



**FACULTAD DE INGENIERÍA Y CIENCIAS
EXACTAS**

TRABAJO PRÁCTICO N°0

**DISEÑO Y PROGRAMACIÓN
ORIENTADA A OBJETOS**

Docente: Ing. Fernando Abad

Alumno: Luciano Muratore

LU:1071433

Carrera: Ingeniería en Telecomunicaciones

- Indice
- Introduccion
- Objetivo
- Clase vector
 - Parámetros
 - Constructores
 - Constructor por Default
 - Constructor por Pase de Argumentos
 - Constructores por Copia
 - Destructores
 - Metodos
 - Sobrecarga de Operadores
 - Operador Suma
 - Operador Resta
 - Operador Multiplicación por Escalar
 - Operador Producto Escalar
 - Operador Producto Vectorial
 - Operador División por una Escalar
 - ++++Operador División(((falta hacer)))
 - Operador Igual
 - Operador IStream
 - Operador OStream

- Funciones (main.cpp)
 - Función Principal
 - Función Menú
 - Función invalida
 - Función Ayuda
 - Función Salir
 - Funciones Operacionales
 - Función Suma
 - Función Resta
 - Función DivisiónEscalar
 - Función MultiplicaciónEscalar
 - Función ProductoVectorial
 - Función Norma

Función Versor

Introducción

El presente informe detalla la estructura del programa desarrollado en lenguaje C++ con la intención de cumplir con las pautas preestablecidas.

El mismo está constituido por 3 archivos:

- a) vectores_dec.h: Archivo con la declaración de la clase `vector`.
- b) vectores_def.h: Archivo con la definición de la clase `vector`.
- c) main.cpp: Estructura principal del programa.

A continuación, se procede a desarrollar de manera detallada el contenido de cada archivo.

Objetivo

Desarrollar una aplicación de consola que permita realizar todas las operaciones matemáticas en vectores de 3 dimensiones:

- a) Sumar dos vectores.
- b) Restar dos vectores.
- c) Multiplicar un vector por una escalar.
- d) Dividir un vector por una escalar.
- e) Realizar el Producto Escalar entre dos vectores.
- f) Realizar el Producto Vectorial entre dos vectores.
- g) Calcular la norma de un vector.
- h) Calcular el versor de un vector.

Además, se implementarán una función Ayuda y una función Salida.

La función Ayuda busca dar a conocer al usuario la manera en que debe ingresar los vectores.

La función Salida finaliza la ejecución del programa.

Pre-Compilación

Se emplearon 4 bibliotecas:

- 1) `iostream`: Acceso a funciones básicas como `cin` o `cout`.

- 2) `cmath`: Acceso a funciones matemáticas como `sqrt`.
- 3) `vectores_def.h`: Acceso a la definición de la clase `vector`.
- 4) `vectores_dec.h`: Acceso a la declaración de la clase `vector`.

Clase Vector

Parámetros

Los tres parámetros que se utilizaron son privados y corresponden a los tres componentes del vector:

```
double x;  
double y;  
double z;
```

Constructores

Se declararon 3 tipos de Constructores:

a) Constructor por Defector

Definición

```
vector();
```

Declaración

```
vector::vector()  
:x(0),y(0),z(0){}
```

b) Constructor por Pase de Argumentos

Definición

```
vector(double , double , double);
```

Declaración

```
vector::vector(double x1, double x2, double x3)  
:x(x1),y(x2),z(x3){}
```

c) Constructor por Copia

Definición

```
vector(const vector &);
```

Declaracion

```
vector::vector(const vector &v)  
:x(v.x),y(v.y),z(v.z){}
```

Destructor

Método estándar de destrucción del objeto vector:

```
~vector();
```

Métodos

Todos los métodos declarados son públicos. En total son 5.

Los 3 siguientes métodos son utilizados para acceder a los datos privados de la clase *get_x()*, *get_y()*, y *get_z()*.

Declaración:

```
double get_x() const;  
double get_y() const;  
double get_z() const;
```

Definición:

```
double vector::get_x() const  
{  
    return x;  
}  
  
double vector::get_y() const  
{  
    return y;  
}  
  
double vector::get_z() const  
{
```

```
    return z;  
}
```

Los 2 métodos restantes son empleados para calcular la norma y el versor de un vector ingresado.

