Manual de prácticas de laboratorio.

Construir programas de acuerdo a las problemáticas dadas.

Unidad de aprendizaje: Programación estructurada.

Datos de identificación.

Programa educativo:	Licenciatura en Informática Administrativa.
Programa de estudios por	
Competencias:	Programación estructurada.
Unidad de competencias I:	Identificar la metodología de programación estructurada para desarrollar la solución de un problema utilizando métodos en un enfoque moderno usando RUP o XP, Objetos y Eclipse.
Créditos de la U.A:	8.
Subtemas:	* Identificar la metodología de programación estructurada. * Describir los elementos básicos del lenguaje estructurado. * Implementar las diversas formas de control básico de un programa mediante estructuras de selección y repetición, funciones, punteros y procedimientos para desarrollar programas. * Analizar los conceptos de arreglos y apuntadores para formular programas para satisfacer las necesidades de las pequeñas y medianas empresas. * Aplicar el uso de archivos y memoria dinámica para el manejo de la información.
U.A impartida en:	CU UAEM Ecatepec
Realizado por:	**Dra. Patricia Delgadillo Gómez Dra. Adriana Mercedes Ruiz Reynoso Y Dr. Salvador Juárez López**
Fecha:	Agosto del 2017 semestre A

Índice:

Presentación.

El modelo de innovación curricular tiene como objetivo formar profesionistas universitarios con conocimientos y valores; hábiles en el uso de las nuevas tecnologías de la información y la comunicación, innovadores, propositivos, proactivos, emprendedores, con conocimiento de lenguas, gestores de su movilidad nacional e internacional y capacidad de comprender y resolver problemas en las pequeñas y grandes empresas.

Para aprovechar una manera adecuada la unidad de aprendizaje, es conveniente que el alumno se familiarice con un lenguaje de programación de alto nivel, como pascal o C en el que comprenda los principales fundamentos como: variables, expresiones, métodos, estructuras de decisión e iteración que permita organizarlas con miras a promover eficientemente las competencias profesionales y evaluarlas.

Una de las actividades primordiales del Docente es provocar en el alumno:

- Aprendizaje activo: La participación activa del alumno dentro del proceso de la unidad de aprendizaje.
- Desarrollo de habilidades: Muchas de las competencias necesarias para resolver problemas usando un lenguaje de programación se generan a partir del uso reiterado de una técnica o metodología.

El Informático Administrativo será capaz de comprender íntegramente los problemas administrativos y productivos de cualquier organización. Analizar, diseñar, implementar, administrar y evaluar sistemas de información manuales y computarizados en la grande, mediana y pequeña empresa. En este sentido, este manual de ejercicios desarrollado conforme al programa de estudios por competencias a través del docente, busca provocar al alumno del desarrollo de habilidades que no solo logren los conocimientos curriculares, que puedan aplicarlos correctamente a problemas concretos, sino que también desarrollen habilidades y comportamientos de trabajo en equipo colaborativa y de investigación, por medio del aprendizaje activo; de esta manera. El manual está conformado por 5 unidades de aprendizaje, en la unidad uno se establece una metodología en la programación estructurada para dar solución a problemas, la unidad dos se basa en que el alumno aprenda la lógica de la programación para diseñar programas cotidianos, también en la unidad tres se destaca que el alumno utilizará las estructuras de selección, repetición, funciones y procedimientos, por otro lado la unidad cuatro hay que hacer notar que se aplican los arreglos y punteros para la creación de programas, por último tenemos la unidad cinco que se basa en el manejo de archivos y memoria dinámica que permita resolver problemas empresariales.

La forma de evaluación será un portafolio de evidencias con los programas ejecutados, con la finalidad de desarrollar el aprendizaje y establecer habilidades de programación.

Estructura de la unidad de aprendizaje.

- Identificar la metodología de programación estructurada para desarrollar la solución de un problema.
- Describir los elementos básicos del lenguaje estructurado para la codificación de programas diseñados en algoritmos y/o diagramas de flujo.
- Implementar las diversas formas de control básico de un programa mediante estructuras de selección y repetición, funciones, punteros y procedimientos para desarrollar programas.
- Analizar los conceptos de arreglos y apuntadores para formular programas para satisfacer las necesidades de las pequeñas y medianas empresas.
- Aplicar el uso de archivos y memoria dinámica para el manejo de la información basados en bases de datos.

Propósito de la unidad de aprendizaje.

El modelo de innovación curricular tiene como objetivo formar profesionistas universitarios con conocimientos y valores; hábiles en el uso de las nuevas tecnologías de la información y la comunicación, innovadores, propositivos, proactivos, emprendedores, con conocimiento de lenguas, gestores de su movilidad nacional e internacional y capacidad de comprender y resolver problemas en las pequeñas y grandes empresas.

Es conveniente que el alumno se familiarice con un lenguaje de programación de alto nivel, como pascal o C en el que comprenda los principales fundamentos como: variables, expresiones, métodos, estructuras de decisión e iteración que permita organizarlas con miras a promover eficientemente las competencias profesionales y evaluarlas.

Unidad I.- Identificar la metodología de programación estructurada para desarrollar la solución de un problema.

En la unidad 1 se identificara la metodología de programación estructurada para desarrollar la solución de un problema utilizando el método top-dow, así como las fases del ciclo de vida.

<table>
<tr><td rowspan="2">UNIDAD DE COMPETENCIA I</td><td colspan="3">ELEMENTOS DE COMPETENCIA</td></tr>
<tr><td>Conocimientos</td><td>Habilidades</td><td>Actitudes/ Valores</td></tr>
<tr><td>Identificar la metodología de programación estructurada para desarrollar la solución de un problema utilizando métodos en un enfoque moderno usando RUP o XP, Objetos y Eclipse.

-</td><td>-Método descendente(top-down)
-Programación modular
-Fases del ciclo de vida de software.</td><td>-Capacidad de abstracción
-capacidad de análisis y síntesis</td><td>-Disposición para trabajo individual y en equipo
-Honestidad
-Perseverancia
-Responsabilidad</td></tr>
<tr><td>ESTRATEGIAS DIDÁCTICAS:

Exposición grupal
Demostración a través de prácticas (solución de problemas)
Desarrollo de mapas conceptuales</td><td colspan="2">RECURSOS REQUERIDOS

Pizarron
Computadora
Rotafolio</td><td>TIEMPO DESTINADO

12 horas</td></tr>
<tr><td rowspan="2">CRITERIOS DE DESEMPEÑO I</td><td colspan="3">EVIDENCIAS</td></tr>
<tr><td colspan="2">DESEMPEÑO</td><td>PRODUCTOS</td></tr>
<tr><td>Solución correcta de problemas utilizando pseudocódigo o diagrama de flujo en forma modular</td><td colspan="2">Solución de problemas utilizando el ciclo de vida del software</td><td>Programas una calculadora que sume, reste, multiplique y divida números, utilizando el paradigma de programación modular.</td></tr>
</table>

Método Descendente (Top-Down)

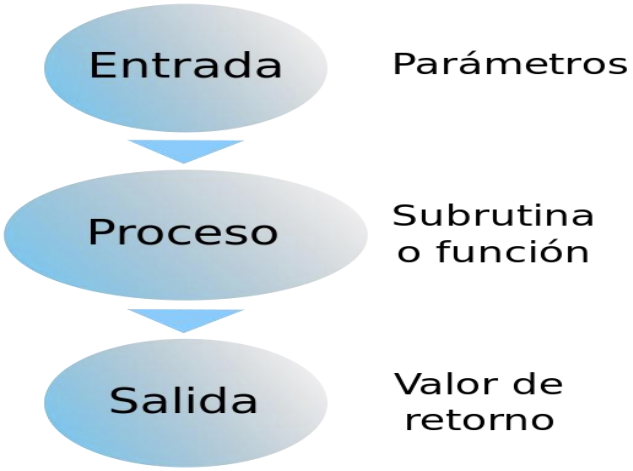
También conocido como de arriba a abajo consiste en establecer una serie de niveles de mayor a menor complejidad que den solución al problema.

Luego se crea una relación entre las etapas de la estructuración de forma que una etapa jerárquica y su inmediato inferior se relacionen mediante una interfaz claramente definida de entradas y salidas de información.

El Top-Down es muy popular por ser metodológico para la enseñanza de la programación, por favorecer la rápida creación de una estructura de diseño inicial flexible y fácil de comprender y por ser muy útil en la solución de problemas complejos.

El problema se descompone en varias estructuras jerárquicas, de forma que se pueda considerar cada estructura desde dos puntos de vista: ¿qué hace? y ¿cómo lo hace?
Las estructuras desde los dos puntos de vista se representan de la siguiente forma:

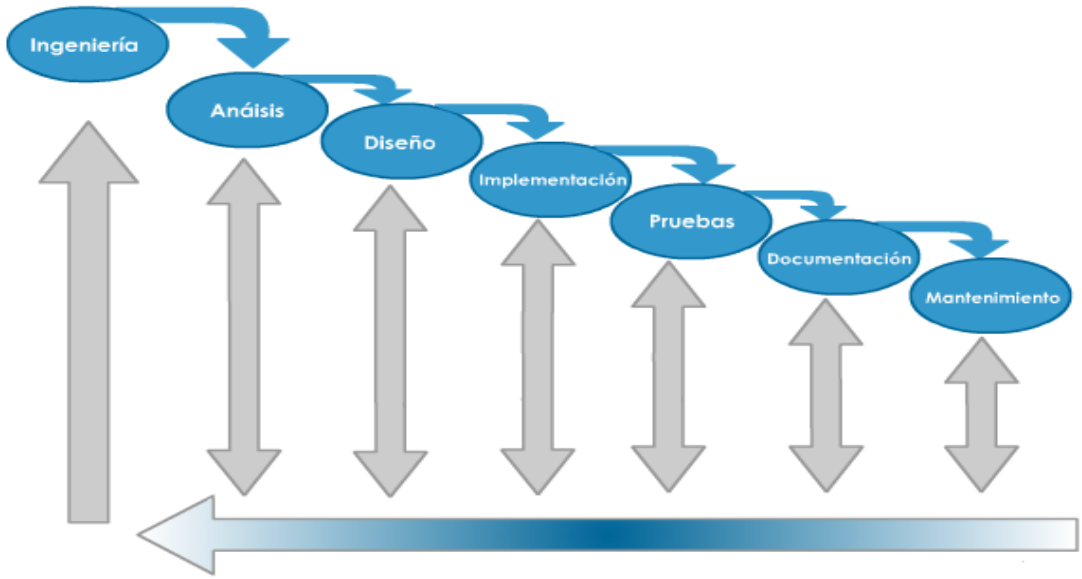
Programación Modular



La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable. (Peña, Cela 2010).

Ciclo de Vida de Software

El ciclo de vida clásico del software siendo uno de los más utilizados tal como lo plantean diferentes autores, está conformado en su versión ampliada por siete etapas que se pueden representar mediante un modelo en cascada:



- ***Ingeniería:*** En esta etapa el analista luego de un minucioso y detallado estudio de los sistemas de una organización, detecta un problema o una necesidad que para su solución y/o satisfacción es necesario realizar un desarrollo de software.

- ***Análisis***: En esta etapa se debe entender y comprender de forma detallada cual es la problemática a resolver, verificando el entorno en el cual se encuentra dicho problema, de tal manera que se obtenga la información necesaria y suficiente para afrontar su respectiva solución.
- ***Diseño***: Una vez que se tiene la suficiente información del problema a solucionar, es importante determinar la estrategia que se va a utilizar para resolver el problema. (Benet,2002).

- ***Implementación:*** partiendo del análisis y diseño de la solución, en esta etapa se procede a desarrollar el correspondiente programa que solucione el problema mediante el uso de una herramienta computacional determinada.
- ***Pruebas:*** Los errores humanos dentro de la programación de los computadores son muchos y aumentan considerablemente con la complejidad del problema. Cuando se termina de escribir un programa de computador, es necesario realizar las debidas pruebas que garanticen el correcto funcionamiento de dicho programa bajo el mayor número de situaciones posibles a las que se pueda enfrentar.
- ***Documentación***: Es la guía o comunicación escrita en sus diferentes formas, ya sea en enunciados, procedimientos, dibujos o diagramas que se hace sobre el desarrollo de un programa. La importancia de la documentación radica en que a menudo un programa escrito por una persona, es modificado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

La documentación se compone de tres partes:

a) Documentación Interna: Son los comentarios o mensajes que se añaden al código fuente para hacer más claro el entendimiento de los procesos que lo conforman, incluyendo las precondiciones y las pos condiciones de cada función.

b) Documentación Externa: Se define en un documento escrito con los siguientes puntos

- Descripción del Problema
- Datos del Autor
- Algoritmo (diagrama de flujo o Pseudocódigo)
- Diccionario de Datos
- Código Fuente (programa)

c) Manual de Usuario: Describe paso a paso la manera cómo funciona el programa, con el fin de que el usuario lo pueda manejar para que obtenga el resultado deseado. (Benet,2002).

