Índice

1. Tema y Objetivos del Proyecto

1.1. Tema

El proyecto consiste en simular una tienda de supermercado en el lenguaje **C++** que incluirá tanto un administrador como clientes.

El administrador tendrá las siguientes funciones:

- Insertar producto
- Modificar producto
- Eliminar producto
- Mostrar los productos
- Cargar productos de un almacén (lectura de archivos)

Un cliente solo tiene la función de comprar. Las compras realizadas por los clientes se guardarán en un archivo de registro (escritura de archivos). Los productos en el almacén se almacenarán en una base de datos, utilizando específicamente el sistema de gestión de bases de datos relacional **MySQL**.

1.2. Objetivos

1.2.1. Objetivos Generales

- Demostrar las competencias adquiridas en el curso a través del desarrollo de un sistema que implemente el uso de punteros en estructuras de datos como pilas, colas y listas, entre otras. - Implementar los conceptos de lectura y escritura de archivos, además de incluir el almacenamiento de datos en una base de datos permanente.

Aquí tienes una versión mejorada:

1.2.2. Objetivos específicos

- Optimizar la administración del supermercado, mejorando el proceso de venta de productos y la experiencia de compra de los clientes. - Mantener un control de los productos disponibles en la tienda. - Proporcionar un proceso de compra y venta dinámico, facilitando la interacción eficiente entre la tienda y los clientes.

2. Descripción de las Funcionalidades del Proyecto

Crear un Nodo
Esta función solicita las propiedades de un producto, como el ID, nombre, precio y cantidad. Retorna una variable de tipo `TpNodo`. Nombre de la función: `CrearNodo()`

Insertar un Nuevo Producto
Agrega un nuevo producto (Nodo) a una lista de forma dinámica solicitando las propiedades del producto a través de la consola. Nombre de la función: `insertar_producto()`

Mostrar los Productos
Muestra todos los productos (Nodos) existentes y los guarda internamente en una base de datos. Nombre de la función: `mostrar()`

Modificar un Producto
Despliega los productos existentes y permite al administrador seleccionar el producto que desea modificar. Nombre de la función: `modificar_producto()`

Eliminar un Producto
Despliega los productos existentes y permite al administrador seleccionar el producto que desea eliminar. Nombre de la función: `eliminar_producto()`

Mostrar la Cola de Clientes
Muestra a los clientes que ingresan al supermercado en forma de cola. Nombre de la función: `mostrar_cola()`

Despachar a un Cliente
Elimina (desencola) a un cliente del supermercado. Nombre de la función: `desencolar_cliente()`

Cargar Productos
Lee un archivo de productos y los carga dinámicamente en una lista. Nombre de la función: `cargar_productos()`

Comprar Productos
Solicita la cantidad de productos que se desea adquirir y calcula el pago total de las compras. Nombre de la función: `comprar()`

Panel de Administración
Despliega todas las opciones disponibles para el administrador. Nombre de la función: `administrador()`

Buscar por ID de Producto
Devuelve una variable de tipo `int` que representa la posición del producto buscado. Nombre de la función: `buscar_nodo()`

Cantidad de Productos
Retorna una variable de tipo `int` que indica la cantidad de productos (Nodos) existentes en el supermercado. Nombre de la función: `nr_nodos()`

Stock de un Producto
Devuelve una variable de tipo `string` que representa la cantidad de un producto convertida a cadena para su concatenación en la función `mostrar()`. Nombre de la función: `stock()`

Guardar los Productos en la Base de Datos
Esta función recorre la lista de productos mediante un bucle `while` y los guarda en la tabla `Productos` de la base de datos `SuperMercado`. Nombre de la función: `guardar_db()`

3. Diseño de la solución, indicando las relaciones entre las funcionalidades

A continuación se muestra una descripción de las diversas funcionalidades que ofrece el programa.

- Resgistros (Nodos)
- Listas (alamacen)

- Colas (clientes)
- Archivos (productos)
- Base de datos (MySQL)

