Profesora: Ana Lilia LAUREANO CRUCES
Alumnos: Sergio Luis PÉREZ PÉREZ
Luis Eduardo URBAN RIVERO
Abraham GONZÁLEZ HERNÁNDEZ

PROGRAMACIÓN ESTRUCTURADA CON C++

Una reflexión…

- Algunas veces se rechaza C++ por ser este un lenguaje demasiado permisivo y conducente a escribir programas no legibles y difíciles de decifrar. Sin embargo, se puede intentar el uso de ciertas características del lenguaje con el fin de modelar apropiadamente el diseño de programas. Esta introducción es un intento por mantener un control sobre la escritura en código C++.

Comunicación entre módulos

- En programación estructurada el diseño es con base en abstracciones:
  - Procedurales o procedimentales y
  - Funcionales
- Y la comunicación entre estas abstracciones es a través de datos:
  - A este mecanismo se le conoce como paso de parámetros

Abstracciones Procedimentales

- Una abstracción procedimental es aquella que integra una serie de acciones sus características son:
  - Los parámetros relaccionados a la abstracción, pueden pasar por valor o por referencia. Lo anterior implica que los valores: 1) se producen, 2) se modifican, o 2) permanecen constantes.

Abstracciones Procedimentales y funcionales en C ++

- En C++ existen los dos tipos de abstracciones, su implementación es de la siguiente forma.

Formalización de abstracciones procedimentales

- Void Nombre_Proc (parámetros)
  - {
    - instrucciones;
  - }

Abstracciones Funcionales en C++

- Integra una serie de operaciones aritméticas, regresando un valor. Este último debe estar asociado algún *tipo de dato* sus características son:
  - Todos los parámetros pasan por valor.
  - Se hace hincapié en el hecho de que la función nos debe regresar un valor ya que éste se generará a través de la integración de las operaciones aritméticas.

Formalización de abstracciones funcionales

- Tipo_de_Dato Nombre_Func (parámetros)
- {
  - Tipo_de_Dato Variable;
  - Instrucciones;
  - Variable ← operaciones aritméticas;
  - return Variable;
- }

- En este caso la función tiene asociado un Tipo_de_Dato.
- Se hace hinacapié en que la Variable debe ser del mismo tipo que el asociado a la función

Paso de parámetros

- Existen dos tipos de paso de parámetros:
  - Referencia: apuntan a una localidad de memoria lo que les da la capacidad de crear o modificar los datos.
  - Valor: solo copia el contenido para ser utilizado dentro de la abstracción. Se mantiene su valor original, una vez terminadas las acciones del procedimiento.

Como se hace en C++

- En el caso de paso de parámetros por referencia se antepone un *ampersand* (&) al nombre de la variable.

- En el caso de paso de parámetros por valor, sólo se pone el nombre de la variable.

  - Por valor
  - Void Mod(int variable)

  Por referencia

  Void Mod(int &variable)

En el caso de los Arreglos...un caso especial

- siempre pasan por referencia, pensando en abstracciones procedimentales, ya que …

- En el caso de las abstracciones funcionales no tiene ningún sentido

Ejemplo de una abstracción procedural y funcional

```
void cubo (float x, &res)
{
    res=(x*x*x);
}
```

```
float cubo(float x)
{
    float res;
    res = (x*x*x)
    return res;
}
```

```
void main()
{
float n, res1, res2;
    cin>>n;
    res1 = cubo(n);
    cubo(n, res2);
    cout<< res1;
    cout<< res2;
}
```

Lenguaje C++ Fundamentos

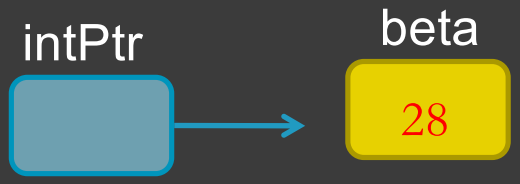
- Primer programa en C++
- // pp1.cpp
- # include < iostream.h>
- main ( )
- {

```
    cout << " hola mundo " << endl;
}
```

