C++ III

C/C++

- **Variables Punteros**
  - **Operador * y &**
  - **Operadores new/delete**
  - **Aritmética de punteros**
  - **Segfault**
  - **Punteros múltiples**
  - **Punteros a funciones**
- Orientación a objetos
  - Objetos
  - Clases
  - Métodos
    - Constructor
    - Polimorfismo
  - Atributos
  - Encapsulación
  - Instanciación
  - Herencia

- Templates
- Sobrecarga de operadores

Punteros

Los punteros en C/C++ son variables que almacenan direcciones de memoria. En otras palabras, un puntero es una variable que apunta a otra variable almacenada en memoria.

Los punteros son una característica fundamental de C/C++ y permiten la manipulación directa de la memoria, lo que puede ser muy poderoso pero también puede ser peligroso si no se manejan adecuadamente.

Los punteros se utilizan comúnmente en C/C++ para la asignación dinámica de memoria y la manipulación de estructuras de datos complejas, como listas enlazadas y árboles. También se utilizan en la implementación de funciones de bajo nivel y en la interacción con el sistema operativo y el hardware del ordenador.

Punteros

- Se definen con el operador *
- Se inicializan con el operador **new**
- Se borran con el operador **delete**
- Se accede al valor con (*ptr)
  - Se puede asignar ***(prt) = valor**
  - O lo retorna **cout<<(*prt)<<endl;**
- Se puede asignar a un puntero el valor de otra variable no puntero usando el operador &, este operador sirve para obtener la dirección de memoria de la variable
- Se puede usar los operadores * y **&** en funciones para recibir referencias o punteros.
- Dos punteros se pueden asignar directamente con = ej: int *a; int *b; a=b;

```
using namespace std;
void incremento_ptr(int *p)
{
    (*p)++;
}
void incremento_ref(int &n)
{
    n++; //como es referencia no necesito la sintaxis de aritmética de punteros
}
int main()
{
    int num;
    //int *p = new int;
    int *p;
    p = new int;
    (*p) = 321;
    cout<<"dirrección de memeria = "<<p<<endl;
    cout<<"valor = "<<*p<<endl;

    incremento_ptr(p);
    cout<<"valor (p) = "<<*p<<endl;
    delete p;
    p=nullptr;
    cout<<"dirrección de memeria (p) = "<<p<<endl;

    p = &num; // asi se le asigna un valor a p, el operador & retorna la posición de memoria de num
    num = 1;

    cout<<"dirrección de memeria (p) = "<<p<<endl;
    cout<<"dirrección de memeria (num) = "<<&num<<endl;
    cout<<"valor (p) = "<<*p<<endl;
    cout<<"valor (num) = "<<num<<endl;

    incremento_ref(num);
    cout<<"valor (p) = "<<*p<<endl;
    cout<<"valor (num) = "<<num<<endl;
    return 0;
}
```

Punteros

- Los punteros también pueden ser arreglos de elementos. Para asignar el número de elementos usamos el operador [ ]
- Los punteros también se pueden inicializar de forma estática con el operador [ ] y asignar los elementos en la inicialización con los valores entre llaves
- Los valores inicializados con [ ] no necesitan ser liberados por delete, son memoria estática asignada en la compilación y se libera de forma automática.
- Veamos algunos ejemplos de manejo de arreglos con punteros.

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

void llenar(int *p,int n, int valor){
    for(int i=0;i<n;i++){
        p[i]=valor;
    }
}
void print(int *p,int n){
    cout<<"[";
    for(int i=0;i<n;i++){
        cout<<p[i]<<" ";
    }
    cout<<"]"<<endl;
}
void sumar(int *suma, int *v1, int *v2,int n){
    for(int i=0;i<n;i++){
        suma[i] = v1[i]+v2[i];
    }
}
int main(){
    unsigned int num=10;
    int *p1 = new int[num]; // creo un arreglo de 10 enteros,
    int p2[] = {1,1,1,1,1,1,1,1,1,1}; // 10 enteros con asignación estática e inicializado
                                      // con valores en 1, no necesita usarse delete, se borra solo
    int *p3 = new int[num]; // creo un arreglo de 10 enteros
    llenar(p1,num,2);
    print(p1,num);
    //llenar(p2,num,2); //está inicializado
    print(p2,num);
    sumar(p3,p1,p2,num);
    print(p3,num);

    delete[] p1;
    //delete[] p2;//no es necesario cuando es estática
    delete[] p3;

