Algoritmos y Estructuras de Datos I

Trabajo Práctico N°10.1	Unidad 10
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo individual
Metodología de Desarrollo: acordar	**Metodología de Corrección:** acordar docente
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

PUNTEROS
MARCO TEÓRICO

1. ¿Explicar con un ejemplo las Direcciones de memoria (tomar un Integer para el Ejemplo)

En memoria los datos se almacenan en varias celdas consecutivas a partir de una dirección base
Las direcciones de memoria son ubicaciones específicas en la memoria donde se almacenan datos. Por ejemplo, si declaramos int i = 5;, el valor 5 se almacena en una ubicación de memoria que ocupa cuatro bytes. La dirección base de esta ubicación, por ejemplo 0F03:1A38, se usa para acceder al valor 5 en memoria. Un puntero int *punt puede almacenar esta dirección para manipular el valor de i indirectamente.

2. ¿Qué entiende por punteros, que representan?

Los punteros son variables que almacenan la dirección de memoria de otros datos, permitiendo así el acceso indirecto a dichos datos. Son útiles para manejar arreglos, memoria dinámica y mejorar el rendimiento al acceder directamente a la memoria.

3. Que utilidades tienen los Operadores Monarios “*” y “&;”?

El operador & permite obtener la dirección de memoria de una variable, mientras que el operador * permite acceder al valor almacenado en esa dirección
Por ejemplo, si int i; y int *punt;, entonces punt = &i; almacena la dirección de i en punt, y *punt accede al valor de i.

4. ¿Qué direcciones válidas (valores) tengo para los Punteros?

Un puntero debe tener una dirección válida para evitar errores. Al declararlo, un puntero no tiene una dirección válida automáticamente, por lo que es común asignarle una dirección mediante & o usar NULL para indicar que no apunta a nada.

5. ¿Cuál es el Problema de no inicializar un Puntero?

Un puntero no inicializado contiene una dirección desconocida, lo cual puede llevar a errores graves si intenta acceder o modificar datos de manera inesperada, lo que podría causar fallos en el programa.

6. Por qué decimos que es Un valor seguro: NULL?

Asignar NULL a un puntero indica que no apunta a nada, evitando errores al intentar acceder a datos inválidos. Si se intenta acceder a un puntero NULL, el error es más fácil de detectar y corregir en depuración.

7. Que entiende por Copia y comparación de punteros

Al copiar un puntero en otro ambos apuntan al mismo dato, lo que permite acceder y modificar dicho dato desde cualquiera de los dos punteros.
En la comparación de punteros es posible determinar si apuntan al mismo dato utilizando operadores relacionales == y != Solo pueden compararse punteros del mismo tipo base

8. Que entendemos por Tipos puntero (piense en tipos de datos)

Un tipo puntero se declara en función del tipo de dato al que apunta, lo que permite utilizar punteros con distintos tipos base.
Por ejemplo, int * apunta a un entero, char * a un carácter, y double * a un doble. Esto permite manipular distintos tipos de datos mediante punteros y realizar las mismas operaciones que se harían con el dato directamente.

9. Dar ejemplos de Punteros a estructuras

Para acceder a una estructura a través de un puntero, primero se declara la estructura y luego se crea un puntero que apunte a ella.
Por ejemplo, si tRegistro es una estructura con campos codigo, nombre, y sueldo, se puede crear un tipo de puntero tRegistro * que permita acceder a sus campos usando -> (ej., puntero->nombre).

10. Diferencie con un ejemplo Punteros a constantes y punteros constantes

Un puntero a constante (const tipo *puntero) apunta a un valor que no se puede modificar, pero la dirección que contiene el puntero sí puede cambiar.
Ejemplo: const int *punt_a_cte = &entero1; donde *punt_a_cte = 5 es válido pero (*punt_a_cte)++ no lo es, ya que el valor no puede modificarse.
A diferencia, un puntero constante (tipo *const puntero) tiene una dirección fija, pero el valor en esa dirección puede modificarse.
Ejemplo: int *const punt_cte = &entero1; donde punt_cte no puede apuntar a otra dirección aunque sí puede modificar el valor

11. Que relación hay entre Punteros y paso de parámetros (relacione referencia y valor)

Los punteros permiten simular el paso por referencia en C++, permitiendo que una función reciba la dirección de una variable y modifique su valor original. Sin embargo, en C++, también se puede usar & para pasar por referencia sin necesidad de punteros.

12. Explicar la relación entre Punteros y Arrays

Los punteros y los arrays en C++ están estrechamente relacionados ya que el nombre de un array es un puntero constante que apunta al primer elemento del array. Esto quiere decir que, los punteros y arrays están relacionados porque el nombre de un array actúa como un puntero al primer elemento del array. Por ejemplo, *dias accede al primer elemento de dias. Cuando se pasa un array a una función, se pasa en realidad un puntero a su primer element

13. Explicar el “Esquema de Memoria y datos del programa”

La memoria en C++ se divide en el stack y el heap.
Stack: Se utiliza para variables locales de funciones y se gestiona automáticamente, creándose y destruyéndose cuando se entra y sale de las funciones.
Heap: Se usa para la memoria dinámica, que el programador debe liberar manualmente, el programador, quien debe liberar manualmente esta memoria cuando ya no se necesita.
Los punteros permiten asignar memoria en el heap y acceder a los elementos de arrays o pasar funciones.