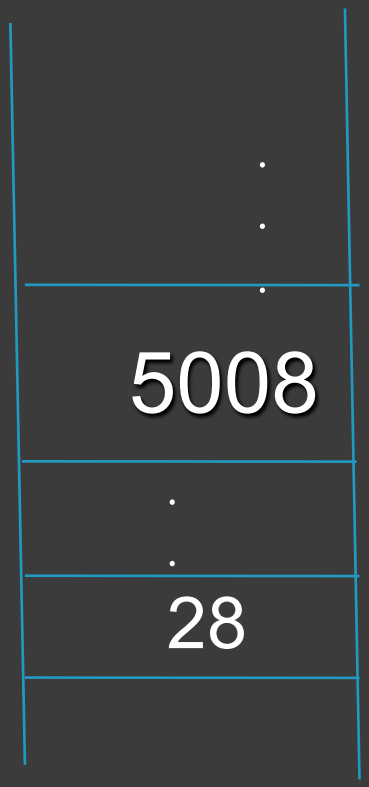
Apuntadoresy el paso de parámetros por referencia

- Los apuntadores son un tipo de datos en C++ que se clasifican como *dirección.*
  - Puntero
  - Referencia
  - ----------
    int beta
  - int* intPtr
  - Podemos hacer que intPtr apunte a beta usando el operador & (dirección de)
  - intPtr = & beta // con esta se puede accesar el contenido de beta

- Memoria



- int beta;
- int *intPtr = & beta;



- 5000 intPtr
- 5008 beta

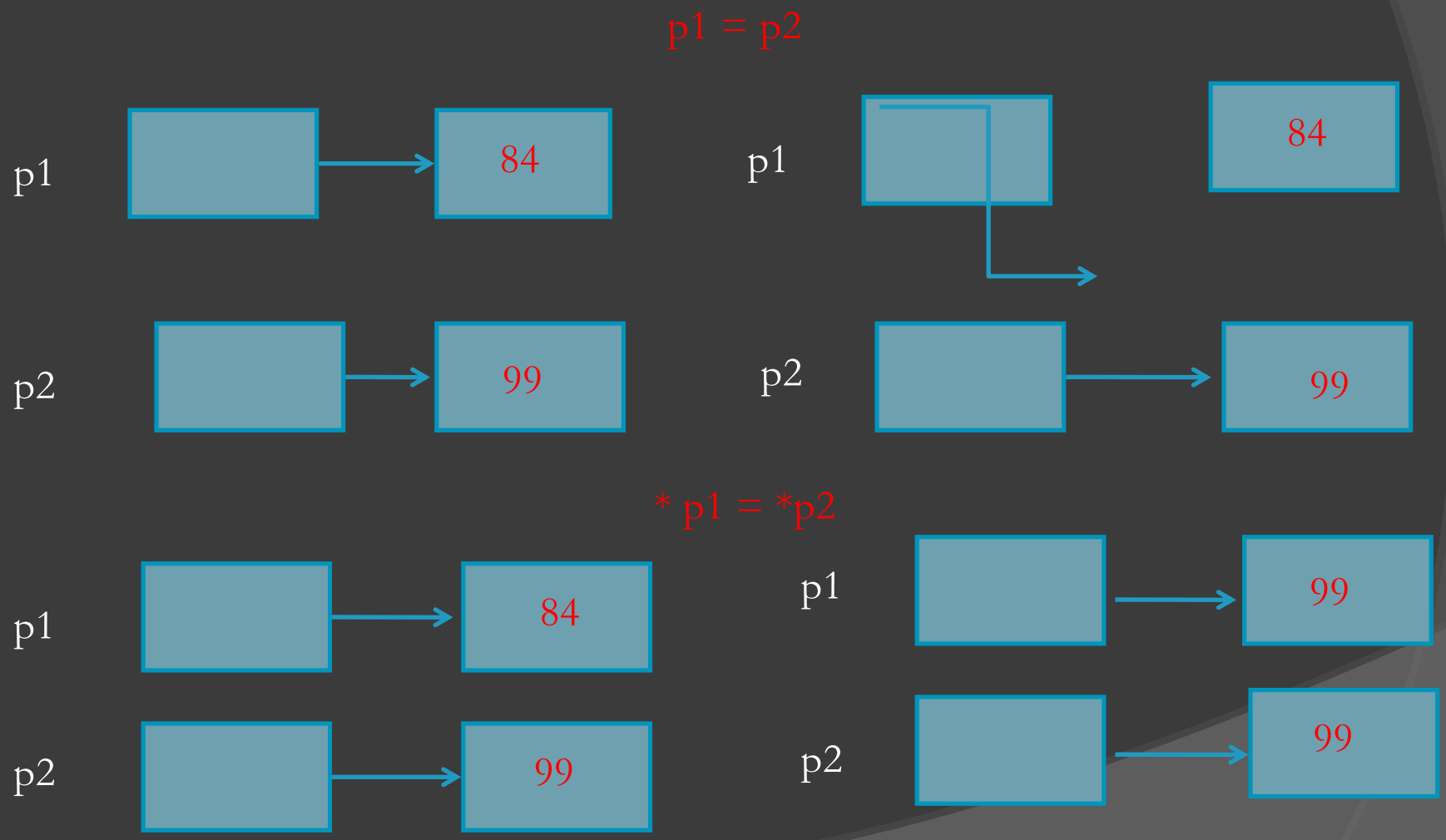
- *intPtr // denota la variable a la que apunta intPtr
- En el ejemplo *intPtr* apunta a *beta*
- Así que la siguiente sentencia *intPtr = 28;
- Guarda el valor 28 en beta. Este es un *direccionamiento indirecto.*
- La sentencia beta = 28; representa el *direccionamiento directo* a *beta*.