- ***Mantenimiento***: una vez instalado un programa y puesto en marcha para realizar la solución del problema previamente planteado o satisfacer una determinada necesidad, es importante mantener una estructura de actualización, verificación y validación que permitan a dicho programa ser útil y mantenerse actualizado según las necesidades o requerimientos planteados durante su vida útil. Para realizar un adecuado mantenimiento, es necesario contar con una buena documentación del mismo.

Para terminar de entender la problemática es importante tener conceptos claros y precisos de lo que es el Análisis y el Diseño de Algoritmos, para resolver problemas y cubrir las necesidades del cliente. (Benet,2002).

Criterio de desempeño de la unidad I:	**Actividades de comprobación:**	**Actividades de evaluación :**
Solución correcta de una problemática utilizando el pseudocódigo o diagrama de flujo modular	Solución de problemas utilizando el ciclo de vida	Ejecución del programa sin ningún tipo de error (lógico, síntesis, regresión) Elaborar un programa que realice una suma, resta, multiplicación.

Instalación del software.

Dev C++

Es importante conocer cómo se instala c++ dev con la cual se trabajaran los ejercicios para la unidad de aprendizaje de programación estructurada.

Requisitos del sistema.

Requerimientos mínimos Dev C++:

- Microsoft Windows 95, 98, NT 4, 2000, XP, Vista, 7, 10.
- 8 MB de RAM con un archivo de intercambio grande.
- 30 MB de espacio libre en el disco duro.

Requerimientos recomendados Dev C++:

- Microsoft Windows 2000, XP, Vista, 7, 10.
- 32 MB de RAM con un archivo de intercambio grande.
- 200 MB de espacio libre en el disco duro.
- Arquitectura de 32 o 64 Bits.

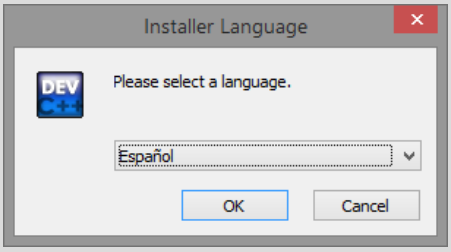
Materiales:

- Computadora.
- Dev C++.

Pasos a seguir.

1er paso:

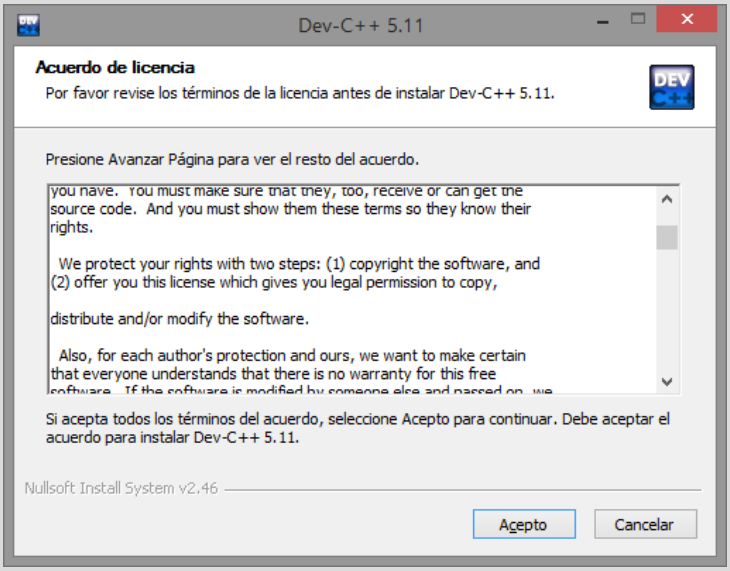
Al ejecutar el instalador de Dev C++ nos pedirá seleccionar el lenguaje:



Elegimos el más conveniente para nosotros.

2do paso:

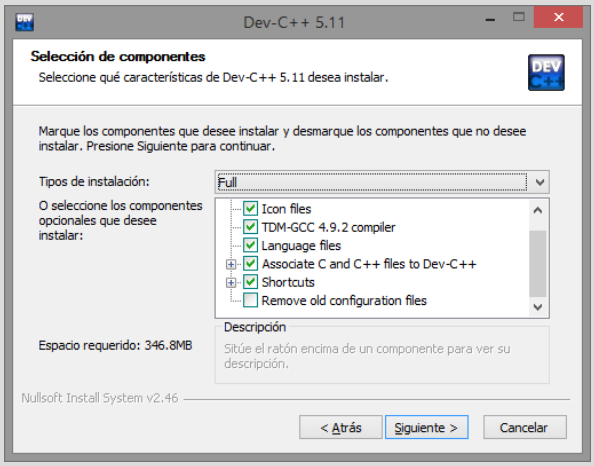
Una vez elegido el lenguaje aparecerán los términos y condiciones del uso del programa y le daremos “Acepto”.



Pasos a seguir.

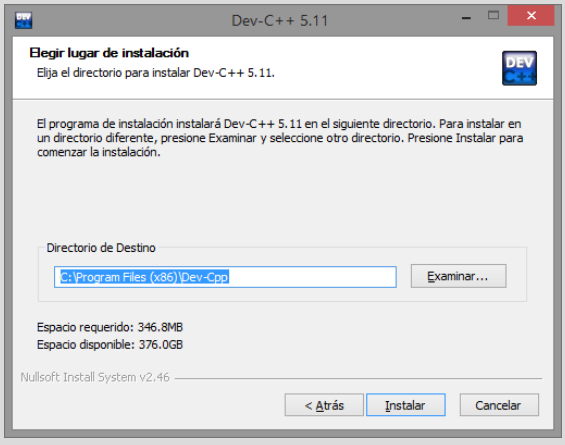
3er paso:

Seleccionaremos los componentes que se instalarán en el programa Dev C++ y nos mostrará los tipos de instalación, se recomienda instalar todos los componentes.



4to paso:

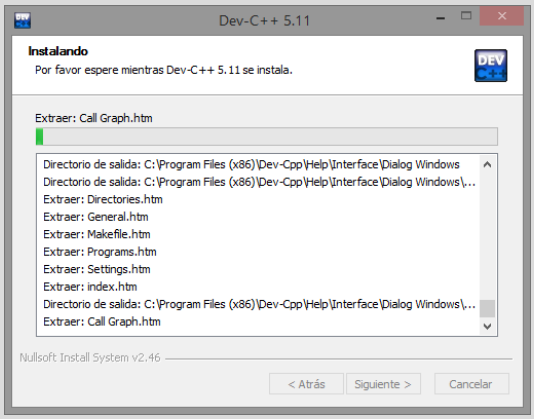
Seleccionaremos la ruta en donde lo instalaremos.



Pasos a seguir.

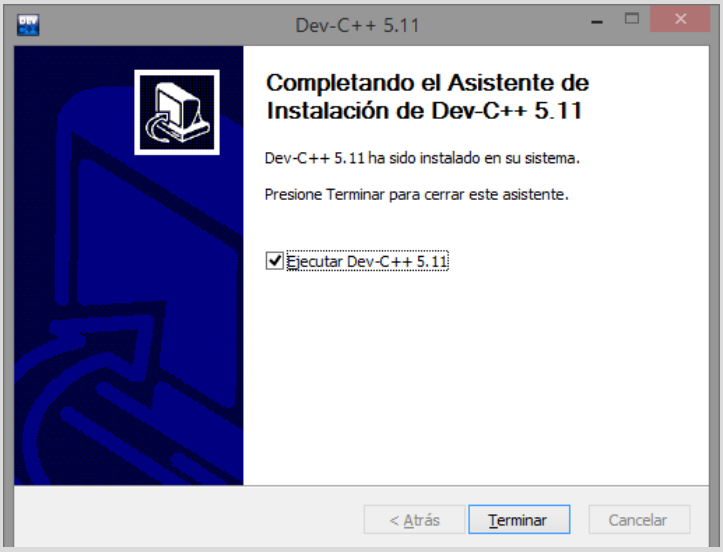
5to paso:

Comienza el proceso de instalación.



6to paso:

Una vez finalizado procedemos a ejecutarlo o terminar la instalación.



Es importante la instalación en sus computadoras para poder ejecutar los programas depurarlos, compilarlos y poder obtener la ejecución de cada uno de ellos, como se muestran las pantallas en negro en las siguientes presentaciones como comprobación de que si son ejecutables las problemática planteada.

Unidad II.- Uso de funciones y procedimientos con y sin parámetro.

En la unidad se describen los elementos básicos para el lenguaje estructurado para la codificación de programas diseñados en algoritmos como son el uso de funciones con parámetro y sin paso de parámetros, los flujos de entrada y salida Flujos de entrada y salida – printf y scanf o o cout y cin.

UNIDAD DE COMPETENCIA II	ELEMENTOS DE COMPETENCIA		
	Conocimientos	**Habilidades**	**Actitudes/ Valores**
Describir los elementos básicos del lenguaje estructurado para la codificación de programas diseñados en algoritmos y/o diagramas de flujo	Uso de funciones y procedimientos con y sin paso de parámetros -Flujos de entrada y salida Declaración y uso de arreglos unidimensionales y bidimensionales -Uso de apuntadores, direcciones y aritmética de direcciones	Asociación Análisis Creatividad Solución de problemas	Tolerancia Disponibilidad de trabajo individual y en equipo Perseverancia

ESTRATEGIAS DIDÁCTICAS:	RECURSOS REQUERIDOS	TIEMPO DESTINADO
Demostración con práctica Uso de mapas mentales Sesión bibliográfica	Pizarron Computadora Bibliografía	8 horas

Introducción.

Describir los elementos básicos del lenguaje estructurado para la codificación de programas diseñados en algoritmos y/o diagramas de flujo.

Descripción de la práctica.

En esta práctica se hará el uso de funciones con parámetro y sin parámetro.

Requisitos previos.

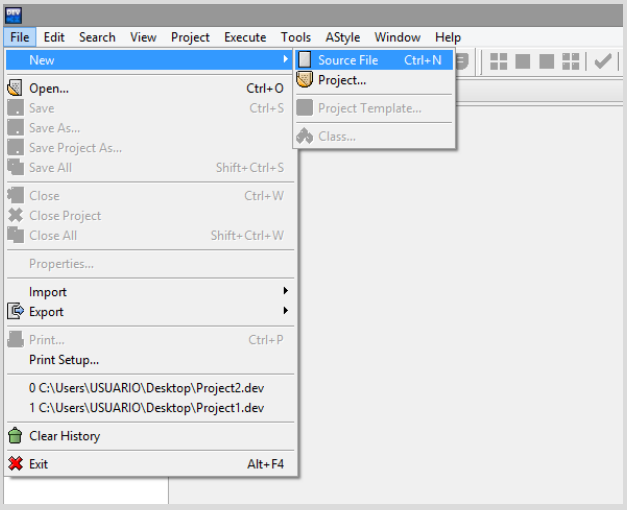
Saber el uso del lenguaje de programación C o C++ y el uso de comandos.

Materiales:

- Computadora.
- Dev C++.

Procedimientos:

Se crea un nuevo archivo para realizar el programa.



Tema: Uso de funciones con parámetros.

Ejemplo 1: Suma de 2 números con el uso de una función.

Nuestro primer programa será una suma de 2 números con el uso de parámetros como observamos:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
int suma(int a,int b)
{
    int r;
    r = a + b;
    cout<<"El resultado de la suma es: "<<r<<endl;

}
int main()
{
    int x,z;
    cout<<"Suma de 2 numeros."<<endl;
    cout<<"Introduzca el primer numero: ";
    cin>>x;
    cout<<"Introduzca el segundo numero: ";
    cin>>z;
    suma(x,z);
 system("pause");
 return 0;
}
```

1. Se declara la función con 2 parámetros de tipo entero.

2. Dentro de la función se realiza la operación.

3. Se manda a llamar a la función con sus 2 variables de sustitución.

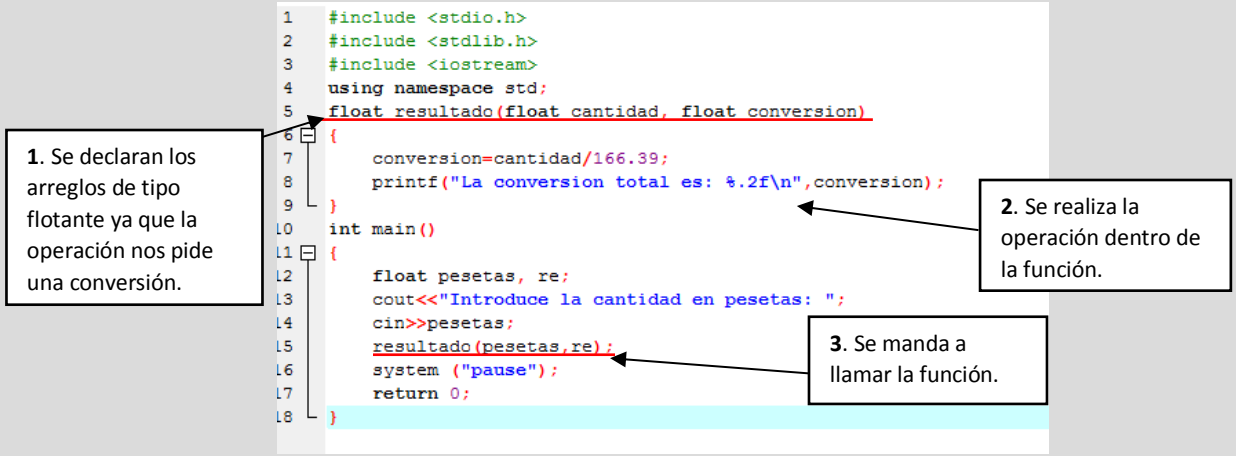
Salida:

```
Suma de 2 numeros.
Introduzca el primer numero: 10
Introduzca el segundo numero: 10
El resultado de la suma es: 20
Presione una tecla para continuar . . .
```

Tema: Uso de funciones con parámetros.

Ejemplo 2: Conversión de pesetas con el uso de una función.

Nuestro segundo programa será realizar una conversión usando una función con parámetros.