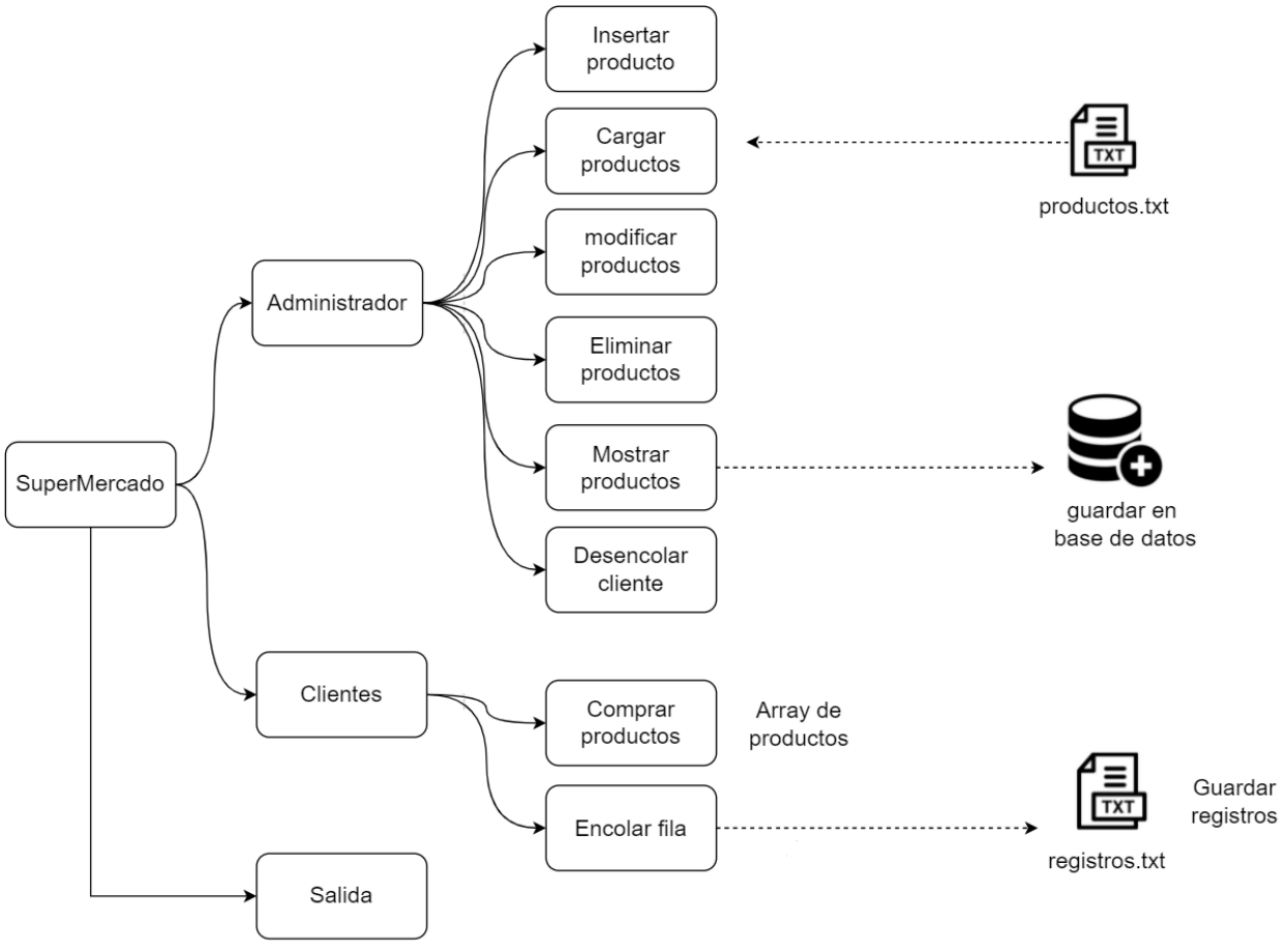

Figura 1: Esquema de la funcionalidad del programa

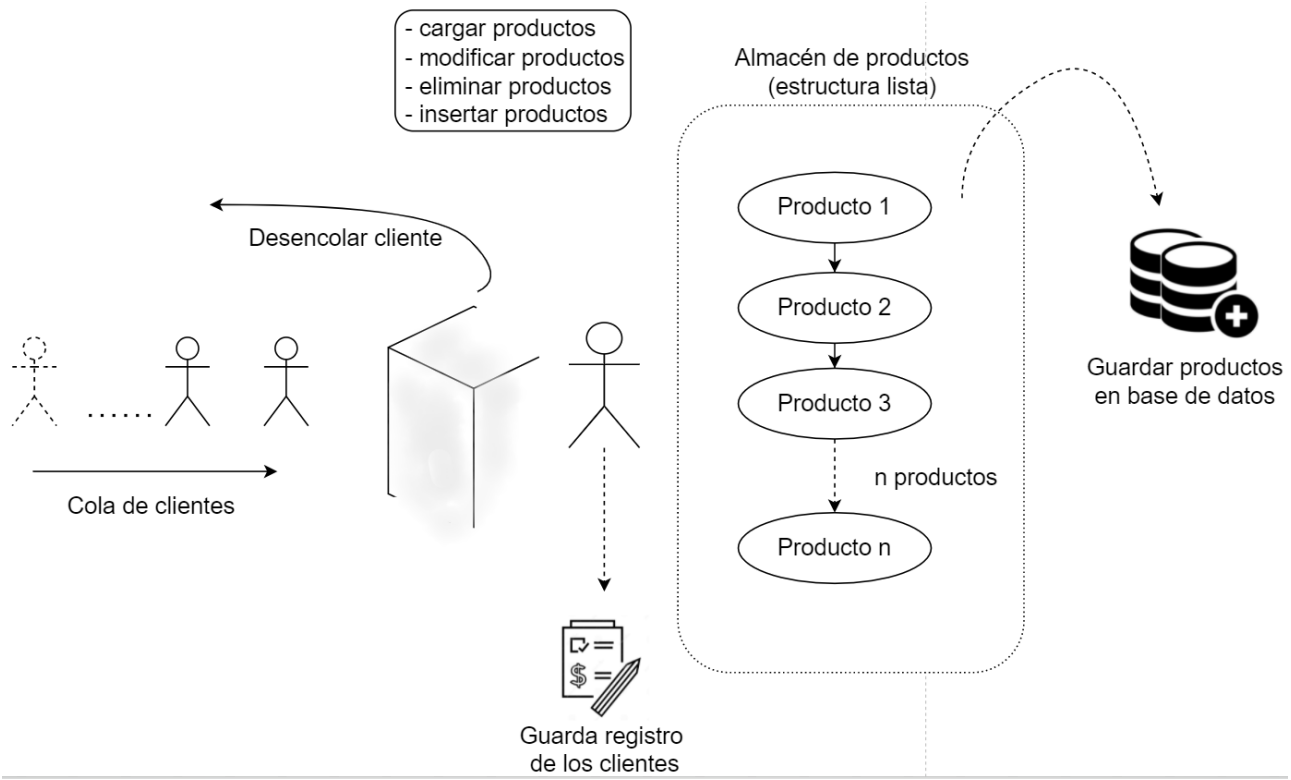

Figura 2: Simulación de la ejecución del programa

4. Resultados por cada funcionalidad y las estructuras de datos utilizadas

4.1. Variables globales

Inicialmente se define un tipo de dato **struct nodo** para representar la lista de productos y un tipo de dato **struct cliente** para gestionar la cola de clientes. Estas estructuras son fundamentales para el manejo de los datos en el sistema

```
typedef struct nodo *TpNodo;
TpNodo lista=NULL, temporal=NULL;

typedef struct cliente *TpCliente;
TpCliente cola_cliente = NULL;
```

Se define otra variable para iniciar una instancia de MySQL.

```
MYSQL *con = mysql_init(NULL);
```

4.2. Registros

Registro Nodo (productos)

```
struct nodo{
    int    id;
    string nombre;
    float  precio;
    int    cantidad;