- *intPtr = 28;

- cout << intPtr << endl;
- cout << *intPtr << endl;

- 5008
- 28

Usos del operador de Asignación



```
//ejem1.cpp
#include<stdio.h>
#include<iostream.h>
main ( )
{
   char Res;
   int *p, q;
   Res = 's';
   WHILE  ((Res=='s') || (Res=='S'))
   {
        q = 100; // se asigna 100 a la variable entera q
        p = &q; //se asigna a p la direccion de q
        cout << *p); // se imprime el contenido de  p
        cout << endl;
        cout << "deseas desplegarlo de nuevo? (S/N) \n";
        cin  >> Res;
    }
}
```

```
//este programa nos muestra el uso de las instrucciones para utilizar
//apuntadores
//pointer1.cpp
#include<iostream.h>
main(void)
{
        int *p;  // p es un apuntador a datos de tipo entero
        p = new int;  //asigna memoria para un entero
        // simpre se debe verificar si la asignación fue exitosa
        if (!p)
        {
              cout<<"fallo la asignacion";
              return 1;
        }
        *p = 20; //asigna a esa memoria el valor de 20
        cout<<"el valor depositado en la dirección apuntada por p es ";
        cout<< *p;  //prueba que funciona mostrando el valor
        delete p; // libera la memoria
        return 0;
}
```

Paso de parámetros por valor y por referencia en C++

- En una abstracción funcional todos los parámetros por default son por valor y la función regresa un valor como resultado.

- En una abstracción procedural los parámetros pueden pasar por valor y/o por referencia.

Paso de parámetros por valor en C++

```
#include <iostream.h> // paso por valores
void swap (int a,int b)
{
  int  temp;
  temp = a;
  a = b;
  b = temp;
  cout <<"Al final y dentro de swap: i = " << a << "  j = " << b << "\n";
}
Main ( )
{
     int i = 421, j = 53;
     cout <<"antes:  i = " << i << " j = " << j << "\n";
     swap(i,j);
     cout <<"despues:  i = " << i << " j = " << j << "\n";
}
```

Paso de parámetros por referencia en C++

```
#include <iostream.h> //  paso por referencia
void swap (int &a,int &b)
 {
  int  temp;
  temp = a;
  a = b;
  b = temp;
 cout <<"Al final y dentro de swap: i = " << a << "  j = " << b << "\n";
 }
main ( )
 {
 int i = 421, j = 53;
 cout <<"antes:  i = " << i << " j = " << j << "\n";
 swap(i,j);
 cout <<"despues:  i = " << i << " j = " << j << "\n";
 }
```

Paso de parámetros por referencia en C++: otra forma

```
#include <iostream.h> //  paso por referencias
 void swap(int *a, int *b); // a y b es una variable de tipo apuntador a entero
 main()
 {int i = 421, j = 53;
  cout <<"antes:  i = " << i << " j = " << j << "\n";
  swap(&i,&j); // se pasan las direcciones de las localidades i y j
  cout <<"despues:  i = " << i << " j = " << j << "\n";
 }
 void swap(int *a,int *b)
 {int  temp;
  temp = a;
  a = b;
  b = temp;
  cout <<"Al final y dentro de swap: i = " << a << "  j = " << b << "\n";
```

```
//refer.cpp
// programa que ejemplifica el paso de parámetros por valor
// y por referencia
# include <iostream.h>
int q, w;
void cambia(int x, int y)
{
int temp;
temp=x;
x=y;
y=temp;
}
void cambia1( int *a, int *b)
{
  int t;
  t = *a;
  *a=*b;
  *b= t;
}
main() {
   cout<<"dame un valor entero";
   cin>> q;
   cout<<"Dame un valor entero ";
   cin>> w;
   cout<<"Escribo los valores antes de mandarlos";
   cout<<"q= "<<q<<" "<<"w= "<<w<<endl;
   cambia(q,w);
   cout<<"escribo los valores despues de haberlos enviado a cambia";
   cout<<q<<" "<<w<<endl;
   cambia1(&q,&w);// intercambio de sus valores
   cout<< "escribo los valores despues de haberlos enviado a cambia 1";
   cout<<q<<" "<<w<<endl;
   return 0;
}
```