```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <iostream>
4   using namespace std;
5   float resultado(float cantidad, float conversion)
6   {
7       conversion=cantidad/166.39;
8       printf("La conversion total es: %.2f\n",conversion);
9   }
10  int main()
11  {
12      float pesetas, re;
13      cout<<"Introduce la cantidad en pesetas: ";
14      cin>>pesetas;
15      resultado(pesetas,re);
16      system ("pause");
17      return 0;
18  }
```

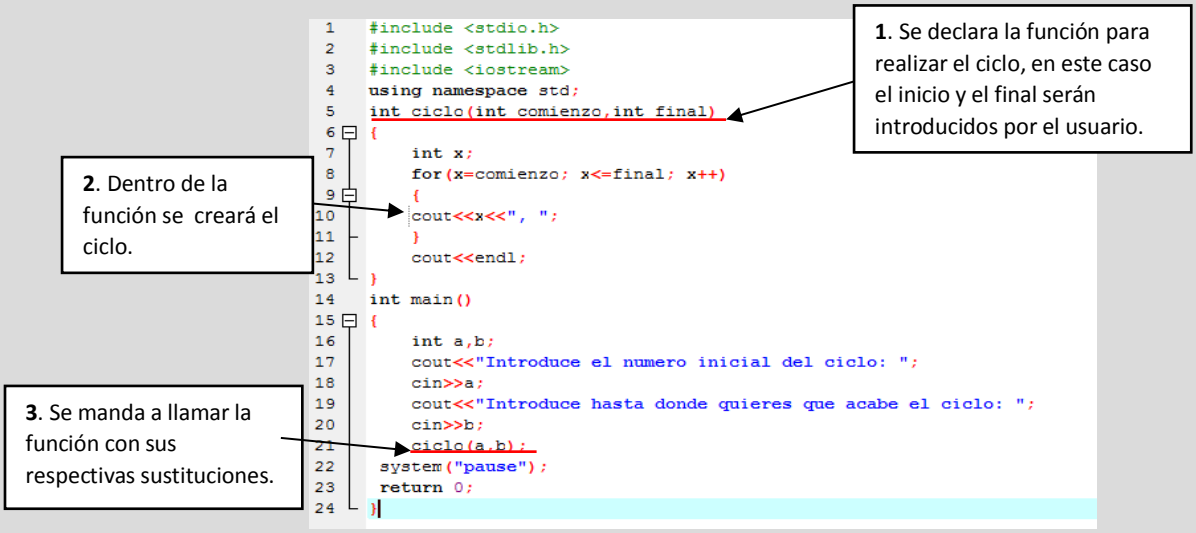
Salida:

```
Introduce la cantidad en pesetas: 100
La conversion total es: 0.60
Presione una tecla para continuar . . .
```

Tema: Uso de funciones con parámetros.

Ejemplo 3: Ciclo dentro de una función.

Nuestro tercer programa será realizar un ciclo dentro de una función con datos asignados por el usuario.




```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <iostream>
4   using namespace std;
5   int ciclo(int comienzo,int final)
6   {
7       int x;
8       for(x=comienzo; x<=final; x++)
9       {
10      cout<<x<<", ";
11      }
12      cout<<endl;
13  }
14  int main()
15  {
16      int a,b;
17      cout<<"Introduce el numero inicial del ciclo: ";
18      cin>>a;
19      cout<<"Introduce hasta donde quieres que acabe el ciclo: ";
20      cin>>b;
21      ciclo(a,b);
22   system("pause");
23   return 0;
24  }
```

Salida:

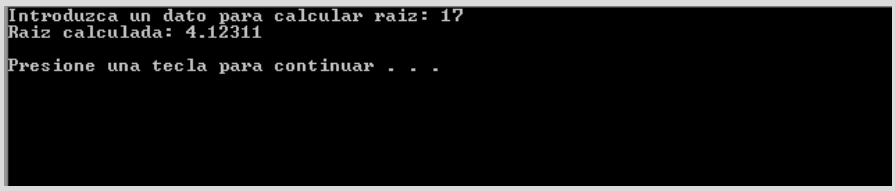
```
Introduce el numero inicial del ciclo: 2
Introduce hasta donde quieres que acabe el ciclo: 20
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
Presione una tecla para continuar . . .
```

Tema: Uso de funciones con parámetros.

Prácticas.

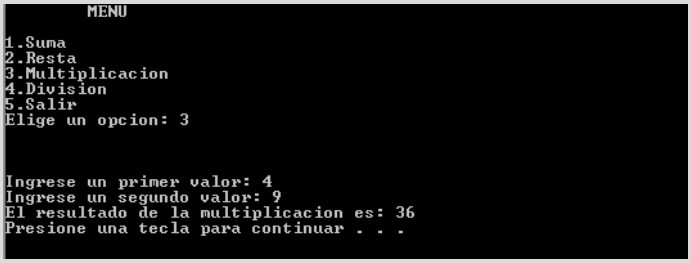
Práctica 1: Realizar un programa que calcule la raíz de un número introducido por el usuario haciendo uso de funciones con parámetros.

Salida ejemplo:



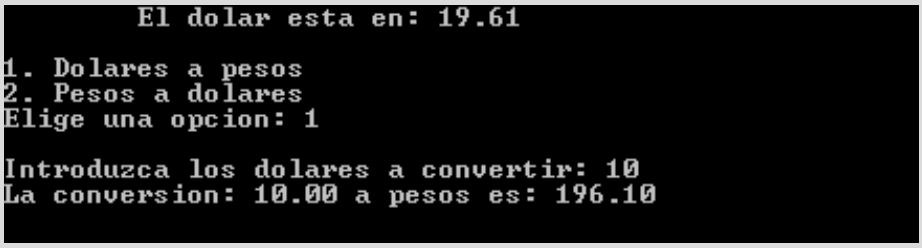
Práctica 2: Realizar un programa que tenga un menú de operaciones básicas dentro de una función con parámetros.

Salida ejemplo:



Práctica 3: Realizar un programa que tenga un menú de conversiones de **dólares** a **pesos** y de **pesos** a **dólares** utilizando una función con parámetros.

Salida ejemplo:



Tema: Uso de funciones sin parámetros.

Ejemplo 1: Programa que limpie la pantalla con una función sin parámetros.
Procedimiento:

El primer ejemplo será realizar un programa que limpie la pantalla por medio de una función sin parámetros, antes nosotros ocupamos diferentes tipos de datos para realizar una función, pero ahora como será una función sin parámetros hacemos uso del comando **Void()** como se muestra a continuación:

1. Función sin parámetro. Usamos **Void()**

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
void limpiar_pantalla()
{
    system("cls");
    cout<<"La pantalla se ha limpiado!"<<endl;
}

int main()
{
    limpiar_pantalla();
}
```

2. Mandamos a llamar a la función.

Salida:

```
La pantalla se ha limpiado!
```

Tema: Uso de funciones sin parámetros.

Ejemplo 2: Programa que mande un saludo por medio de una función sin parámetros.
Procedimiento:

El programa realizará un saludo con el uso de una función sin uso de parámetros.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
void saludo()
{
    cout<<"Hola! Bienvenido a mi programa"<<endl;
}

int main()
{
    saludo();
}
```

1. Función sin parámetro. Usamos **Void()**

2. Mandamos a llamar a la función.

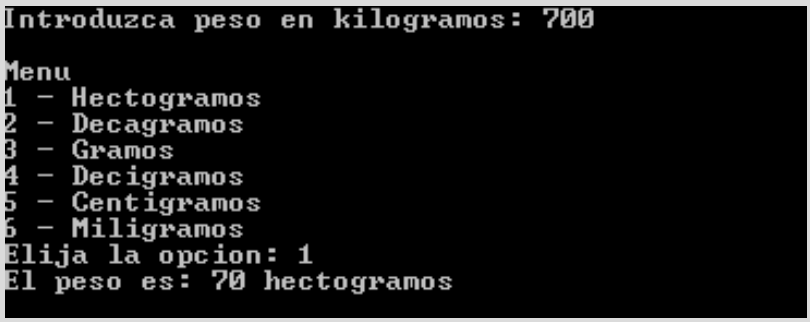
Salida:

```
Hola! Bienvenido a mi programa
```

Tema: Uso de funciones sin parámetros.

Práctica 1: Realizar un programa que contenga un menú por medio de uso de funciones sin parámetros que contenga las conversiones de kilogramos a: hectogramos, decagramos, gramos, decigramos, centigramos y miligramos y que muestre el resultado.

Salida ejemplo:



Tema: Flujos de entrada y salida – printf y scanf.

En este tema veremos cuáles son los flujos de entrada y salida existentes, aprenderemos el uso de **printf** y **scanf** ya que nos permiten la entrada y salida de datos, también el uso de la librería **<stdio.h>** que es la que nos permitirá el uso de estos comandos.

Ejemplo 1: Empezaremos con un programa en el que se le pida al usuario un número y se imprima por el uso de comandos de **printf** y **scanf**.

La sintaxis es la siguiente: **printf**(“Texto de salida”);

```c
#include <stdio.h>

main()
{
    int numero;
    printf("Introduce un numero: ");
    scanf("%d",&numero);
    printf("Tu numero es: %d",numero);
    return 0;
}
```

Salida:

```
Introduce un numero: 5
Tu numero es: 5
```

Tema: Flujos de entrada y salida – printf y scanf.

Ejemplo 3: En este ejemplo haremos uso de la variable de tipo flotante por medio de los comandos **printf** y **scanf** que pida un número en decimal y lo imprima.

```c
#include <stdio.h>

main()
{
    float numero;
    printf("Introduce un numero: ");
    scanf("%f",&numero);
    printf("Tu numero es: %f",numero);
    return 0;
}
```

1. Se utiliza el tipo de variable **float** para los números con punto decimal.

2. Utilizamos el **%f** ya que es una variable tipo carácter.

Salida:

```
Introduce un numero: 10.3
Tu numero es: 10.300000
```

```c
#include <stdio.h>

main()
{
    float numero;
    printf("Introduce un numero: ");
    scanf("%f",&numero);
    printf("Tu numero es: %.2f",numero);
    return 0;
}
```

Para que no salgan demasiados puntos decimales, podemos asignarle cuantos queremos que salgan de la siguiente manera:

```
Introduce un numero: 10.3
Tu numero es: 10.30
```

Tema: Flujos de entrada y salida – Cout y Cin.

En los programas que veremos en este manual se hará uso de diferentes flujos de entrada y salida, ya sea **printf** y **scanf** o **cout** y **cin**, en este caso también mostraremos cómo es que se usa el **cout** y **cin** y la librería que ocupa la cual es **<iostream>** estos flujos de entrada y salida son mucho más sencillos, mostraremos los siguientes ejemplos:

Ejemplo 1: Como en el ejemplo anterior realizaremos un programa que lea un número entero y lo imprima con el uso de los comandos **cout** y **cin**, también es igual con todos los tipos de variables como **float**.

La sintaxis es la siguiente: **cout**<<"Texto de salida";
cin>>variable;

```
#include <iostream>

using namespace std;

main(){
    int numero;
    cout<<"Introduce un numero: ";
    cin>>numero;
    cout<<"Tu numero es: "<<numero;

    return 0;
}
```

1. Utilizamos la librería **iostream**, note que esta no necesita el **.h** en algunos casos.

2. Este comando es muy importante ya que es el que nos permite utilizar los comandos **cout** y **cin** abreviados.

El comando **cout** nos servirá para imprimir lo que queramos en pantalla.

El comando **cin** nos permitirá leer lo que el usuario introduzca. Note que aquí no se necesita especificar si es un tipo de dato **int**, **float** o **char**, se detecta solo.

OJO:
En el **cout** que es salida los operadores son menores: **<<**, en el **cin** los operadores son mayores: **>>**

3. En la salida de la variable en la cual se guardó el dato también no se necesita especificar qué tipo de dato es ya que lo detecta en automático.

Salida:

```
Introduce un numero: 20
Tu numero es: 20
```

Tema: Flujos de entrada y salida – Cout y Cin.

Ejemplo 2: Ahora haremos un ejemplo con la variable de tipo carácter con los comandos **cout** y **cin**.