    struct nodo *sgte;
};
```

Registro de cliente

```
struct cliente{
    int    nombre;
    string pago_total;
    struct nodo *sgte;
};
```

4.3. Funcionalidades más resaltantes

4.3.1. Cantidad de nodos

La función retornará un valor entero (int) que hará referencia a la cantidad de nodos de la lista haciendo un recorrido a través de un bucle while.
Se reutilizará en las siguientes funciones: mostrar(), modificar_producto(), eliminar_producto() y comprar()

```
int nr_nodos(){
  int nodos=0;
  TpNodo p=lista;
  while(p != NULL){
    p=p->sgte;
    nodos=nodos+1;
  }
  return nodos;
}
```

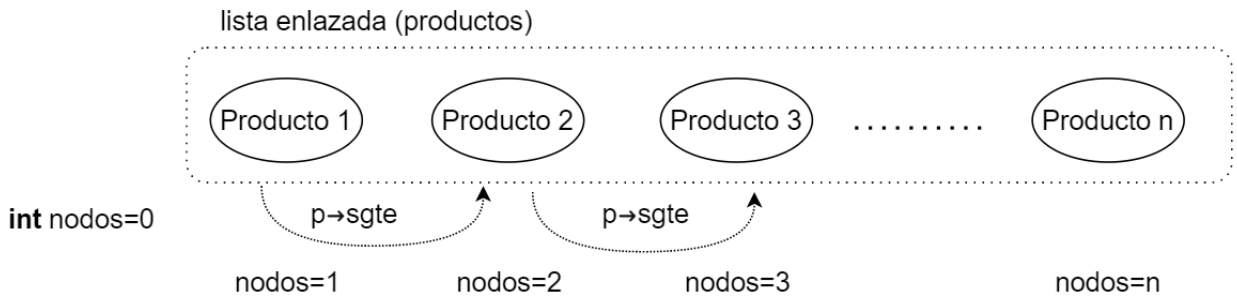

Figura 3: Esquema de la función nr_nodos()

4.3.2. Cargar productos desde un archivo

Dentro de esta función **cargar_productos()** se crea un objeto de la clase ifstream con el nombre de 'archivo', seguidamente con el método 'open()' le indico le indico el archivo que va a leer.