    return 0;
}
```

Aritmética de Punteros

- La aritmética de punteros permite a través de sumar y restas jugar con las posiciones de memoria.
- Sumar un valor **n** a un puntero quiere decir moverse a la “derecha” **n** posiciones de memoria y restar es moverse a la izquierda.
- Para acceder al valor que se encuentra en la posición de memoria toca hacer un juego en la sintaxis con los operadores () y *, como se muestra en el siguiente ejemplo.

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

void print(int *p,int n){
    cout<<"[";
    for(int i=0;i<n;i++){
        cout<<p[i]<<" ";
    }
    cout<<"]"<<endl;
}
void suma(int *sum,int *v1, int *v2, int n){
    for(int i=0;i<n;i++)
    {
        *(sum + i) = *(v1 + i) + *(v2 + i);
    }
}
int main()
{
    int arr[5] = {1, 2, 3, 4, 5};
    int* p = arr; // p apunta al primer elemento de arr
    int* q = p + 2; // q apunta al tercer elemento de arr (p + 2)
    int* r = q - 1; // r apunta al segundo elemento de arr (q - 1)
    cout<<q[0]<<endl;
    cout<<r[0]<<endl;
    cout<<(*p + 4)<<endl;
    q = p;
    int *s=new int[5];
    suma(s,p,q,5);
    print(s,5);
    delete s;
    return 0;
}
```

Errores de manejo de memoria

- Segmentation fault: cuando se trata de acceder una posición de memoria no reservada.
- Double free corruption: cuando se trata de liberar dos veces el mismo puntero
- No liberar memoria asignada a un puntero.
- Pedir una cantidad de memoria que no se puede asignar.

```
#include<iostream>
#include <limits>       // std::numeric_limits
using namespace std;
void print(int *p,int n){
    cout<<"[";
    for(int i=0;i<n;i++){
        cout<<p[i]<<" ";
    }
    cout<<"]"<<endl;
}
void llenar_matriz(int **m,int n,int m, int valor){
    for(int i=0;i<n;i++){
        for(int j=0;)
    }
}
int main(){
    int *p;
    //caso 1 segfault, la posicioón 100 del vertor p no existe
    //cout<<p[100]<<endl;
    // caso 2 liberar p sin haber sido asignado
    //delete p;
    p=new int[2];
    p[0] = 1;
    p[1] = 1;
    int *t=p;
    print(p,2);
    print(t,2);
    cout<<p<<endl<<t<<endl;
    delete p;
    //delete t; // ¿qué va a pasar?

    //para este ejemplo vamos a asignar mas memoria de la disponible
    //unsigned long int max = std::numeric_limits<unsigned long int>::max();
    //unsigned  int max = std::numeric_limits<unsigned int>::max();
    unsigned  int max = 5;
    cout<<max<<endl;
    int **m;
    m=new int*[max];
    for(auto i=0;i<max;i++){
        m[i]=new int[max];  //ojo al correr este código te va  a bloquear el pc
    }
    return 0;
}
```

Punteros a funciones

- Las funciones también son mapeadas en direcciones de memoria que pueden ser asignadas a una variable puntero
- El prototipo de una función es la forma que tiene la función, ósea el valor que retorna y los parámetros, para definir el puntero a función se hace con si prototipo.
- Las funciones pueden ser argumentos de otras funciones

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

int suma(int a,int b){
    return a+b;
}
void suma_vec(int *sum,int *a, int *b, int n){
    for(int i=0;i<n;i++){
        sum[i]=a[i]+b[i];
    }
}
void resta_vec(int *sum,int *a, int *b, int n){
    for(int i=0;i<n;i++){
        sum[i]=a[i]-b[i];
    }
}
void operar(void (*op)(int*,int*,int*,int),int *sum,int *a, int *b, int n){
    op(sum,a,b,3);
}

int main(){
    int (*suma_prt)(int,int);
    suma_prt = &suma;
    cout<<suma_prt(2,2)<<endl;

    void (*suma_v_prt)(int*,int*,int*,int);
    suma_v_prt = &suma_vec;

    int a[]={1,1,1};
    int b[]={2,2,2};
    int c[3];

    //suma_v_prt(c,a,b,3);
    //cout<<c[0]<<" "<<c[1]<<" "<<c[2]<<endl;

    operar(suma_vec,c,a,b,3);
    cout<<c[0]<<" "<<c[1]<<" "<<c[2]<<endl;

    operar(resta_vec,c,a,b,3);
    cout<<c[0]<<" "<<c[1]<<" "<<c[2]<<endl;
    return 0;
}
```

Ejercicios

- Hacer una función que reciba una función y calcule la integral con las sumas de Reimann
- Calcular la norma de un vector
- Sumar dos matrices doble puntero