógicas y separadas, lo que permite una mejor organización y modularidad del código. Los módulos pueden ser implementados utilizando diversas técnicas, como archivos de cabecera, archivos de código fuente separados y bibliotecas compartidas.

Ejemplo: Primero hacemos una esturctura

```
struct Persona {
  std::string nombre;
  int edad;
  std::string direccion;
};
```

Creamos variables de tipo Persona y acceder a sus miembros de la siguiente manera:

```
Persona p1;
p1.nombre = "Juan";
p1.edad = 30;
p1.direccion = "Av. Siempreviva 123";
```

Utilizamos punteros para acceder a los miembros de una variable de tipo Persona:

```
Persona p2;
Persona* ptrP2 = &p2;
(*ptrP2).nombre = "Ana";
ptrP2->edad = 25;
ptrP2->direccion = "Calle Principal 456";
```

## Estructuras de control selectivas

Las estructuras de control selectivas son una herramienta esencial en la programación que nos permiten tomar decisiones basadas en el valor de una o más condiciones. El uso de la estructura de control selectiva adecuada dependerá del problema que se esté resolviendo y de las necesidades específicas de cada programa.

## Flujos y archivos

Los flujos son secuencias de bytes que se utilizan para enviar o recibir información entre diferentes fuentes o destinos de datos. En C++, existen cuatro tipos de flujos:

std::cin y std::cout: Estos flujos se utilizan para la entrada y salida estándar, respectivamente. Es decir, para interactuar con el usuario a través de la consola.

std::ifstream y std::ofstream: Estos flujos se utilizan para la lectura y escritura de archivos, respectivamente. Se pueden crear objetos de estos flujos para abrir un archivo y leer o escribir información en él.

```
std::ifstream archivoEntrada("nombre_del_archivo.txt");
```

Para abrir un archivo de salida, podemos usar la siguiente sintaxis:

```
std::ofstream archivoSalida("nombre_del_archivo.txt");
```

Para crear un objeto de este tipo, podemos usar la siguiente sintaxis:

```
std::stringstream flujo;
```

## Listas enlazadas

Las listas enlazadas son una estructura de datos en la que cada elemento (o nodo) contiene un valor y un puntero que apunta al siguiente nodo de la lista. En C++, se pueden implementar listas enlazadas mediante el uso de punteros y la creación de una clase que represente cada nodo de la lista.

### Ejemplo lista enlazada

```
class ListaEnlazada {
private:
  Nodo* primero;
  int tamano;
public:
  ListaEnlazada() {
    primero = nullptr;
    tamano = 0;
  }
  void agregarElemento(int valor) {
    Nodo* nuevoNodo = new Nodo;
    nuevoNodo->valor = valor;
    nuevoNodo->siguiente = primero;
    primero = nuevoNodo;
    tamano++;
  }
  void eliminarElemento(int valor) {
    Nodo* actual = primero;
    Nodo* anterior = nullptr;
    while (actual != nullptr && actual->valor != valor) {
      anterior = actual;
      actual = actual->siguiente;
    }
    if (actual != nullptr) {
      if (anterior == nullptr) {
        primero = actual->siguiente;
      }
      else {
        anterior->siguiente = actual->siguiente;
      }
      delete actual;
      tamano--;
    }
  }
  bool buscarElemento(int valor) {
    Nodo* actual = primero;
    while (actual != nullptr) {
      if (actual->valor == valor) {
        return true;
      }
      actual = actual->siguiente;
    }
    return false;
  }
};
```

En este ejemplo, el método agregarElemento agrega un nuevo elemento al principio de la lista, el método eliminarElemento elimina un elemento con un valor específico de la lista y el método buscarElemento busca un elemento con un valor específico en la lista.

## Colas

Las colas son una estructura de datos que permite agregar elementos al final de la cola y sacar elementos del frente de la cola. En C++, se pueden implementar colas utilizando punteros y creando una clase que represente cada nodo de la cola.

### Ejemplo de cola hecho en clase

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

---

# *Algoritmos de ordenación*

## Arboles

los árboles se pueden implementar mediante el uso de estructuras de datos y punteros. Una estructura de árbol consta de un nodo raíz y cero o más nodos hijos. Cada nodo puede tener un número variable de hijos, y cada hijo a su vez puede tener sus propios hijos.