```
#include <iostream>

using namespace std;

main(){
    char ap[10];
    cout<<"Introduce tu apellido: ";
    cin>>ap;
    cout<<"Tu apellido es: "<<ap;

    return 0;
}
```

Salida:

```
Introduce tu apellido: Lopez
Tu apellido es: Lopez
```

Criterio de desempeño de la unidad II:	**Actividades de comprobación:**	**Actividades de evaluación :**
Programa estructurado en lenguaje c++ dev utilizando tipo de datos y expresiones.	Diseño de programas utilizando variables, tipos de datos, expresiones y sintaxis de c++ dev.	Ejecución correcta del programa sin ningún tipo de error (lógico, síntesis, regresión) Elaborar un programa usando tipos de datos y expresiones.

Unidad III.- Implementar las diversas formas de control básico de un programa mediante estructuras de selección y repetición, funciones, punteros y procedimientos para desarrollar programas.

En la unidad III en esta unidad se trabajara con el uso de estructuras de control de selección, así como el uso de estructuras de repetición (while, do while); así como el uso de funciones y procedimientos con y sin parámetro.

<table>
<tr><td rowspan="2">UNIDAD DE COMPETENCIA III</td><td colspan="3">ELEMENTOS DE COMPETENCIA.</td></tr>
<tr><td>Conocimientos</td><td>Habilidades</td><td>Actitudes/Valores</td></tr>
<tr><td>Implementar las diversas formas de control básico de un programa mediante estructuras de selección y repetición, funciones, punteros y procedimeintos para desarrollar programas.</td><td>*Uso de estructuras de control de selección.
*Uso de estructuras de repetición (while, do while).
*Uso de funciones y procedimientos con y sin paso de parámetros.
*Flujos de entrada y salida.</td><td>Comprensión
Comparación
Capacidad de análisis
Solución de problemas</td><td>Disponibilidad de trabjo individual y/o en equipo
Responsabilidad
Perseverancia</td></tr>
<tr><td>ESTRATEGIAS DIDÁCTICAS:

* Mapas conceptuales
*Implementación en el lenguaje correspondiente</td><td colspan="2">RECURSOS REQUERIDOS:

* Computadora
* Pizarrón
* Rotafolio</td><td>TIEMPO DESTINADO:

32 horas</td></tr>
<tr><td rowspan="2">CRITERIOS DE DESEMPEÑO III</td><td colspan="3">EVIDENCIAS</td></tr>
<tr><td>DESEMPEÑO</td><td colspan="2">PRODUCTOS</td></tr>
<tr><td>Programa estructurado que contenga las estructuras de control (if, case, while, do-while), funciones y procedimientos.</td><td>Desarrollo de programas con el uso de estructuras de control, procedimientos y funciones.
Desarrollo de programas con el uso de las diferentes metodologías.</td><td colspan="2">Ejecución correcta de los programas estructurados en C que utilicen estructuras de selección, repetición, funciones y procedimientos.</td></tr>
</table>

Tema: Uso de estructuras de control de selección.

Las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa. En estos primeros ejemplos empezaremos con la estructura de control **IF**.

Ejemplo 1: Programa que le pida al usuario su edad y que se determine si tiene permiso para votar o no mediante una estructura de control **IF**.

1. Se declara una variable de tipo entero en la cual almacenaremos el número que compararemos con la sentencia **IF**.

```
#include<stdlib.h>
#include<iostream>
using namespace std;
int main()
{
    int edad;
    cout<<"\nIntroduce tu edad:
    cin>>edad;
    if(edad>=18)
     cout<<"Puedes votar"<<endl;
    else
     cout<<"No puedes votar"<<endl;
    system("pause");
    return 0;
}
```

2. Se realiza la condición **IF>=18** ya que esa es la edad permitida para votar.

3. Si la condición se cumple, se imprimirá el mensaje de que puede votar.

4. Si la condición no cumple con el valor introducido se imprimirá el mensaje de que no puede votar por medio del comando **ELSE.**

Salida ejemplo**:**

IF = SI…

```
Introduce tu edad: 20
Puedes votar
Presione una tecla para continuar . . .
```

ELSE = SINO

```
Introduce tu edad: 15
No puedes votar
Presione una tecla para continuar . . .
```

Tema: Uso de estructuras de control de selección.

Ejemplo 2: Realizar un programa que pida dos números y muestre cuál de los 2 es el mayor mediante uso de una estructura de control.

1. Hacemos la primera condición comparando los dos números.

```
#include<stdlib.h>
#include<iostream>
using namespace std;
int main()
{
    int num, num2;
    cout<<"\nIntroduce un numero: ";
    cin>>num;
    cout<<"\nIntroduce un segundo numero: ";
    cin>>num2;
    if(num>num2)
     cout<<"\nEl primer numero: "<<num<<" es mayor"<<endl;
    else
     cout<<"\nEl segundo numero: "<<num2<<" es mayor"<<endl;
    system("pause");
    return 0;
```

2. Si el primer número es mayor **(>)** entonces se muestra el número más grande.

3. Si el primer número no cumple la condición entonces el segundo será mayor.

Salida ejemplo**:**

```
Introduce un numero: 10

Introduce un segundo numero: 2

El primer numero: 10 es mayor
Presione una tecla para continuar . . .
```

```
Introduce un numero: 5

Introduce un segundo numero: 6

El segundo numero: 6 es mayor
Presione una tecla para continuar . . .
```

Tema: Uso de estructuras de control de selección.

Ejemplo 3: Realizar un programa que contenga un menú de saludos con una estructura de control **IF**.

```
#include<iostream>
using namespace std;
int main()
{
    int opcion;
    cout<<"Menu de saldos."<<endl;
    cout<<"1. Decir hola."<<endl;
    cout<<"2. Decir adios."<<endl;
    cout<<"3. Decir suerte."<<endl;
    cout<<"4. Salir."<<endl;
    cout<<"\nQue opcion elige?: ";
    cin>>opcion;
    if(opcion == 1)
        cout<<"Hola! Bienvenido a mi programa"<<endl;
    if(opcion == 2)
        cout<<"Adios! Vuelve pronto"<<endl;
    if(opcion == 3)
        cout<<"Suerte en tu vida!"<<endl;
    if(opcion == 4)
        cout<<"Has salido del programa"<<endl;

    return 0;
}
```

1. Declaramos una variable tipo entero que nos ayudará a guardar la selección del usuario.

2. Antes de pedir qué opción desea elegir, escribiremos la lista de opciones de menú.

3. Preguntamos por la opción deseada y la guardaremos en la variable declarada.

4. Ahora aquí con una simple condición realizaremos uno por uno de todo el menú.

Salida:

```
Menu de saldos.
1. Decir hola.
2. Decir adios.
3. Decir suerte.
4. Salir.

Que opcion elige?: 1
Hola! Bienvenido a mi programa
```

Tema: Uso de estructuras de control de selección.

Ahora utilizaremos la estructura de control **Switch** la cual consiste en una serie de etiquetas **case** y con un **default** que nos ayudará a identificar si una opción no es encontrada. Lo mostraremos en los siguientes ejemplos.

Ejemplo 4: Realizar un programa que contenga un menú de vocales donde el usuario elija la opción deseada por medio de una estructura de control **Switch**.

La **sintaxis** del comando **switch** es la siguiente:

```
switch(variable)
{
case 1:
  Contenido;
break;
default:
}
```

```
#include<iostream>
using namespace std;
int main()
{
    int opcion;
    cout<<"Menu de vocales."<<endl;
    cout<<"1. A"<<endl;
    cout<<"2. E"<<endl;
    cout<<"3. I"<<endl;
    cout<<"4. O"<<endl;
    cout<<"5. U"<<endl;
    cout<<"6. Salir"<<endl;
    cout<<"\nQue opcion elige?: ";
    cin>>opcion;
    switch(opcion){
        case 1:
        cout<<"Vocal: A"<<endl;
        break;
        case 2:
        cout<<"Vocal: E"<<endl;
        break;
        case 3:
        cout<<"Vocal: I"<<endl;
        break;
        case 4:
        cout<<"Vocal: O"<<endl;
        break;
        case 5:
        cout<<"Vocal: U"<<endl;
        break;
        case 6:
        cout<<"Hasta luego!"<<endl;
        break;
    default:
        cout<<"Opcion no encontrada!"<<endl;
    }
    return 0;
}
```

1. Es algo parecido que con la condición **if** sólo que ahora usaremos el comando **switch** para la selección, dentro del **switch** tiene que ir la variable que almacenó la respuesta del usuario.

2. Ahora haremos uso del comando **case:** el cual nos servirá para almacenar el contenido de la opción elegida del menú.

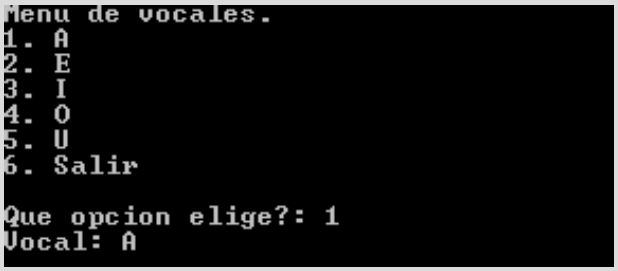
Como se muestra en este ejemplo los case tienen que ir ordenados.

3. Al finalizar un **case** debemos de colocar el comando **break** que nos permitirá finalizar con el **case** y seguir con los demás.

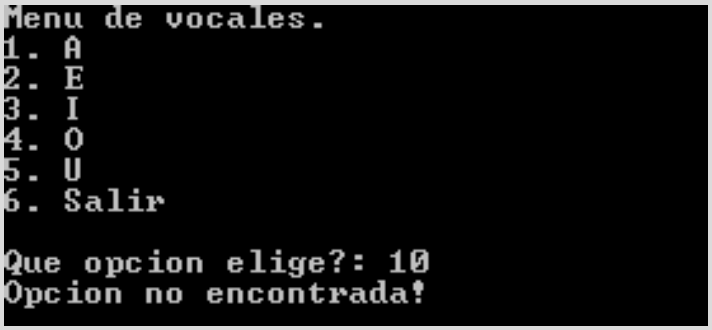
4. Por último utilizaremos el comando **default** para que cuando el usuario introduzca un número, por ejemplo **10** que no aparece en el menú se salga del programa y le mande un error.
Es importante que el **default** se coloque hasta el final de la sentencia **switch**, también note que el comando **default** no necesita utilizar el **break**.

Tema: Uso de estructuras de control de selección.

Salida ejemplo**:**



Probando el comando **default:**



El comando **default** nos ayuda para evitar ese tipo de errores cometidos por los usuarios.

Tema: Uso de estructuras de control de selección.

Ejemplo 5: Ahora realizaremos un parecido al anterior, pero ahora en lugar de utilizar números para la selección utilizaremos **letras**. Realizar un programa con un menú de operaciones básicas con el uso de la estructura de control **switch**.

1. Ahora utilizaremos letras en lugar de números, en este caso para que el **case** sea en letra se tienen que colocar entre comillas simples **' '** también se puso un doble **case** que nos sirve para que el usuario pueda introducir ya sea **A** o **a**, esto nos ayuda porque si sólo ponemos **A** el usuario puede colocar **a** y entonces el programa nos daría un error.

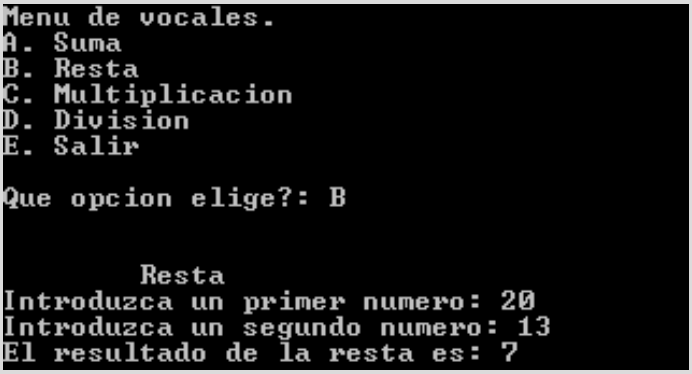
```
#include<iostream>
using namespace std;
int main()
{
    char opcion;
    int a,b,r;
    cout<<"Menu de vocales."<<endl;
    cout<<"A. Suma"<<endl;
    cout<<"B. Resta"<<endl;
    cout<<"C. Multiplicacion"<<endl;
    cout<<"D. Division"<<endl;
    cout<<"E. Salir"<<endl;
    cout<<"\nQue opcion elige?: ";
    cin>>opcion;
    switch(opcion){
        case 'A':
        case 'a':
            cout<<"\n\n\tSuma"<<endl;
            cout<<"Introduzca un primer numero: ";
            cin>>a;
            cout<<"Introduzca un segundo numero: ";
            cin>>b;
            r = a+b;
            cout<<"\n\nEl resultado de la suma es: "<<r;
        break;
        case 'B':
        case 'b':
            cout<<"\n\n\tResta"<<endl;
            cout<<"Introduzca un primer numero: ";
            cin>>a;
            cout<<"Introduzca un segundo numero: ";
            cin>>b;
            r = a-b;
        cout<<"El resultado de la resta es: "<<r;
        break;
```

Ojo que aunque pongamos 2 **cases** no necesitamos de 2 **break**.

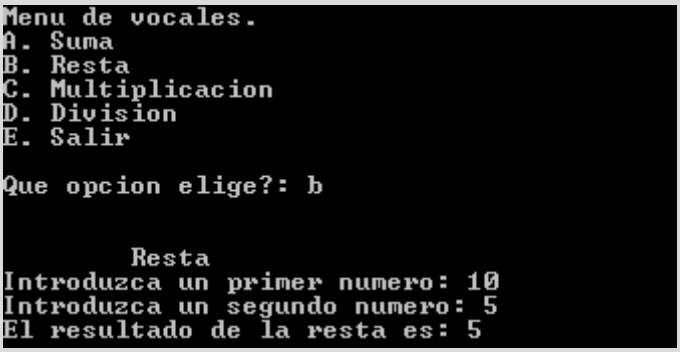
```
        case 'C':
        case 'c':
            cout<<"\n\n\tMultiplicacion"<<endl;
            cout<<"Introduzca un primer numero: ";
            cin>>a;
            cout<<"Introduzca un segundo numero: ";
            cin>>b;
            r = a*b;
            cout<<"El resultado de la multiplicacion es: "<<r;
        break;
        case 'D':
        case 'd':
            cout<<"\n\n\tDivision"<<endl;
            cout<<"Introduzca un primer numero: ";
            cin>>a;
            cout<<"Introduzca un segundo numero: ";
            cin>>b;
            r = a/b;
            cout<<"El resultado de la divion es: "<<r;
        break;
        case 'E':
        case 'e':
            cout<<"Hasta luego!"<<endl;
        break;

    default:
        cout<<"Opcion no encontrada!"<<endl;
    }
    return 0;
}
```

Tema: Uso de estructuras de control de selección.

Salida ejemplo: Aquí usamos colocamos la opción **B**.



En esta opción utilizamos también la letra **b**.



Tema: Uso de estructuras de control de selección.

Prácticas.

Práctica 1: Realizar un programa donde se introduzca un número entero y mediante la estructura de selección **if** determinar si es positivo o negativo.

Salida ejemplo:

```
Introduzca un numero: -5
Es negativo
```

Práctica 2: Realizar un programa donde se pidan 3 números enteros y los imprima de menor a mayor utilizando una estructura de selección **if**.

Salida ejemplo:

```
Introduzca el primer numero: 10
Introduzca el segundo numero: 5
Introduzca el tercer numero: 6
El orden queda asi: 5, 6, 10
```

Práctica 3: Realizar un programa que pida 3 números enteros e imprima si el tercer número es igual a la suma de los dos primeros.

Salida ejemplo:

```
Introduzca el primer numero: 10
Introduzca el segundo numero: 10
Introduzca el tercer numero: 20
El tercer numero es la suma del primero y segundo
```

Tema: Uso de estructuras de control de selección.

Prácticas.

Práctica 4: Realizar un programa que pida el lado y la base del triángulo y que en un menú estén las opciones de triángulo equilátero, isósceles y escaleno, y calcule el perímetro dependiendo de la opción elegida.

Salida ejemplo:

```
Lado del triangulo: 15
Base del triangulo: 40
        Menu
1. Equilatero
2. Isasceles
3. Escaleno
Elija la opcion: 2
El perimetro es: 70
```

Tema: Estructuras de repetición (for, while y do-while).

Empezaremos por la estructura de repetición **for** y mostraremos los siguientes ejemplos:

Ejemplo 1: Realizaremos un programa que imprima una numeración del 1 al 10 usando la estructura de repetición **for**.

La **sintaxis** del comando **for** es la siguiente:
for(variable = valor inicial; variable <= valor final; variable incremento **++**)
{
bloque de instrucciones
}

1. Declaramos una variable de tipo entero para el uso del ciclo.

```
#include <iostream>

using namespace std;

main(){
int x;
     for(x = 1; x <= 10; x++)
     {
          cout<<x;
     }

return 0;

}
```

4. Por último le daremos un incremento, en este caso nos especificamos así que lo hará de **1** en **1**.

3. Asignamos hasta qué número queremos terminar.

Salida:

```
12345678910
```

Tema: Estructuras de repetición (for, while y do-while).

Ahora haremos un ejemplo pero con la estructura de repetición **while**.

Ejemplo 2: Realizaremos un programa que imprima una numeración repetición **while**.

La **sintaxis** de **while** es la siguiente:
variable = inicio
while (variable <= final)
{
bloque de instrucciones;
contador++;
}

```
#include <iostream>

using namespace std;

main(){
    int a;

    a = 1;
    while (a<=20)
    {
        cout<<a<<",";
        a++;
    }

return 0;
}
```

1. Empezaremos con el valor inicial y va arriba de la estructura **while** y se asigna desde donde quiere empezar.

2. Aquí asignamos hasta donde queremos que llegue nuestro número.

3. Imprimimos la variable del ciclo.

4. El contador lo colocamos después de lo que imprimamos.

Salida:

```
12345678910
```

Tema: Estructuras de repetición (for, while y do-while).

Ahora haremos un ejemplo pero con la estructura de repetición **while**.

Ejemplo 3: Realizaremos un programa que imprima una numeración del 1 al 100 usando la estructura de repetición **do-while**.

La **sintaxis** del **do-while** es la siguiente:
variable = inicio;
do
{
bloque de instrucciones;
contador++;
}
while(variable <= final);

```
#include <iostream>

using namespace std;

main(){
    int z;

    z = 5;
    do{

        cout<<z<<",";
        z++;

    }
    while(z <= 20);

return 0;

}
```

1. Al igual que **while** se comienza antes del comando.

2. Se empieza por un **do** antes del **while**.

3. Adentro se colocan las instrucciones y el contador.

4. Al final utilizamos **while** y asignamos hasta donde se quiere finalizar el ciclo. Note que en este **while** se hace uso del punto y coma.

Salida:

```
12345678910
```

Tema: Estructuras de repetición (for, while y do-while).

Prácticas.

Práctica 1: Realizar un programa en donde el usuario pida el número de inicio y el número final utilizando la estructura de repetición **for**.

Salida ejemplo:

```
Introduce el numero de inicio del ciclo: 5
Introduce el numero final del ciclo: 15

5 6 7 8 9 10 11 12 13 14 15
```

Práctica 2: Realizar un programa le pida al usuario qué tabal de multiplicar quiere y mostrarla con el uso de la estructura de repeticición **for**.

Salida ejemplo:

```
        Tabla de multiplicar

Que tabla de multiplicar deseas?: 5

5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

}

Práctica 3: Realizar un programa que pida número de inicio, final y número de saltos utilizando la estructura de repetición **while**.

Salida ejemplo:

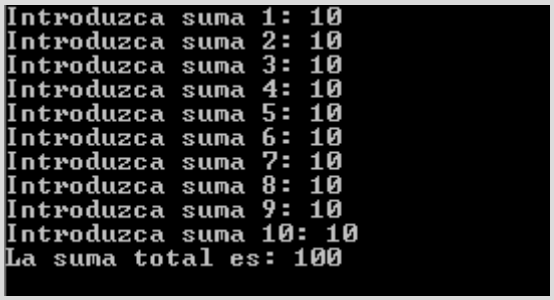
```
Numero de inicio: 2
Numero final: 20
Numero de saltos: 2
2 4 6 8 10 12 14 16 18 20
```

Tema: Estructuras de repetición (for, while y do-while).

Prácticas.

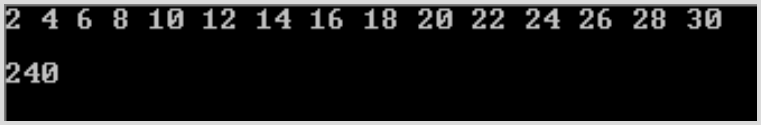
Práctica 4: Realizar un programa que sume 10 números introducidos por el usuario utilizando la estructura de repetición **while**.

Salida ejemplo:



Práctica 5: Realiza un programa que realice una numeración del 1 al 30 que identifique los números **pares** y que los sume utilizando la estructura de repetición **do-while**.

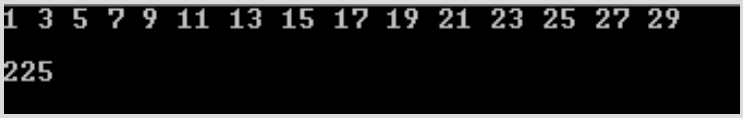
Salida ejemplo:



}

Práctica 6: Realizar el mismo programa anterior pero ahora con los números **impares** y que los sume utilizando la estructura de repetición **do-while**.

Salida ejemplo:



Criterio de desempeño de la unidad III:	**Actividades de comprobación:**	**Actividades de evaluación :**
Programa estructurado que contenga (if while, do while)	Desarrolle programas con el uso de estructuras de control, procedimientos y funciones	Ejecución correcta del programa que utilice las estructura de selección y repeticiones.

Unidad IV.- Uso de arreglos y apuntadores

En la unidad se analiza la declaración de arreglos unidimensionales y bidimensionales así como el uso de apuntadores con la finalidad de formular programas para satisfacer las necesidades de las pequeñas y medianas empresas.

		utilcen metodologías actuales

UNIDAD DE COMPETENCIA IV	**ELEMENTOS DE COMPETENCIA**		
	Conocimientos	**Habilidades**	**Actitudes/ Valores**
Analizar los conceptos de arreglos y apuntadores para formular programas para satisfacer las necesidades de las pequeñas y medianas empresa.	-Declaración y uso de arreglos unidimensionales y bidimensionales -Uso de apuntadores, direcciones y aritmética de direcciones	Análisis y Comprensión Comparación Capacidad de análisis Solución de problemas	Investigación y prueba sistemática

ESTRATEGIAS DIDÁCTICAS:	**RECURSOS REQUERIDOS**	**TIEMPO DESTINADO**
-Planteamiento de problemas -Solución de ejercicios -Codificación y ejecución de programas	Sesión bibliográfica (Aitken y Jones, Aprendindo C en 21 días, Prentice May, México, 1994) Pizarrón Proyector de acetatos Equipo de cómputo	22 horas

CRITERIOS DE DESEMPEÑO IV	**EVIDENCIAS**	
	DESEMPEÑO	**PRODUCTOS**
Elaborar 5 programas para procesar cadenas utilizando arreglos, elaborar cinco programas que calculen medidas de tendencia central, mínimos y máximos utilizando arreglos enteros	Desarrollo de programas utilizando arreglos unidimensinales para menejo de enteros y cadenas	Programa y ejecución utilizando arreglos unidimensionales Programa y ejecución utilizando arreglos bidimensionales

Tema: Declaración y uso de arreglos unidimensionales y bidimensionales.

Los arreglos **unidimensionales** son estructuras que usan posiciones de memoria que están contiguas y se indexan de forma numérica.

Ejemplo 1: Arreglo unidimensional en el cual se introducirán números para llenarlo.
Procedimiento:

1. Se declara una variable de tipo entero con 4 espacios en memoria.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

using namespace std;

int main(){

int numero[4];
for(int ciclo = 0; ciclo <=4; ciclo++){

    cout<<"Introduce "<<ciclo<<" numero: ";
    cin>>numero[ciclo];
}
system("pause");
return 0;
}
```

2. Se realiza un ciclo en el cual el usuario introducirá números y el arreglo se irá llenando.

Salida:

```
Introduce 0 numero: 20
Introduce 1 numero: 10
Introduce 2 numero: 2
Introduce 3 numero: 5
Introduce 4 numero: 6
Presione una tecla para continuar . . .
```

Tema: Declaración y uso de arreglos unidimensionales y bidimensionales.

Ejemplo 2: Arreglo unidimensional con datos ya asignados y que se impriman.
Procedimiento:

Podemos crear arreglos con datos ya reservados los cuales podemos asignar y mostrarlos.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

using namespace std;

int main(){

int numeros[10]={3,4,5,10,45,10,11,56,89,75};
for(int ciclo = 0; ciclo <=9; ciclo++){
    cout<<ciclo[numeros]<<" ";
}
cout<<endl;
system("pause");
return 0;
}
```

1. Declaramos un arreglo con 10 espacios los cuales ya se han asignado.

Dentro de los corchetes se ponen los valores que queremos asignar al arreglo.

2. Se crea un ciclo para poder imprimir todos los datos que tiene el arreglo.

Salida:

```
3 4 5 10 45 10 11 56 89 75
Presione una tecla para continuar . . .
```

Tema: Declaración y uso de arreglos unidimensionales y bidimensionales.

Ejemplo 3: Arreglo bidimensional que imprima del 1 al 100.
Procedimiento:

Los arreglos **bidimensionales** o **matrices** son arreglos de dos dimensiones que ocupan datos grandes del mismo tipo, se pueden utilizar para representar datos que pueden verse como una tabla con filas y columnas.

1. Se declara el arreglo bidimensional que en este caso tendrá un espacio de 100 números.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x,y, numeros[10][10];
    for (x=0;x<10;x++)
    {
        for (y=0;y<10;y++)
        {
            numeros[x][y]=(x*10)+1+y;
        }

    }
    for (x=0;x<10;x++)
    {
        for (y=0;y<10;y++)
        {
            printf("%d ",numeros[x][y]);
        }
        printf("\n");
    }

    system("PAUSE");
    return 0;
}
```

2. Se crea un ciclo anidado para leer los números empezando desde el 1 hasta el 100.

Salida:

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
Presione una tecla para continuar . . .
```

Tema: Declaración y uso de arreglos unidimensionales y bidimensionales.