14. Que entiende por Memoria dinámica? ¿Que uso tiene?

La memoria dinámica en C++ se refiere a la memoria que se asigna y libera durante la ejecución del programa en el heap, de acuerdo con las necesidades específicas en cada momento. Esto permite almacenar grandes volúmenes de datos o estructuras que pueden exceder la capacidad de la memoria estática. La gestión de esta memoria está a cargo del Sistema de Gestión de Memoria Dinámica, que reserva espacio en el heap y devuelve su dirección base cuando se solicita memoria. Esta memoria debe liberarse cuando ya no es necesaria para evitar fugas. La memoria dinámica es útil cuando el tamaño o cantidad de datos cambia con el tiempo, ajustándose a las demandas del programa sin desperdiciar recursos.

MARCO PRÁCTICO

Tomando como Marco el Programa de la Unidad 7 (copiado al final) modificando Modularizándolo mediante el uso de Bibliotecas.
Tener en Cuenta:

- Modularizar el Programa.
- Proteger contra Inclusiones Múltiples.
- Aplicar Espacios de Nombres.
- Aplica apropiadamente los conceptos de abstracción, encapsulación y ocultamiento de información.
- Realiza una apropiada distribución de responsabilidades entre las entidades del espacio de la solución.
- Desarrolla para reusar.
- Reusa apropiadamente las entidades desarrolladas en el espacio curricular.
- Demuestra un uso apropiado de la sintaxis y semántica del lenguaje de programación C++.
- Reusa apropiadamente las entidades desarrolladas en el espacio curricular.
- Demuestra un uso apropiado de la sintaxis y semántica del lenguaje de programación C++.

COPIA UNIDAD 7
Desarrollar un Programa que:

1. Sin ejecutarlo, ¿qué mostraría el siguiente código?

Aca inicia X con valor 5, Y con el valor 12 y Z en espera.
int x = 5, y = 12, z;
Se introduce los punteros
int *p1, *p2, *p3;
el valor de referencia de &x se le asigna a la memoria dinámica de p1
p1 = &x;
el valor de referencia de &x se le asigna a la memoria dinámica de p1
p2 = &y;
El valor de puntero P1 y el valor de P2 dara un producto que es el valor asignado de z
z = *p1 * *p2;
el valor de referencia obtenido entre el producto de P1y deP2 es Z y se le asigna a la memoria dinámica de P3
p3 = &z;
El valor asignado en la memoria incrementa en uno
(*p3)++;
se iguala el valor de la memoria dinámica P3 a P1
p1 = p3;
Se muestra en pantalla los valores a los que apunta P1, P2, P3
cout<<*p1 <<" "<<*p2<<" "<< *p3;

2. ¿Qué problema hay en el siguiente código?

Hay un problema en la declaración del puntero, porque:
se introduce el valor 5 al iniciar el dato
int dato = 5;
El P1 toma el valor de int. El P2 no toma el valor de int porque no es un puntero
int *p1, p2;
el valor de referencia de &dato se le asigna a la memoria dinámica de p1
p1 = &dato;
Acá no hay puntero, por lo que no se iguala el valor de la memoria dinámica P1 a P2.
p2 = p1;
No tiene valor de referencia
cout << *p2;

3. ¿Qué problema hay en el siguiente código?

Se declara y se les asigna valores a D, E y F
double d = 5.4, e = 1.2, f = 0.9;
Se introducen la declaración de punteros a double
double *p1, *p2, *p3;
el valor de referencia de d se le asigna al P1
p1 = &d;
El valor de referencia de P1 se incrementa en 3, y se le asigna a p1
(*p1) = (*p1) + 3;
el valor de referencia de e, se le asigna a la memoria dinámica de p2
p2 = &e;
el valor de referencia de p1 y p2 se suma, y se le asigna a p3 pero aca esta el problema porque no inicializa y no toma el valor de f
(*p3) = (*p1) + (*p2);
Se muestra el valor de p1, p2 y p3.
cout<<*p1 <<" "<<*p2<<" "<< *p3;

4. ¿Cómo declararías un puntero constante p para apuntar a una constante entera? (Repasa las diapositivas 877-878 de la presentación del tema.)

const int *const p;
const int *const p = &valor;

5. Dado el siguiente tipo:
typedef struct {
string nombre;
double sueldo;
int edad;
} tRegistro;
Y el siguiente subprograma:

void func(tRegistro &reg, double &irpf, int &edad) {
const double TIPO = 0.18;
reg.edad++;
irpf = reg.sueldo * TIPO;
edad = reg.edad;
}
Reescribe el subprograma para que implemente el paso de parámetros por variable con punteros,
en lugar de las referencias que usa ahora (modifica el prorotipo y la implementación
convenientemente).

Lic. Oemig José Luis.