Con el modo **ios::in** se abre el fichero para leer datos.

```
  ifstream archivo;
  archivo.open("productos.txt", ios::in);
```

Posteriormente va a entrar el un bucle **while** donde leerá cada linea del archivo mientras no haya más información que leer.
Durante este bucle, se emplearon diversas técnicas para facilitar la lectura de las variables.

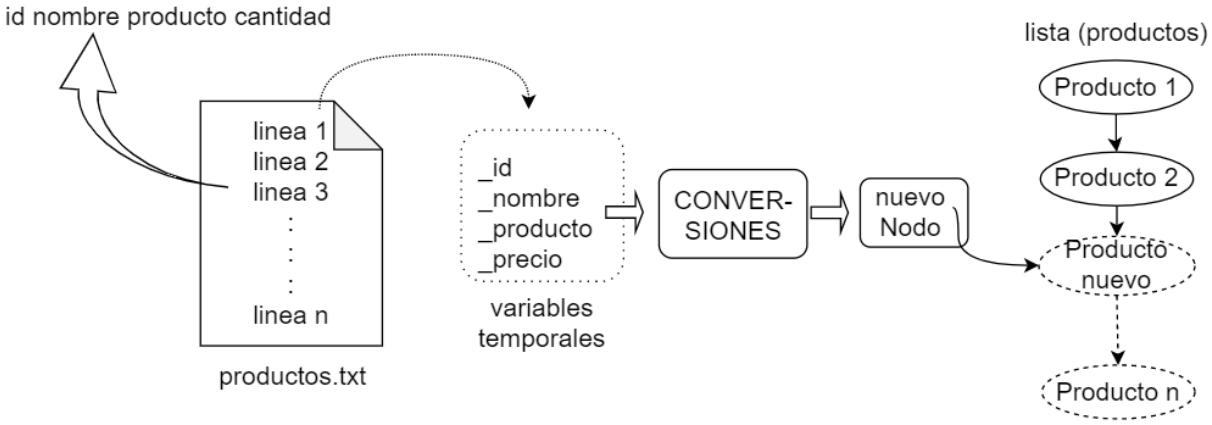

Figura 4: Lectura de archivo producto

A continuación se muestran las conversiones de variables que se usó:

4.3.2.1. Convertir string a entero

Para convertir una cadena a un entero utilizé la función **atoi**.
La primera función **c_str()** descarta todos los espacios vacíos hasta que encuentra un primer carácter, y es hasta ahí donde interpreta el valor numerico.

A continuación se muestra un código de ejemplo transformando una variable de tipo **string** a **int**.

```
int main(){
    //Declaracion de variables en string
    string numero;
    //pide y guarda los valores en numero
    cout << "Digite un numero: " << endl;
    cin >> numero;

    //convierte tu cadena a un entero le suma 2 y te muestra el resultado
    cout << "Tu numero es " <<  atoi(numero.c_str())+2;
    return 0;
}
```

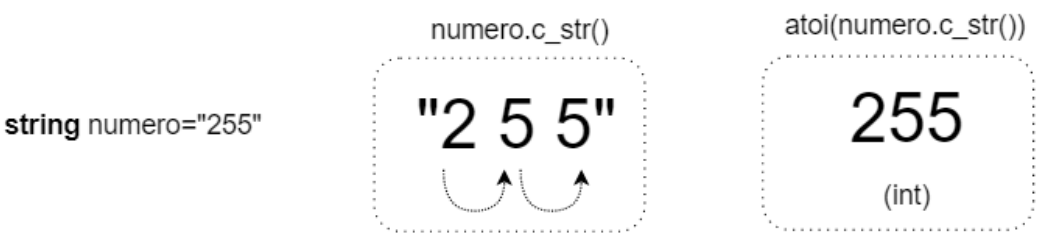

Figura 5: String a entero

4.3.2.2. Convetir arreglo char a variable float

Para realizar la transformación se hizo uso de la función **strtod()** que toma toma una cadena y un puntero a un carácter como parámetro, posteriormente interpreta el contenido de la cadena como un número flotante y devuelve ese valor.

Aquí se muestra un ejemplo extaido del siguiente link

```
int main()
{
    char s[100] = "4.0840";
    cout<<"valor flotante: "<<strtod(s,NULL)<<endl;
    printf("Float value : %4.3f\n",strtod(s,NULL));
    return 0;
}
```