Ejemplo 4: Arreglo bidimensional que sumen los valores de los arreglos.
Procedimiento:

En este caso sumaremos las dos tablas que tendrán un espacio de 2 números enteros los cuales se sumarán para dar un resultado de 3, se puede cambiar el número dentro del arreglo para que el resultado sea diferente.

```
#include<stdio.h>
#include<conio.h>

void sumar(int a[2][2],int b[2][2]){
    int suma[2][2],i,j;

    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            suma[i][j] = a[i][j]+b[i][j];
        }
    }

    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            printf("%i ",suma[i][j]);
        }
        printf("\n");
    }
}
int main(){
    int tabla1[2][2] = {{1,2},{2,1}};
    int tabla2[2][2] = {{2,1},{1,2}};
    int i,j;

    sumar(tabla1,tabla2);

    getch();
    return 0;
}
```

1. Creamos un ciclo anidado para llenar los espacios del arreglo y sumarlos.

2. Ahora se imprimen las sumas ya realizadas del arreglo.

3. Ahí se asignan los valores que queremos tener en el arreglo.

Salida:

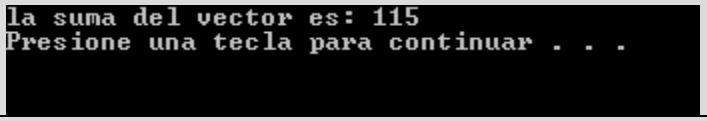
```
3 3
3 3
```

Tema: Declaración y uso de arreglos unidimensionales y bidimensionales.

Prácticas.

Práctica 1: Realizar un programa con un arreglo **unidimensional** que sume los números dentro de este, los cuales serán los siguientes: 10, 21, 2, 12, 31, 5, 7, 9, 11, 7

Salida ejemplo:

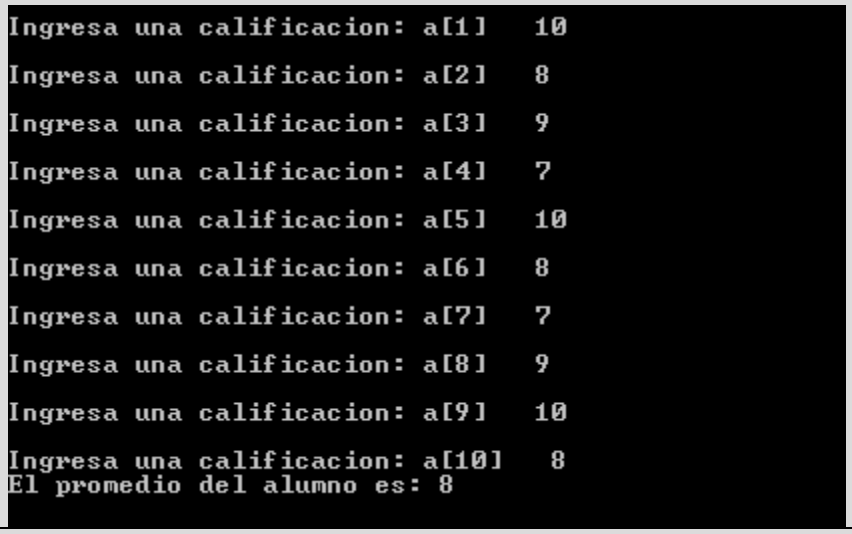


Práctica 2: Realizar un programa con un arreglo **unidimensional** que contenga las vocales y que sólo imprima la letra A y U.

Salida ejemplo:

Práctica 3: Realizar un programa con un arreglo **unidimensional** que pida 10 calificaciones y las sume e imprima el resultado.

Salida ejemplo:

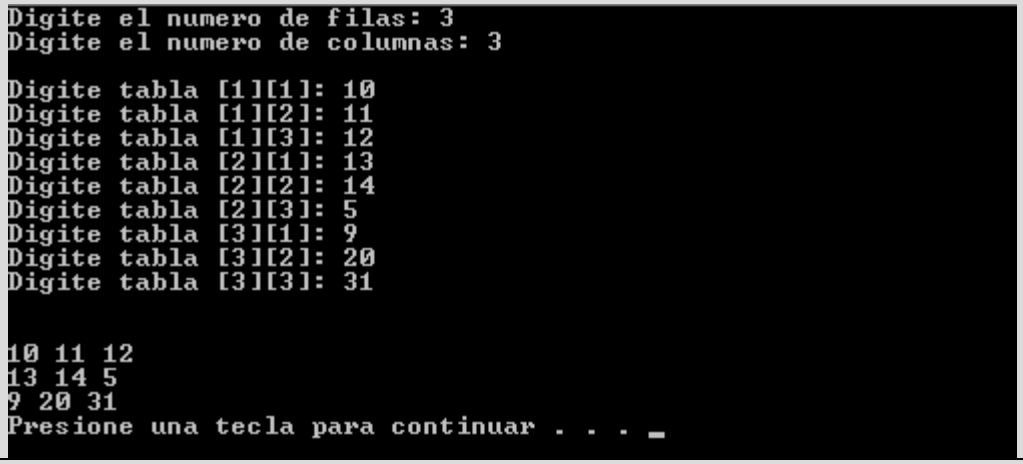


Tema: Declaración y uso de arreglos unidimensionales y bidimensionales.

Prácticas.

Práctica 4: Realizar un programa con un arreglo **bidimensional** en el cual se introduzca el número de filas y columnas, y que el usuario las vaya llenando, al final imprimirá el resultado de los datos introducidos.

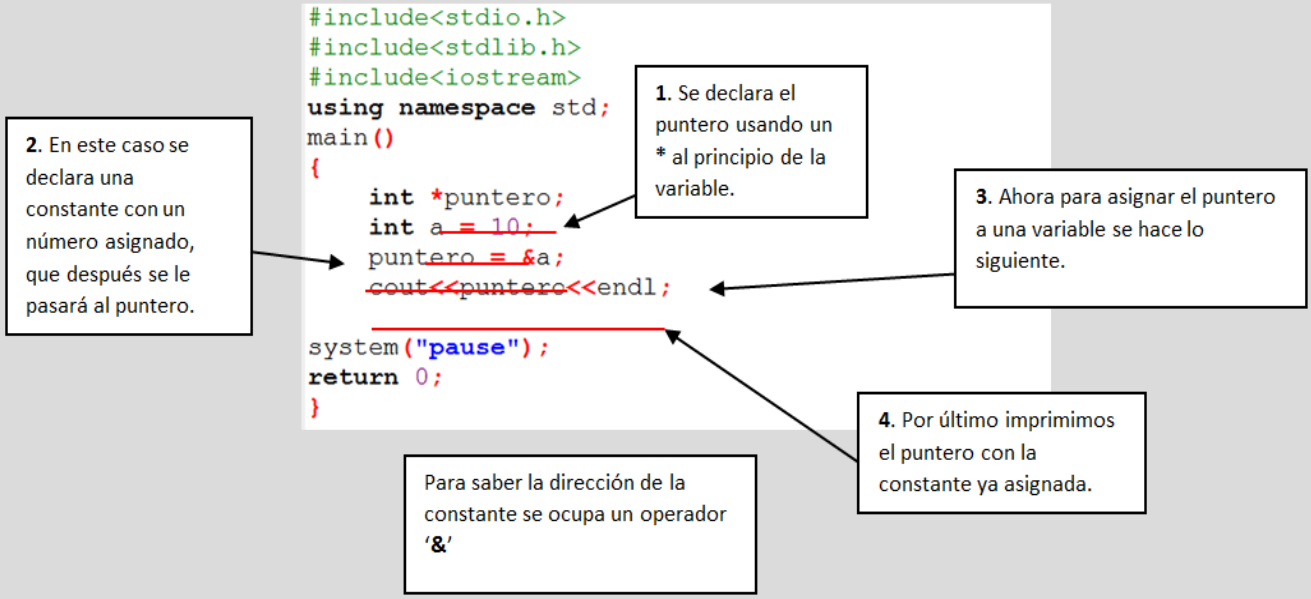
Salida ejemplo:



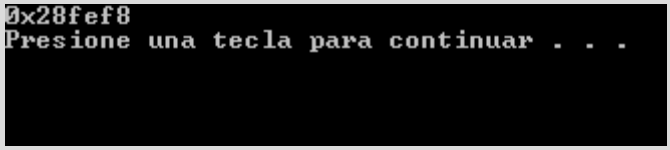
Tema: Uso de apuntadores, direcciones y aritmética de direcciones.

Un puntero o un apuntador es una variable que contiene la dirección de memoria de un dato o de otra variable que contiene al dato en un arreglo. En este ejemplo veremos cómo obtener la **dirección** de un puntero o apuntador.

Ejemplo 1: Crear un puntero de tipo entero que muestre su dirección de la memoria en pantalla.

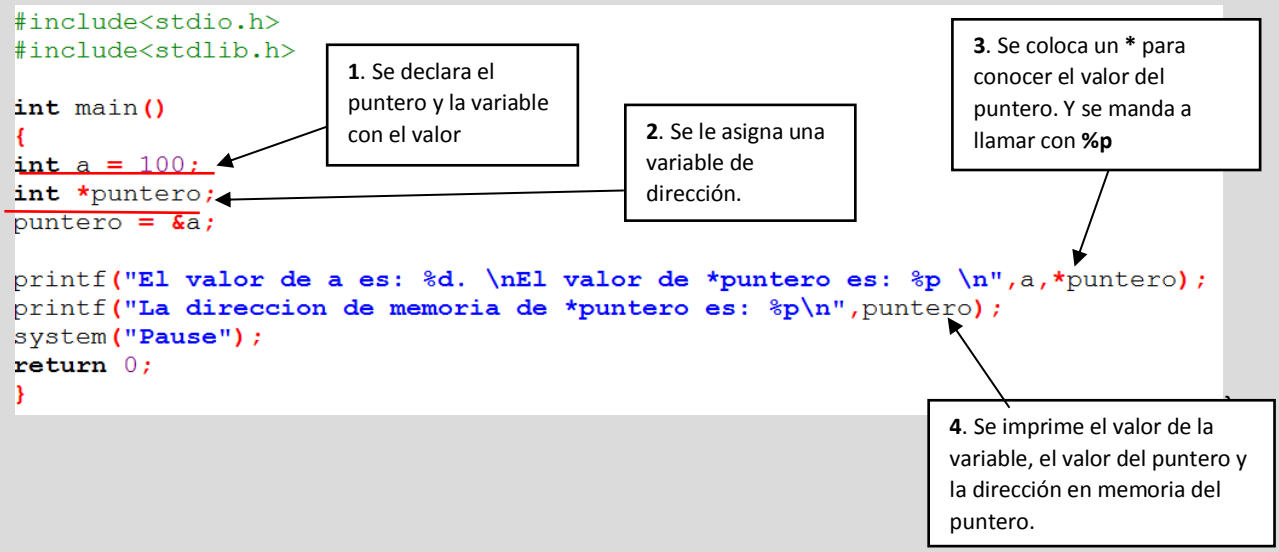


Salida:

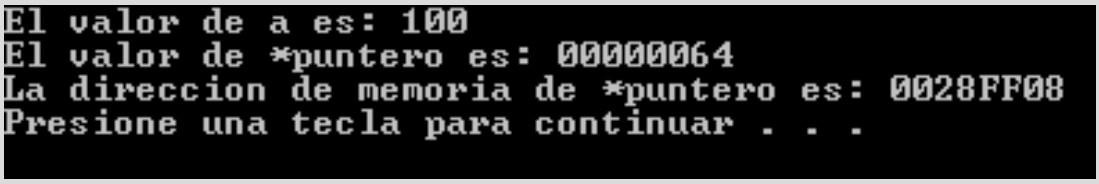


Tema: Uso de apuntadores, direcciones y aritmética de direcciones.

Ejemplo 2: Crear un programa que muestre el valor de la variable, el valor del puntero y la dirección en memoria que tiene el puntero.



Salida:



Tema: Uso de apuntadores, direcciones y aritmética de direcciones.

Ejemplo 3: Crear un programa que haga una suma entre dos números enteros, que muestre el resultado, la dirección del puntero y la dirección en memoria del resultado.