### Ejemplo de arbol

```
// Arbol binario de busqueda aplicado en archivos, insertar registro en un nodo
(escribir), mostrar los registros uso del algoritmo y eliminar archivo ABB.
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <conio.h>
#include <iomanip>
using namespace std;

struct registro{
    int  NR;
```

```
    char dato[12];
    int  PD;
    int  PI;
    int  ARE;
}r, a;

struct encabezado{
    int NRS;    // numero de registros
    int RAIZ;
    int URE;
}e;

FILE *fd;
int lr, le;   // longitud de registro y encabezado

/*               Funcion Escribir Archivo
------------------------------------------------------------*/
void escribir()
{
    int x, pos;
    char rpt, lado;

    if( (fd=fopen("arbol_binario.txt", "w+"))==NULL )
    {
        cout << " No se creo el archivo arbol_binario.txt"<< endl;
        return;
    }
    // Inicializando variables
    e.NRS  = 0;
    e.RAIZ = -1;
    e.URE  = -1;
    fwrite(&e, le, 1, fd);

    do{
        r.NR = ++e.NRS;
        fflush(stdin);
        cout << " Nombre : "; gets(r.dato);
        r.PI = -1;
        r.PD = -1;
        r.ARE = 0;
        fseek(fd, 0, 2);  // al final
        fwrite(&r, lr, 1, fd);
        cout <<"1."<<endl;
        if(e.RAIZ == -1)
        {
            e.RAIZ = r.NR;
            cout << " * Raiz" << endl;
        }
        else
        {
            x = e.RAIZ;

            while( x != -1 )
            {
```

```
                cout <<"2."<<endl;
                pos = (x-1)*lr + le;
                fseek(fd, pos, SEEK_SET);
                fread(&a, lr, 1, fd);

                if( strcmp(r.dato, a.dato)>0 ){
                    x = a.PD;
                    lado = 'D';
                    continue;
                }
                if( strcmp(r.dato, a.dato)<0 ){
                    x = a.PI;
                    lado = 'I';
                    continue;
                }

            }
            if(lado=='D')
            {
                a.PD = r.NR;
                cout << " * Lado der. de "<< a.dato << endl;
            }
            if(lado=='I')
            {
                a.PI = r.NR;
                cout << " * Lado izq. de "<< a.dato << endl;
            }
            cout <<"3."<<endl;
            fseek(fd, pos, SEEK_SET);
            fwrite(&a, lr, 1, fd);
            cout <<"4."<<endl;
        }

        cout << " Mas registros [s/n]: "; cin >> rpt;
        cout << endl;
    }while(rpt!='n');

    fseek(fd, 0, SEEK_SET);
    fwrite(&e, le, 1, fd);
    fclose(fd);
}

/*               Funcion Mostrar Archivo
------------------------------------------------------------*/
void mostrar()
{
    if( (fd=fopen("arbol_binario.txt", "r"))==NULL )
    {
        cout << " No se pudo abrir el archivo"<< endl;
        return;
    }

    fread(&e, le, 1, fd);
```

```
    cout << " ----------------------------------"   << endl;
    cout << " |  NRS: " << e.NRS << "    RAIZ: " ;
    cout << e.RAIZ << "    URE: " << e.URE << "   |" << endl;
    cout << " ----------------------------------"   << endl<<endl;

    cout << setw(3) << "NR" << setw(10) << "NOMBRE" << setw(5 );
    cout << "PI" << setw(5) << "PD" << setw(6) << "ARE" << endl << endl;


    while(fread(&r, lr, 1, fd)!=NULL)
    {
        cout << " " << r.NR << "\t" << r.dato;
        cout << "\t" << r.PI << "\t" << r.PD;
        cout << "\t" << r.ARE << endl;
    }
/*
    fread(&r, lr, 1, fd);
    while( !feof(fd) )
    {
        cout << setw(3) << r.NR << setw(10) << r.dato;
        cout << setw(5) << r.PI << setw(5) << r.PD;
        cout << setw(5) << r.ARE << endl;
        fread(&r, lr, 1, fd);
    }
*/
    fclose(fd);
}

/*                  Funcion Leer
------------------------------------------------------------*/
void leer()
{
    int x, pos;
    bool band;
    char v_dato[12];

    if( (fd=fopen("arbol_binario.txt", "r"))==NULL )
    {
        cout << " No se pudo abrir ela archivo"<< endl;
        return;
    }

    fflush(stdin);
    cout << " Nombre : "; gets(v_dato);
    fread(&e, le, 1, fd);
    x = e.RAIZ;
    band = false;

    while( x != -1 )
    {
        pos = (x-1)*lr + le;
        fseek(fd, pos, SEEK_SET);
        fread(&r, lr, 1, fd);
```

```
        if( strcmp(v_dato, r.dato)>0 ) {
            x = r.PD;
            continue;
        }
        if( strcmp(v_dato, r.dato)<0 ) {
            x = r.PI;
            continue;
        }

        band = true;
        cout << "\n >> Dato " << r.dato << " encotrado..!" << endl;
        break;
    }

    if(band==false)
        cout << "\n\n >> No existe..!" << endl;

    fclose(fd);