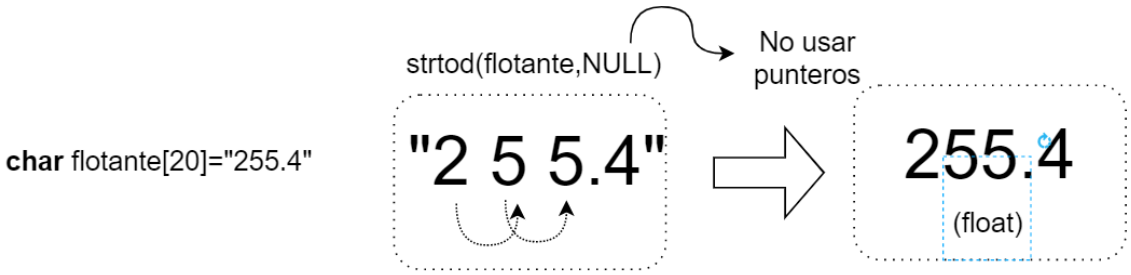

Figura 6: Arreglo char a float

Finalmente, así se visualizará en pantalla después de haber leído el archivo.

```
                    PRODUCTOS EXISTENTES:

ID              Nombre                  Precio          Cantidad
================================================================|
82301           aceite(1L)              7.9             12
91001           lata_atun               2.9             40
65659           pack_leche              19.5            10
54234           pollo(1kg)              8.49            25
87341           azucar(1kg)             10              34
98120           frijol(1kg)             6.2             65
12548           sal_marina(1kg)         1.89            14
65128           cebolla(250g)           0.95            65
54971           papaya(1kg)             4.99            65
65924           zanahoria(1kg)          2.99            76
================================================================|
[+] Productos totales: 10
```

Figura 7: Carga de productos

4.4. Función stock

Esta función recibirá como argumento una variable **int** y retornará una variable **string**.
Lo más resaltante de aquí es como se hizo esa transformación de variable.

```
string stock(int cantidad){
  stringstream temp;

    temp<<cantidad;
    string cant = temp.str(); // con

  if(cantidad<=0)
      return "Agotado";
  else
      return cant;
}
```

4.4.1. Variable entero a string

Dentro de la función se declara una variable **temp** de tipo **stringstream**, donde este tipo de dato es un buffer en memória que simula comportarse como un archivo.

Seguidamente se le pasará la función **str()** que devolverá una una copia de la variable temp de tipo string.

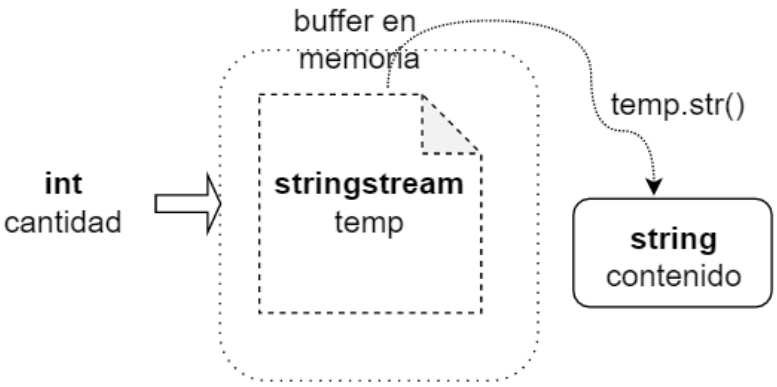

Figura 8: entero a string

4.5. Conexión con la base de datos

Primeramente se tiene que agregar la librería **mysql.h**.En mi caso al obtener errores por archivos dll faltantes o alguna incompatibilidad con la version del servicio MySQL obté por usar una distribución de Linux por ser más facil de configurar.
La única diferencia es en el nombre de la librería, en este caso sería **mariadb/mysql.h**

Aquí muestro algunos errores que me ocurrieron usando el sistema operativo Windows como primera opción.

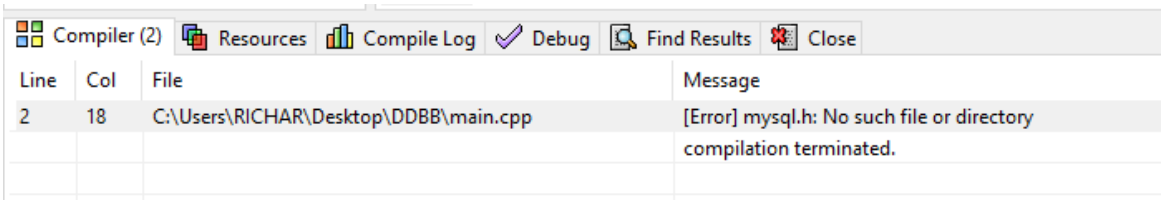

Figura 9: Librería no encontrada

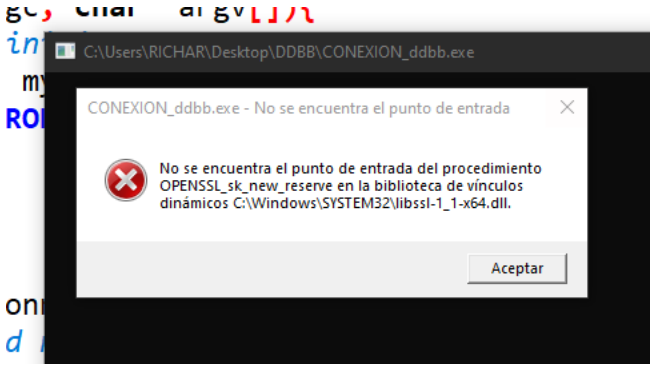

Figura 10: Archivo DLL faltante

Para la conexión con el servicio de **MySQL** se aplicó la siguiente función **mysql_real_connect()**.