1. Se realizan las declaraciones siguientes:

2. El resultado que se guardará en la variable **re** se quedará en el puntero.

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
int num1,num2,re;
int *puntero;
puntero = &re;
printf("Introduce un numero: ");
scanf("%d",&num1);
printf("Introduce un segundo numero: ");
scanf("%d",&num2);
re = num1+num2;
printf("\nEl resultado de la suma es: %d\n",re);
printf("El valor del puntero es: %p\n",*puntero);
printf("La direccion en memoria de la suma es: %p\n",puntero);
system("Pause");
return 0;
}
```

}

Salida:

```
Introduce un numero: 10
Introduce un segundo numero: 20

El resultado de la suma es: 30
El valor del puntero es: 0000001E
La direccion en memoria de la suma es: 0028FF00
Presione una tecla para continuar . . .
```

Tema: Uso de apuntadores, direcciones y aritmética de direcciones.

Ejemplo 4: Crear un programa que haga una suma entre dos números enteros, que muestre el resultado, la dirección del puntero y la dirección en memoria del resultado.

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
int num1,num2,re;
int *puntero;
puntero = &re;
printf("Introduce un numero: ");
scanf("%d",&num1);
printf("Introduce un segundo numero: ");
scanf("%d",&num2);
re = num1+num2;
printf("\nEl resultado de la suma es: %d\n",re);
printf("El valor del puntero es: %p\n",*puntero);
printf("La direccion en memoria de la suma es: %p\n",puntero);
system("Pause");
return 0;
}
```

1. Se realizan las declaraciones siguientes:

2. El resultado que se guardará en la variable **re** se quedará en el puntero.

}

Salida:

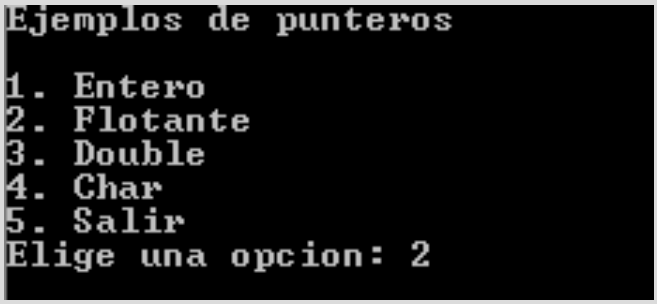
```
Introduce un numero: 10
Introduce un segundo numero: 20

El resultado de la suma es: 30
El valor del puntero es: 0000001E
La direccion en memoria de la suma es: 0028FF00
Presione una tecla para continuar . . .
```

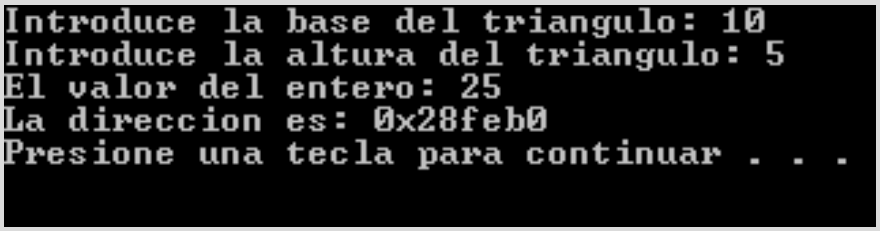
Tema: Uso de apuntadores, direcciones y aritmética de direcciones.

Prácticas.

Práctica 1: Realizar un programa que contenga un menú con los tipos de datos: **Entero, Flotante, Double y Char**, y que en cada uno de ellos tenga un ejemplo de ese tipo de dato y que con un puntero muestre el valor y la dirección del puntero.

Salida ejemplo:





Criterio de desempeño de la unidad IV:	**Actividades de comprobación:**	**Actividades de evaluación :**
Elaborar programas para realizar arreglos unidimensionales y bidimensionales.	Desarrollo de programas utilizando arreglos unidimensionales para manejo de enteros y cadenas	Programa y ejecución utilizando arreglos unidimensionales. Programa y ejecución utilizando arreglos bidimensionales.

Unidad V.- Aplicar el uso de archivos y memoria dinámica para el manejo de información

En la unidad III en esta unidad se trabajara con el uso de un archivo con los comandos de abrir, cerrar, actualizar en distintos modos de lectura y escritura, así como los archivos temporales reservado y liberación de memoria.

<table>
<tr><td rowspan="2">UNIDAD DE COMPETENCIA V</td><td colspan="3">ELEMENTOS DE COMPETENCIA.</td></tr>
<tr><td>Conocimientos</td><td>Habilidades</td><td>Actitudes/Valores</td></tr>
<tr><td>Aplicar el uso de archivos y memoria dinámica para el manejo de la información.</td><td>*Control de archivo (abrir, cerrar, actualizar en los distintos modos de lectura y/o escritura)
*Uso de archivos temporales
*Reservado y liberación de espacio en memoria
*Cambio de tamaño de los bloques en</td><td>*Análisis
*Resolución de problemas
*Optimización de recursos</td><td>*Constancia
*Interés
*Flexibilidad en la resolución de problemas</td></tr>
<tr><td>ESTRATEGIAS DIDÁCTICAS:

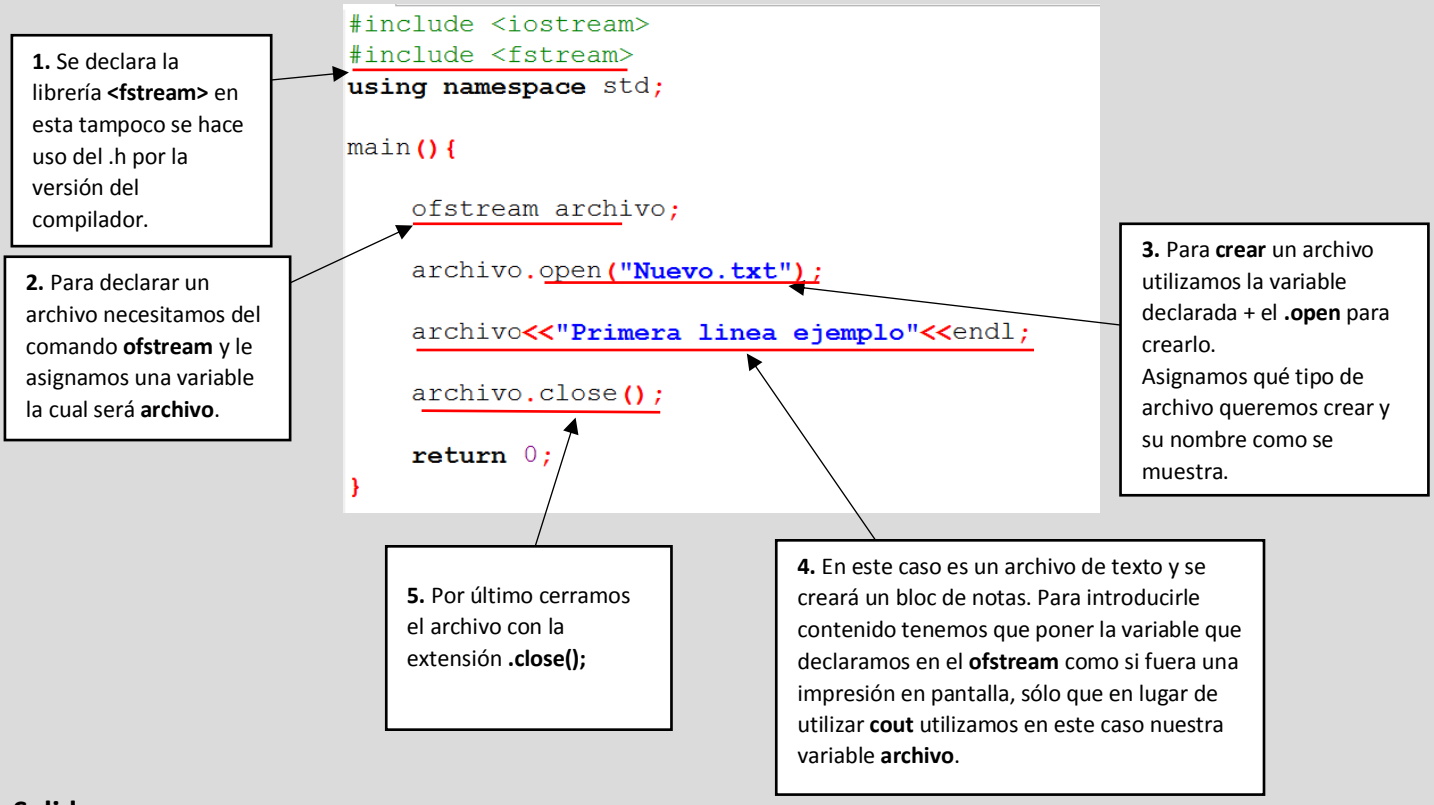
*Demostración con la práctica
*Mapas mentales
*Solución de problemas</td><td colspan="2">RECURSOS REQUERIDOS:

* Computadora
* Pizarrón</td><td>TIEMPO DESTINADO:

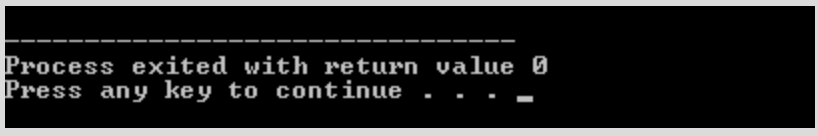
22 horas</td></tr>
<tr><td rowspan="2">CRITERIOS DE DESEMPEÑO III</td><td colspan="3">EVIDENCIAS</td></tr>
<tr><td>DESEMPEÑO</td><td colspan="2">PRODUCTOS</td></tr>
<tr><td>Uso de archivos y memoria dinámica</td><td>Desarrollo de aplicaciones utilizando archivos y memoria dinámica</td><td colspan="2">Realizar un programa que administre archivos: altas, bajas, cambios y consultas de registros</td></tr>
</table>

Tema: Control de archivo (abrir, cerrar, actualizar en distintos modos de lectura y/o escritura).

Para la creación de archivos se necesita de la librería **fstream** que nos ayudará a crear archivos, abrirlos, cerrarlos y actualizarlos, como veremos en el siguiente ejemplo:



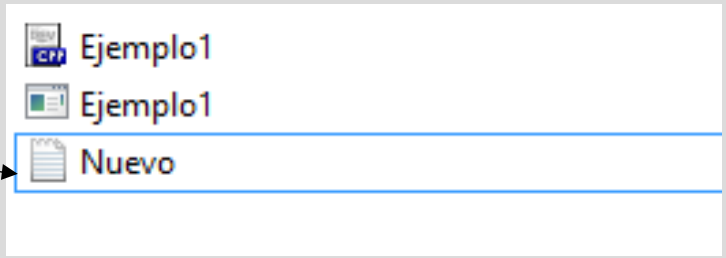
Salida:



En la salida no nos arrojará nada pero se creará el archivo en la misma ubicación donde tengas guardado tu programa.

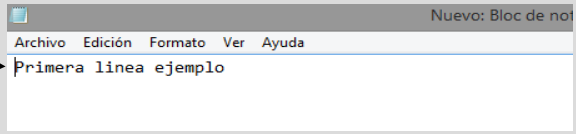
Tema: Control de archivo (abrir, cerrar, actualizar en distintos modos de lectura y/o escritura).

Se nos creará el archivo **.txt** con el nombre que le asignamos.



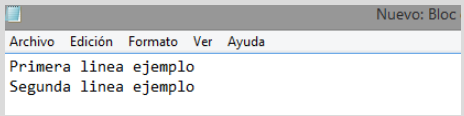

```
archivo<<"Primera linea ejemplo"<<endl;
```

Si abrimos el archivo lo que contenga será lo que imprimimos.



Si creamos otra línea nos aparecerá lo siguiente:

```
archivo<<"Primera linea ejemplo"<<endl;
archivo<<"Segunda linea ejemplo"<<endl;
```



Tema: Control de archivo (abrir, cerrar, actualizar en distintos modos de lectura y/o escritura).