}

/*                  Funcion Insertar
------------------------------------------------------------*/
void insertar()
{
    int x, pos;
    char rpt, lado;

    if( (fd=fopen("arbol_binario.txt", "r+"))==NULL )
    {
        cout << " No se pudo abrir el archivo"<< endl;
        return;
    }

    fread(&e, le, 1, fd);

    do{
        r.NR = ++e.NRS;
        fflush(stdin);
        cout << " Nombre : "; gets(r.dato);
        r.PI = -1;
        r.PD = -1;
        r.ARE = 0;
        fseek(fd, 0, SEEK_END);  // al final
        fwrite(&r, lr, 1, fd);

        if(e.RAIZ == -1)
        {
            e.RAIZ = r.NR;
            cout << " * Raiz" << endl;
        }
        else
        {
            x = e.RAIZ;
```

```
            while( x != -1 )
            {
                pos = (x-1)*lr + le;
                fseek(fd, pos, SEEK_SET);
                fread(&a, lr, 1, fd);

                if( strcmp(r.dato, a.dato)>0 ){
                    x = a.PD;
                    lado = 'D';
                    continue;
                }
                if( strcmp(r.dato, a.dato)<0 ){
                    x = a.PI;
                    lado = 'I';
                    continue;
                }
            }
            if(lado=='D')
            {
                a.PD = r.NR;
                cout << " * Lado der. de "<< a.dato << endl;
            }
            if(lado=='I')
            {
                a.PI = r.NR;
                cout << " * Lado izq. de "<< a.dato << endl;
            }

            fseek(fd, pos, SEEK_SET);
            fwrite(&a, lr, 1, fd);
        }

        cout << " Insertar mas registros [s/n]: "; cin >> rpt;
        cout << endl;
    }while(rpt!='n');

    fseek(fd, 0, SEEK_SET);
    fwrite(&e, le, 1, fd);
    fclose(fd);
}

/*                Funcion Eliminar
------------------------------------------------------------*/
void eliminar_registro()
{
    cout << "\n\n Implementando..!" << endl;
}

/*                Funcion Eliminar Archivo
------------------------------------------------------------*/
void eliminar_archivo()
{
    remove("arbol_binario.txt");
    cout << " >> Archivo arbol_binario.txt eliminado..!" << endl;
```

```
}

/*                 Menu de Opcion
-------------------------------------------------------------*/
void menu()
{
    cout << "\t\t ARBOLES BINARIOS DE BUSQUEDA EN ARCHIVOS \n\n";
    cout << "\t 1. Escribir              \n";
    cout << "\t 2. Mostrar               \n";
    cout << "\t 3. Leer                  \n";
    cout << "\t 4. Insertar              \n";
    cout << "\t 5. Eliminar Registro     \n";
    cout << "\t 6. Eliminar Archivo      \n";
    cout << "\t 7. Salir                 \n";
    cout << "\t >> Ingrese opcion:  ";
}

/*                 Funcion Principal
-------------------------------------------------------------*/
int main()
{
    int op;  // opcion
    lr = sizeof(struct registro);
    le = sizeof(struct encabezado);

    do
    {
        menu(); cin >> op; cout << "\n\n";

        switch(op)
        {
            case 1:
                escribir(); break;
            case 2:
                mostrar(); break;
            case 3:
                leer(); break;
            case 4:
                insertar(); break;
            case 5:
                eliminar_registro(); break;
            case 6:
                eliminar_archivo(); break;
            case 7:
                exit(0);
        }
        cout <<"\n\n ";
        //system("pause"); system("cls");

    }while(op>0);

    return 0;
}
```

## Arboles Binarios

Los árboles binarios son una de las estructuras de árboles más simples y comunes en programación. En un árbol binario, cada nodo tiene como máximo dos hijos, llamados el hijo izquierdo y el hijo derecho. Estos hijos, a su vez, también pueden ser nodos binarios con sus propios hijos izquierdo y derecho.

```
//-->> Arboles Binaros de busqueda - Recorridos por amplitud por Orden, Pre-Orden
y Post-Orden
#include <iostream>
#include <cstdlib>
using namespace std;

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

typedef struct nodo *ABB;
/* es un puntero de tipo nodo que hemos llamado ABB, que ulitizaremos
   para mayor facilidad de creacion de variables */

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;
}
// DEBER : candida de numeros a ingresar deber ser mayor 6...
//     Task                          group
//     verArbol( arbol, 0);            1    5    9
//     enOrden(arbol);                 2    6    ...
//     preOrden(arbol);                3    7
//     postOrden(arbol);               4    8
```