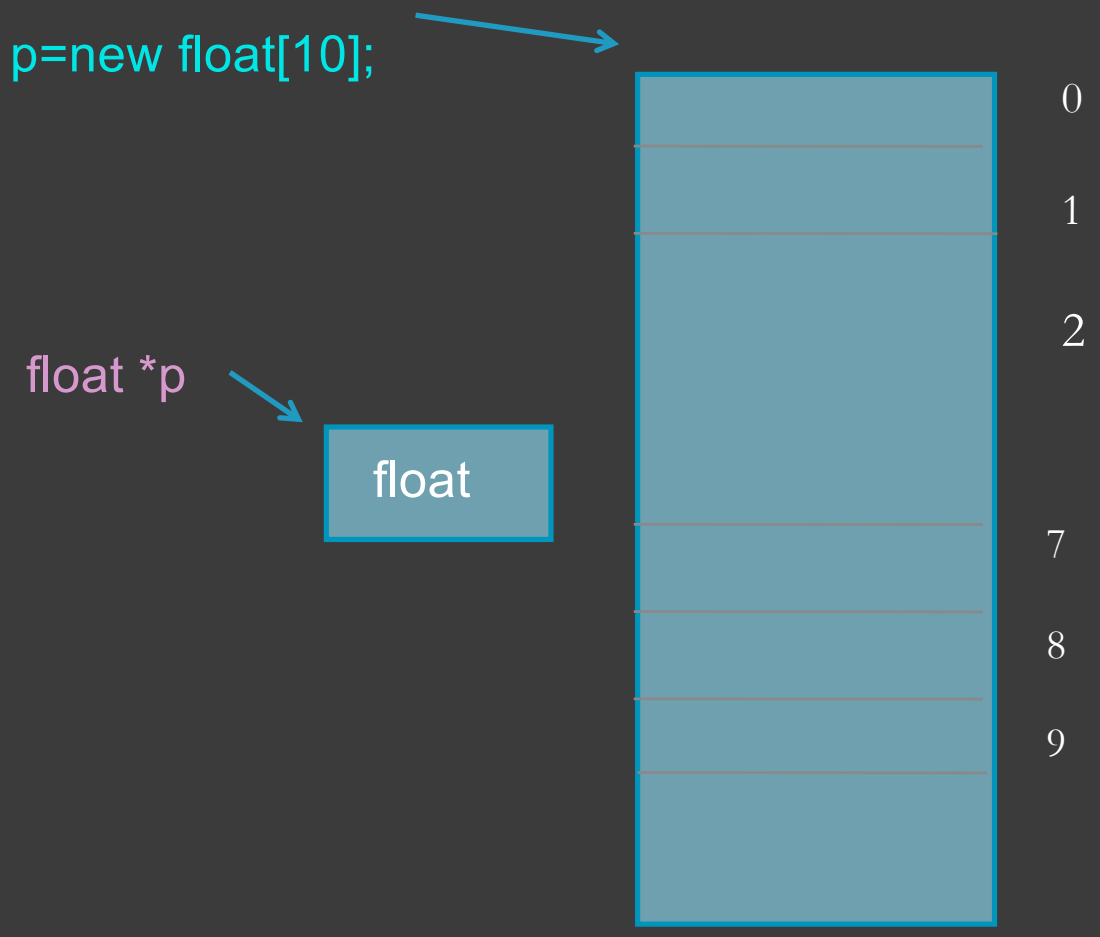
Para manejo dinámico de memoria (New y Delete)

- New:
  - asigna memoria para un objeto del tipo y tamaño especificados.
  - Devuelve un apuntador a un objeto del tipo especificado, que hace referencia al espacio reservado.
- Delete:
  - libera la memoria reservada para un objeto.
  - Sólo se puede aplicar a apuntadores que han sido retornados por el operador new.