Aquí se muestra el fragmento de código dentro de la función main().

```
[...]
if (mysql_real_connect(con, "localhost", "noroot", "noroot", "SuperMercado",
    3306, NULL, 0) == NULL){
      cout<<"\t\t[!] Error en la conexion"<<endl;
      system("pause");
      return 0;
  }
[...]
```

La función recibe los siguiente paramtros respectivemente: un conexión tipo puntero, la ip del equipo a conectar,usuario, contraseña,base de datos, puerto, socket y una flag.

```
mysql_real_connect(MYSQL *mysql,
                   char ip,
                   char user,
                   char passwd,
                   char db,
                   int port,
                   char socket,
                   long flag)
```

Para el proyecto elaborado los dos ultimo argumentos son irrelevantes, ya que estos se aplican para aplicaciones más complejas

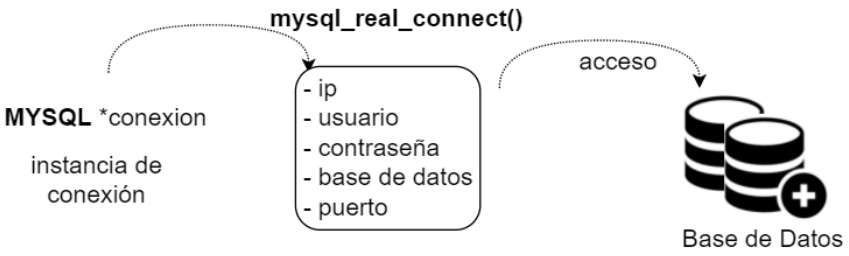

Figura 11: Esquema de la conexión

4.6. Guardar lista en la base de datos

En forma resumida se realizan son consultas en MySQL, la primera elimina la tabla **Productos** en caso de que exista y posteriormente vuelve a crearla.

```
mysql_query(con, "DROP TABLE IF EXISTS Productos");
  mysql_query(con, "CREATE TABLE Productos(id INT NOT NULL, nombre VARCHAR
    (20) NOT NULL, precio FLOAT NOT NULL, cantidad VARCHAR(20) NOT NULL);");
```

Seguidamente a través una variable 'p' que se iguala a 'lista' se aplica un bucle while en el cual se insertará los nodos de la lista guardandolo en la tabla **Productos**.

```
string consulta="INSERT INTO Productos VALUES("  +
            buff.str()          + ", '"  +
            p->nombre           + "', '" +
            buff2.str()         + "', '"  +
            stock(p->cantidad) + "');";
```