Ejemplo 2: Realizar un programa que guarde 3 nombres introducidos por el usuario en un archivo **.txt**

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

main(){
    char nom1[10], nom2[10], nom3[10];
    ofstream nombres;

    nombres.open("Nombre.txt");
    cout<<"Introduce el primer nombre: ";
    cin>>nom1;
    cout<<"Introduce el segundo nombre: ";
    cin>>nom2;
    cout<<"Introduce el tercer nombre: ";
    cin>>nom3;
    nombres<<nom1<<endl;
    nombres<<nom2<<endl;
    nombres<<nom3<<endl;

    nombres.close();

    return 0;
}
```

1. Declaramos 3 variables de tipo carácter en este caso.

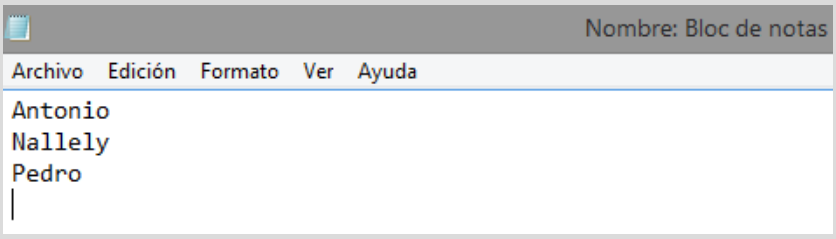
2. Creamos la variable que creará el archivo.

3. Declaramos el tipo de archivo y su nombre.

4. Una vez introducidos los nombres por el usuario los mostraremos en el archivo.

Salida:

```
Introduce el primer nombre: Antonio
Introduce el segundo nombre: Nallely
Introduce el tercer nombre: Pedro
```



Tema: Control de archivo (abrir, cerrar, actualizar en distintos modos de lectura y/o escritura).

Ejemplo 3: Realizar un programa que pida nombre, apellido y edad de una persona y que lo guarde en un archivo de texto.

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

int main(){
    ofstream base;
    int s = 0;
    int op;
    int ed;
    char nom[10], ap[10];
    while(!s){
    cout<<"1. Crear archivo"<<endl;
    cout<<"2. Insertar datos"<<endl;
    cout<<"3. Ver datos"<<endl;
    cout<<"4. Salir"<<endl;
    cout<<": ";
    cin>>op;
    switch(op){
    case 1:
            if(base){
                cout<<"\n\n\n";
                base.open("Base de datos.txt");}
            if(!base) {
                base.open("Base de datos.txt");
                cout<<"Ya se ha creado un archivo!"<<endl;
        }
        base.close();
    break;
```

1. Creamos un menú con las opciones siguientes.

2. En la primera opción crearemos el archivo que en este caso es **.txt**

```
    case 2:
        base.open("Base de datos.txt");
            if(base == NULL){
                cout<<"No se ha creado un fichero"<<endl;
            }
            else{

            cout<<"\tIntroduce su nombre: ";
            cin>>nom;
            cout<<"\tIntroduce su apellido: ";
            cin>>ap;
            cout<<"\tIntroduce la edad: ";
            cin>>ed;
            }
            base.close();

    break;
    case 3:
        base.open("Base de datos.txt");
        cout<<"\n\n";

            cout<<"\tNombre: "<<nom<<endl;
            cout<<"\tApellido: "<<ap<<endl;
            cout<<"\tEdad: "<<ed<<endl;
            base<<nom<<endl;
            base<<ap<<endl;
            base<<ed<<endl;
            base.close();
    break;
```

3. En la segunda opción si ya se ha creado el archivo se podrán introducir los datos.

4. En la tercera opción mostraremos los datos introducidos y los guardaremos en el archivo.

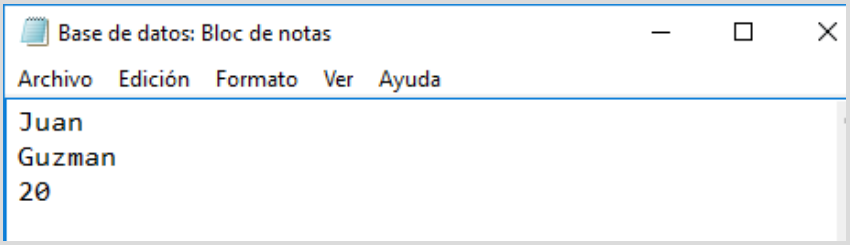
Tema: Control de archivo (abrir, cerrar, actualizar en distintos modos de lectura y/o escritura).

```
        case 4:
                cout<<"ADIOS"<<endl;
                s = 1;
        break;
        default:
                cout<<"Esta opcion no existe"<<endl;

    }
}
    base.close();

    return 0;
}
```

4. En la cuarta opción acabaremos con el ciclo.

Salida:

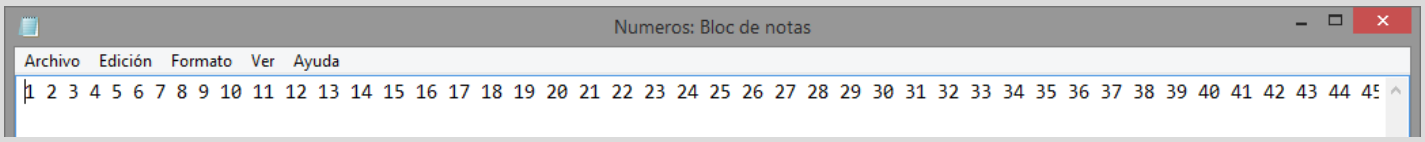
```
1. Crear archivo
2. Insertar datos
3. Ver datos
4. Salir
: 1



1. Crear archivo
2. Insertar datos
3. Ver datos
4. Salir
: 2
        Introduce su nombre: Juan
        Introduce su apellido: Guzman
        Introduce la edad: 20
1. Crear archivo
2. Insertar datos
3. Ver datos
4. Salir
: 3


        Nombre: Juan
        Apellido: Guzman
        Edad: 20
1. Crear archivo
2. Insertar datos
3. Ver datos
```



Tema: Control de archivo (abrir, cerrar, actualizar en distintos modos de lectura y/o escritura).

Práctica 1: Realizar un programa que realice una numeración del 1 al 100 utilizando estructuras de repetición y que lo guarde en un archivo **txt**.

Salida:



Práctica 2: Realizar un programa que registre los datos de 10 empleados por medio de un ciclo y los guarde en un archivo **txt**.

Tema: Uso de archivos temporales.

Ejemplo 1: Realizar un programa que abra un archivo temporal para modificarlo.

```c
#include <stdio.h>
 int main()
{
   FILE *ptrArchivo;
   FILE *ptrArchivoTemp;
   int c;
   char nombreArchivo[ 30 ];
   printf("Introduzca archivo para modificar: " );
   scanf( "%29s", nombreArchivo );
   if ( ( ptrArchivo = fopen( nombreArchivo, "r+" ) ) != NULL ) {
      if ( ( ptrArchivoTemp = tmpfile() ) != NULL ) {
         printf( "\nEl archivo original es: \n" );
         while ( ( c = getc( ptrArchivo ) ) != EOF ) {
            putchar( c );
            putc( c == 't' ? ' ': c, ptrArchivoTemp );
         }
         rewind( ptrArchivoTemp );
         rewind( ptrArchivo );
         printf( "\n\nEl archivo modificado: \n" );
         while ( ( c = getc( ptrArchivoTemp ) ) != EOF ) {
            putchar( c );
            putc( c, ptrArchivo );
         }
      }
      else {
         printf( "No se puede abrir el archivo temporal!\n" );
      }
    }
   else {
      printf( "No se puede abrir el archivo %s \n", nombreArchivo );
   }
  return 0;
}
```

1. Declaramos los archivos.

2. Pedimos que introduzca el nombre del archivo a modificar.

Salida:

```
Introduzca archivo para modificar: Archivo

El archivo original es:
Archivo modificado


El archivo modificado:
Archivo modificado
```

Tema: Reservación, liberación y reasignación de espacio en memoria.

En este tema veremos diferentes funciones las cuales nos permitirán realizar la reservación, liberación y reasignación de espacio en memoria, con las funciones **malloc, realloc y free**.

Ejemplo 1: Asignar un total de bytes en la memoria por medio del comando **malloc**.

```
#include<stdio.h>
#include<stdlib.h>
int main ()
{
int *p ;

int nbytes = 100;

p = (int*)malloc(nbytes*sizeof(int));

if (p == NULL)
    printf("Insuficiente memoria \n \n");
else
printf("Se han asignado %i bytes de memoria \n \n",nbytes);


system("pause");
return 0;
}
```

1. Se declara un puntero para asignar el espacio en memoria.

2. Asignamos un espacio total de bytes asignados.

3. Declaramos al puntero la cantidad de espacio en memoria que utilizaremos.

4. Si los bytes asignados exceden de memoria saldrá el mensaje que hay insuficiente memoria, sino se asignarán los bytes a la memoria

La **sintaxis** del comando **realloc** es:

(**tipo de variable***)**malloc**(variable***sizeof**(**tipo de variable**));

Salida:

```
Se han asignado 100 bytes de memoria
```

Tema: Reservación, liberación y reasignación de espacio en memoria.

Ejemplo 2: Realizar una reasignación de espacio en la memoria por medio del comando **realloc**.

```c
#include <stdlib.h>
#include <stdio.h>
int main(){
    int *arreglo,*arreglo_nuevo;
    int x; //Contador
    arreglo = (int*)malloc(3*sizeof(int));
    arreglo[0] = 1;
    arreglo[1] = 2;
    arreglo[2] = 3;
    printf("Espacio asignado: \n");
    for(x=0; x<3; x++){
        printf("%i ",arreglo[x]);
    }
    arreglo_nuevo = (int*)realloc(arreglo,5*sizeof(int));
    arreglo[3] = 4;
    arreglo[4] = 5;
    printf("\n\n");
    printf("Nuevo espacio reasignado: \n");
    for(x=0; x<5; x++){
        printf("%i ",arreglo_nuevo[x]);
    }
    printf("\n");

    system("pause");
    return 0;
}
```

1. Declaramos dos punteros, uno para la asignación en memoria y el otro para la reasignación.

2. Realizamos la asignación en memoria como ya vimos en el ejemplo anterior, sólo que en este caos le daremos **3** espacios asignados.

3. Haremos uso de arreglos para asignarle los valores que queramos dentro de ellos.

4. Ahora haremos uso del **realloc** y la sintaxis es parecida a la de **malloc**. Asignamos los nuevos espacios que en este caso serán **2** espacios nuevos por eso colocamos el número **5**.

5. Ahora declararemos los otros **2** arreglos y les asignaremos un espacio.

Salida:

```
Espacio asignado:
1 2 3

Nuevo espacio reasignado:
1 2 3 4 5
```

Tema: Reservación, liberación y reasignación de espacio en memoria.

Ejemplo 3: Realizar la liberación de memoria en un programa utilizando la función **free()**.

```c
#include<stdio.h>
#include<stdlib.h>
int main ()
{
int *ptr ;

ptr=(int*)malloc(sizeof(int));

if (ptr == NULL){
    printf("Insuficiente memoria \n \n"); }
else{
printf("Hay espacio en memoria!\n\n");
*ptr = 10000;
printf("Se han asignado %i bytes de memoria \n \n",*ptr);
}

free(ptr);
printf("Se ha liberado espacio: %p\n\n",ptr);
system("pause");
return 0;
}
```

1. Asignamos el espacio en memoria.

2. Si no hay espacio arrojará el mensaje.

3. Si hay espacio se asignarán **1000** bytes que declaramos.

4. Utilizamos el comando **free** para liberar el espacio en memoria que se le asignó al puntero.

5. Por último mostraremos el espacio en memoria que se liberó.

Salida:

```
Hay espacio en memoria!

Se han asignado 10000 bytes de memoria

Se ha liberado espacio: 00690D38
```

Tema: Reservación, liberación y reasignación de espacio en memoria.

Ejemplo 4: Realizar un programa que asigne espacio en memoria, después lo reasigne y lo libere.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main ()
{
    int*p=NULL, *q=NULL;
    int nbytes = 100;
    if
    ((p=(int *)malloc(nbytes)) ==NULL)
    {
        printf("Insuficiente espacio de memoria \n");
        return -1;
    }
    printf("Se han asignado %d bytes de memoria \n", nbytes);
    if (nbytes == 0){
    free(p);
    printf("\n El bloque ha sido liberado \n");
    return -1;
}
    q = (int*)realloc(p, nbytes * 2);
    if(q == NULL){
    printf("La reasignacion no ha sido posible \n");
    printf("Se conserva l bloque originar \n");
}
    else{
    p=q;
    printf("Bloque reasignado\n");
    printf("Nueva cantidad %d bytes \n", nbytes*2);
    }
    free(p);
    printf("\n\n El bloque ha sido liberado: %p \n\n",p);
    system("pause");
    return 0;
}
```

Salida:

```
Se han asignado 100 bytes de memoria
Bloque reasignado
Nueva cantidad 200 bytes


 El bloque ha sido liberado: 00650D58
```

Tema: Reservación, liberación y reasignación de espacio en memoria.

Prácticas.

Ejercicio 1: Realizar un programa en el cual se le asigne en memoria un nombre, después se le incluya más espacio para su apellido y por último que se libere la memoria.

Salida:

```
Nombre: Paco

Nombre completo: Paco Hernandez

El espacio se ha liberado!
```

Ejercicio 2: Realizar un programa que pida al usuario un número de elementos para reservar la memoria y llenarlos.

Salida:

```
Digite el numero de elementos: 4

Digite valores:
Digite un numero: 5
Digite un numero: 10
Digite un numero: 3
Digite un numero: 4
5
10
3
4
La memoria se ha liberado!: 003C0D50
```

Criterio de desempeño de la unidad V:	**Actividades de comprobación:**	**Actividades de evaluación :**
Manejo de archivos temporales y reservación, liberación y reasignación de memoria.	Desarrolle de aplicaciones utilizando archivos y memoria dinámica.	Elaborar programas que administren archivos y el manejo memoria.

Conclusiones

En conclusión este manual de ejercicios sirve de apoyo a la asignatura de programación estructurada con el uso del lenguaje de programación c++ dev, debido a que permite tener una compilación más rápida y que presenta herramientas de desarrollo con las funciones de bibliotecas, clases y objetos que es muy sencillo de instalar y práctico para llevar acabo los ejercicios propuestos de una manera fácil y sencilla de aprender, utilizando las sentencias adecuadas para cada problemática usándolas de forma correcta.

Referencias bibliográficas

- Benet Campderrich Falgueras (2002) ingeniería del software
- Brice-Arnaud Guérin (2005) Lenguaje c++.
- Harvey M. Deitel, Paul J. Deitel (2004) Cómo programar en C/C++ y Java.
- Jesús J. Rodríguez Salas (2003) Introducción a la programación: teoría y práctica.
- Joyce Farrell (2001) Introducción a la Programación lógica y diseño.
- Leobardo López R. (2006) Programación Estructurada
- Marco A. Peña Basurto, José María Cela Espín (2010) Introducción a la Programación en C.