El manejo del tamaño en new

- En el caso de arreglos se especifica explícitamente.
- En otros casos viene definido por el tipo.
  - El tipo puede ser definido por el usuario.

```
//programa que utiliza la asignación de arrays utilizando new
//este programa realiza lo mismo que el anterior pero de una manera
//distinta
//array2.cpp
#include<iostream.h>
main(void)
{
        float *p; // la variable a la que apunta p es de tipo flotante
        int i;
        p = new float[10]; //es un arreglo de 10 apuntadores
        if (!p) {
                cout<<"fallo en la asignacion\n";
                return 1;
        }
        // asigna valores de 100 a 109
        for (i=0; i<10; i++) p[i]=100.00 +i; // asigna contenidos al array. Y lo que esta
                                             moviendo es el apuntador
        for (i=0; i<10; i++) cout <<p[i]<<" "; // muestra los contenidos del array
        delete [10] p; // borra el arreglo de 10 apuntadores
        return 0;
}
```


p=new float[10];
float *p
float
0
1
2
7
8
9

Tipos definidos por el usuario

- ESTRUCTURAS (o Registros)

```
struct date_s {
    int day, month, year;
  };
```

Podemos definir variables de este tipo referenciando la estructura por el nombre:

```
date_s   AnotherDate;
```

Lenguaje C++
Tipos definidos por el usuario

- Enumeraciones
  - enum_nombre {lista_de_elementos}
  - Así como lo indica el formato a la lista de elementos se le asignan valores enteros secuencialmente iniciando por cero o los que el usuario asigne:
  - enum colores { negro, rojo, verde, azul}
  - enum colores { negro = 2, rojo, verde, azul}

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

  struct persona
              { int edad;
          string nombre;
          string apellido;
        };
  main ()
{
   persona emi;
   cout<<("dime tu nombre")<<endl;
       cin>>emi.nombre;
   cout<<emi.nombre;
 }
```

Lenguaje C++
unos operadores especiales

Los operadores de incremento ++ y -- pueden ser explicados por medio del siguiente ejemplo. Si tienes la siguiente secuencia de instrucciones

a = a + 1; b = a;

Se puede usar

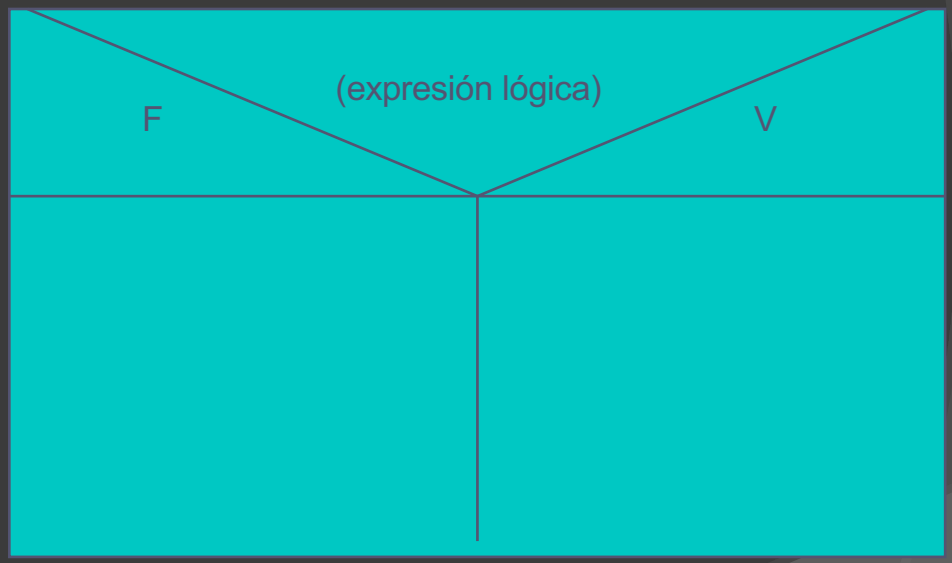
b = a++; // asigna el valor de *a* a *b* y después lo incrementa

b = ++a; // incrementa el valor de *a* y después lo asigna a *b*

Estructuras de Control en C++: Selecciones

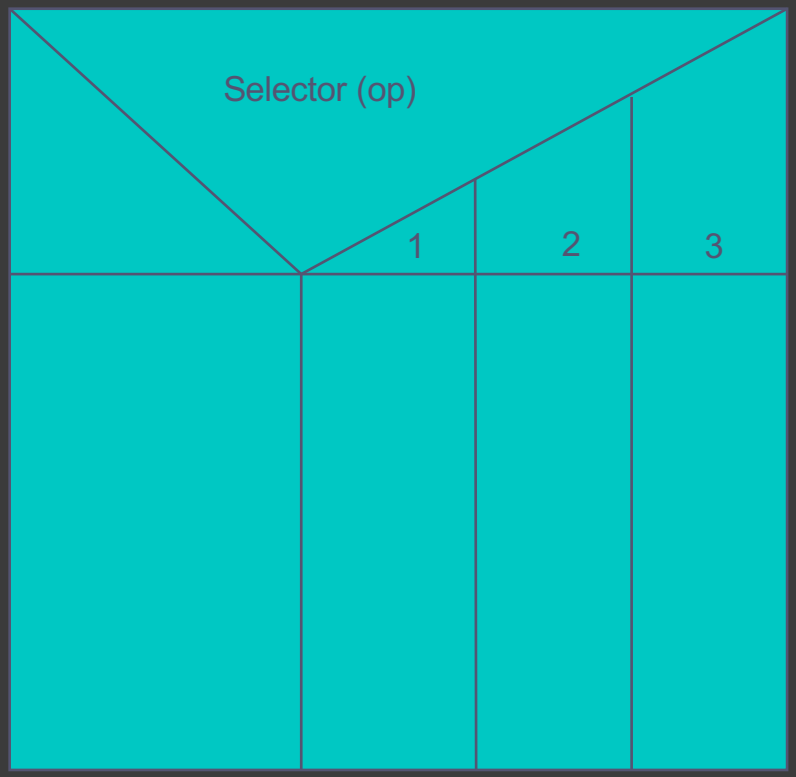
```
if (expresión lógica)
{ /*Verdadero*/
   . . .
}
else
{ /*Falso*/
   . . .
}
```

Selección Sencilla