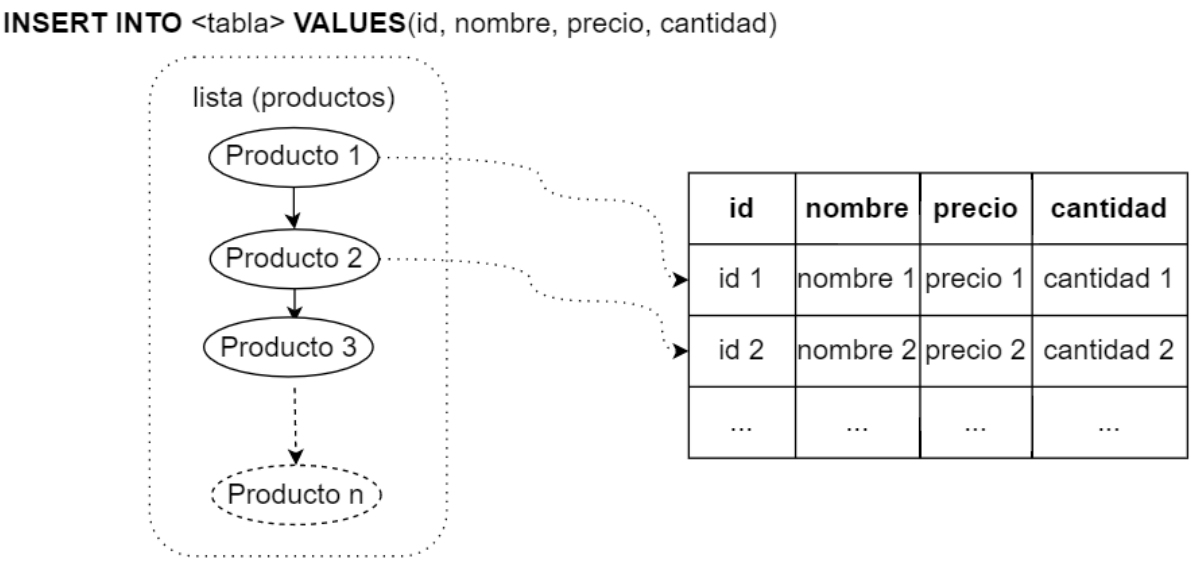

Figura 12: Inserción de productos

+ Opciones

id	nombre	precio	cantidad
82301	aceite(1L)	7.9	12
91001	lata_atun	2.9	40
65659	pack_leche	19.5	10
54234	pollo(1kg)	8.49	25
87341	azucar(1kg)	10	34
98120	frijol(1kg)	6.2	65
12548	sal_marina(1kg)	1.89	14
65128	cebolla(250g)	0.95	65
54971	papaya(1kg)	4.99	65
65924	zanahoria(1kg)	2.99	76

Figura 13: Tabla de productos guardados

4.6.1. Cola de clientes y escritura en archivo

El proceso se lleva a cabo en la funcion **encolar_cliente()**.
Primero se crea varibles para obtener el formarto de fecha y hora.

```
|============ LISTA DE NOMBRES DE CLIENTES ==============|
 Cliente 1: pepe
 Cliente 2: calamardo
 Cliente 3: patricio
```

Figura 14: Cola de clientes

4.6.1.1. Obtener el tiempo con la libreria time.h

Para acceder a las funciones y estructuras relacionadas con la fecha y la hora, es necesario incluir la librería **time.h**.

time_t: Este tipo de dato devolverá el tiempo de calendario actual del sistema en el número de segundos transcurridos desde el 1 de enero de 1970. Si el sistema no tiene tiempo, se devuelve 1.

struct tm tstruct:El tipo de estructura **tm** contiene la fecha y la hora en forma de una estructura C.

tstruct=*localtime(&t_actual): Esto devuelve un puntero a la estructura tm que representa la hora local.

strftime(): es una función que se usa para formatear la fecha y la hora. La sintaxis de strftime() es la que se muestra a continuación:

```
strftime(char *s, size_t max, const char *format, const struct tm *tm);
```

%Y: año decimal
%m: Mes como número decimal (01-12)
%d: día del mes en decimal
%X: cadena de tiempo estándar
La información fue extraida del siguiente enlace.

```
time_t t_actual = time(0);
struct tm tstruct;
char tiempo[80];
tstruct = *localtime(&t_actual);
strftime(tiempo, sizeof(tiempo), "Fecha: %Y-%m-%d\t\tHora: %X", &tstruct);
```

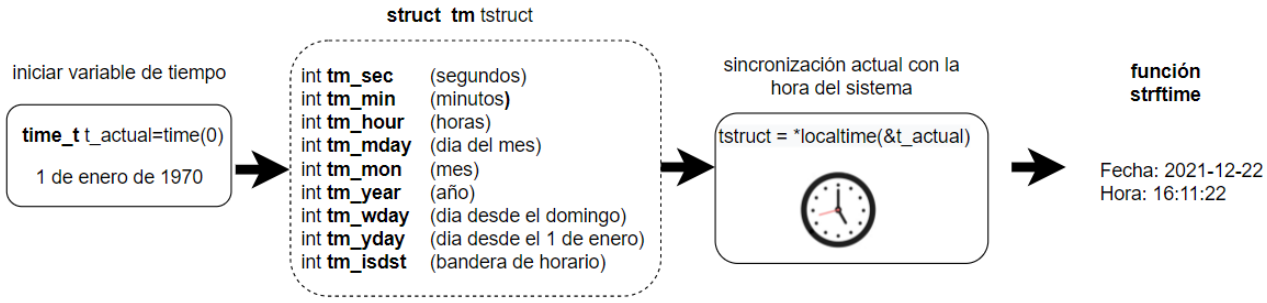

Figura 15: Obteniendo hora y fecha

Posteriormenete se creará una variable **ofstream** que abrirá un archivo en modo de salida encaso de que no exista lo creará.

```
[...]
ofstream archivo;
archivo.open("registro_clientes.txt",ios::app);
[...]
```

Por ultimo se entrará en un bucle while donde se insertará un nodo en la estructura de **cola_cliente**, está parte de código es similar a la función de 'ingresar nodo al final' que se vió en clase de pilas y colas.

Así se visualizaría el archivo de registros.

```
Archivo  Edición  Formato  Ver  Ayuda
Fecha: 2021-12-22        Hora: 16:11:22        Cliente: pepe          Pago_Total: S/ 54
Fecha: 2021-12-22        Hora: 16:27:38        Cliente: calamardo     Pago_Total: S/ 396
Fecha: 2021-12-22        Hora: 16:31:14        Cliente: patricio      Pago_Total: S/ 16
```

Figura 16: Registro de clientes

4.7. Cargar productos de la tabla Productos

Varibles y funciones de la librería **mysql.h** que se usaron para hacer las consultas SQL.