```
switch (op)
{
  case 1:
  {
    …
  }break;
  case 2:
  {
    …
  }break;
  case 3:
  {
    …
  }break;
  default:
  {
    …
  }break;
}
```

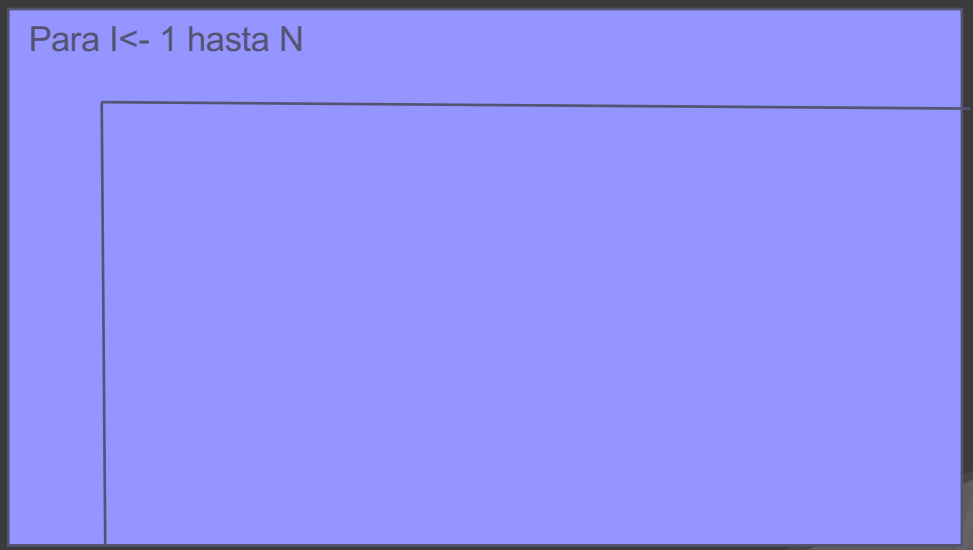
Caso



Estructuras de Control en C++: Iteración: NO Condicional

```
for (i = 1; i <= N; i++)
{
   .
   .
   .
}
```

Progresión Aritmética



Estructuras de Control en C++: Iteración: Condicionales

```
while (expresión lógica = v)
{
   .
   .
   .
}
```

```
do
{
   .
   .
   .
} while (expresion lógica = V)
```

NOTA: C++ no cuenta de forma explícita con un REPITE_HASTA, se maneja como DO-WHILE.

Implica desarrollar una expresión lógica complemento de la diseñada en código neutro.

Un ejemplo de la Vida …

fin