MYSQL
Esta estructura representa el controlador para una conexión de base de datos. Se utiliza para casi todas las funciones de MySQL.

MYSQL_ROW
Es tipo de dato que representa una fila de datos de una consulta SQL.

MYSQL_RES
Esta estructura representa el resultado de una consulta que devuelve filas

mysql_query
Realiza una consulta SQL.

mysql_store_result
Devuelve un conjunto de resultados almacenado en búfer de la última consulta ejecutada.

mysql_fetch_row
recupera la siguiente fila de un conjunto de resultados:

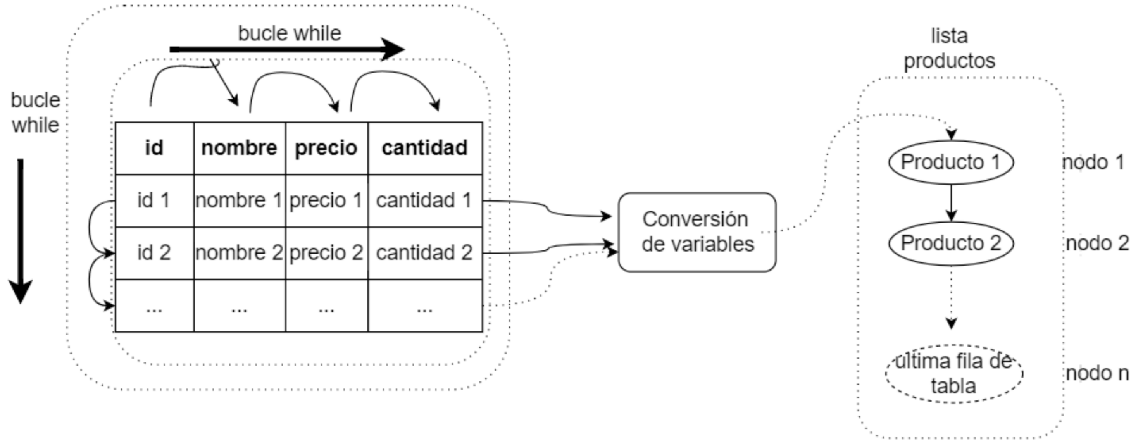

Figura 17: Cargar tabla Productos

4.8. Compra de productos

para un mejor análisis elaboré un pseudocódigo de la función **comprar()**.

```
SI lista=NULL:
  return
SINO:
  art=cantidad de productos
  MIENTRAS(i=1 <art):
    TpNodo actual=lista
    _id=id del producto
    pos=posicion del producto buscado

    SI(pos<=nr_nodos()):
      MIENTRAS(actual->id != id):
        actual=actual->sgte
      FIN MIENTRAS

      _cantidad=cantidad por producto

      SI(actual->cantidad < _cantidad):
        continue
      FIN_SI

      productos[]=actual->nombre
      c++

      pago=pago + (actual->precio * _cantidad)
      actual->cantidad = actual->cantidad - _cantidad
    SINO
      continue
    FIN_SI

  cliente=nombre de cliente

  PARA(i=0 hasta art)
    IMPRIMIR productos[i]
  FIN_PARA

  encola_cliente(cliente, pago)
  IMPRIMIR pago
FIN_SI
```

5. Conclusiones

El programa se centró basicamente en aplicar lo aprendido sobre las estructuras de dato como las listas enlazadas, adicionalmente se introdujo el el concepto de archivos (lectura y escritura) y finalmente para darle más realismo, conexión con una base de datos.

Complementarios:

- Agregación de colores
- Uso de la hora y fecha (time.h)
- Estética del programa

Puntos por mejorar

- Manejo adecuado de excepciones
- Implementación de una nueva tabla para los clientes
- Modularidad, dividir el programa en varios archivos

6. Referencias

- Convert Data Types. (s.f.). Convert C++. https://www.convertdatatypes.com/Convert-cPlusPlus.html
- Educative. (s.f.). How to convert a string to an int in C++. https://www.educative.io/edpresso/how-to-convert-a-string-to-an-int-in-cpp
- Stack Overflow. (s.f.). How to convert string to float. https://stackoverflow.com/questions/7951019/how-to-convert-string-to-float
- Yo soy dev. (s.f.). Cómo convertir cadena a entero en C. https://yosoy.dev/como-convertir-cadena-entero-en-c/
- Tutorialspoint. (s.f.). C++ Date and Time. https://www.tutorialspoint.com/cplusplus/cpp_date_time.htm
- GeeksforGeeks. (s.f.). strftime() function in C. https://www.geeksforgeeks.org/strftime-function-in-c/