Declaración del método *Norma*

```
double const norma();
```

Definición del método *Norma*

```
double const vector::norma()  
{  
    return sqrt(x*x+y*y+z*z);  
}
```

Declaración del método *Versor*

```
vector const versor();
```

Definición del método *Versor*

```
vector const vector::versor()  
{  
    return vector(x/norma(), y/norma(), z/norma());  
}
```

Sobrecarga de Operadores

Se sobrecargaron los siguientes operadores para poder realizar las operaciones matemáticas de vectores.

Operador *Suma*

Dicho operador se encarga de sumar dos vectores y devuelve el vector resultado:

Declaración

```
friend vector const operator+(vector const &, vector const &);
```

Definición

```
vector const operator+(vector const & a1, vector const & a2)
{
    return vector (a1.get_x()+
a2.get_x(),a1.get_y()+a2.get_y(),a1.get_z()+a2.get_z());
}
```

Operador *Resta*

Se encarga de restar dos vectores y devuelve el vector resultado:

Declaración

```
friend vector const operator-(vector const &, vector const & );
```

Definición

```
vector const operator-(vector const & a1, vector const & a2)
{
    return vector (a1.get_x()+
a2.get_x(),a1.get_y()+a2.get_y(),a1.get_z()+a2.get_z());
}
```

Operador *Multipliación*

Se realizaron dos sobrecargas, uno correspondiente a la Multipliación de un vector por una escalar y el Producto Escalar.

Multipliación por un Escalar

Recibe un vector y una escalar como argumentos, y devuelve un dato de tipo double.

Declaración

```
friend vector const operator*(vector const & , double );
```

Definición

```
vector const operator*(vector const & a1, double l)
{
    return vector (a1.get_x()*l,a1.get_y()*l,a1.get_z()*l);
}
```

Producto Escalar

El mismo consiste en la recepción de dos vectores, y devuelve el producto escalar entre los dos vectores recibidos.

Declaración

```
friend double const operator*(vector const &, vector const &);
```

Definición

```
double const operator*(vector const & v1, vector const & v2) {
    double aux = v1.get_x() * v2.get_x() + v1.get_y() * v2.get_y() + v1.get_z() *
v2.get_z();

    return aux;
}
```

Producto Vectorial

Dado que se utilizan los mismo argumentos que antes, y se diferencia por el retorno, se decidió remover la parametrización const para que el compilador pueda diferenciarlos.

Declaración

```
friend vector operator*(vector &, vector &);
```

Definición

```
vector operator*(vector & v1, vector & v2) {
    vector v3(
        (v1.get_y() * v2.get_z()) - (v2.get_y() * v1.get_z()),
        (v1.get_x() * v2.get_z()) - (v2.get_x() * v1.get_z()),
        (v1.get_x() * v2.get_y()) - (v2.get_x() * v1.get_y())
    );
}
```



```
    return v3;
}
```

Operador *División* por una Escalar

Toma como argumentos un vector y una escalar, y devuelve un vector cuyas coordenadas son las del vector recibido dividido por la escalar.

Declaración

```
friend vector const operator%(vector const & , double );
```

Definición

```
vector const operator%(vector const & a1, double k)
{
    return vector (a1.get_x()/k,a1.get_y()/k,a1.get_z()/k);
}
```

Operador *Igual*

Se declara sin usar el parámetro friend.

Declaración

```
vector const &operator=(vector const &);
```

Definición

```
vector const & vector::operator=(vector const & v)
{
    x = v.get_x();
    y = v.get_y();
    z = v.get_z();
    return *this;
}
```

Operador *IStream*

Se lo utiliza para facilitar el ingreso de vectores por parte de los usuarios.

Declaración

```
friend std::istream &operator>>(std::istream &, vector &);
```

Definición

```
istream & operator >>(istream &in, vector & p)
{
    double x1 = 0;
    double x2 = 0;
    double x3 = 0;
    int ok = 0;
    char ch = 0;

    if (in >> ch
        && ch == '(') {
        if (in >> x1
            && in >> ch
            && ch == ', '
            && in >> x2
            && in >> ch
            && ch == ', '
            && in >> x3
            && in >> ch
            && ch == ')')
            ok = 1;
    }
    else {
        in.putback(ch);
        if (in >> x1)
            ok = 1;
    }

    if (ok)
        p.x=x1 , p.y=x2, p.z=x3;
```

```

    else
        in.clear(ios::badbit);

    cout << endl;
    return in;
}

```

Operador *OStream*

Se usa para facilitar la impresión por pantalla de un vector.

Declaración

```
friend std::ostream &operator<<(std::ostream &, const vector &);
```

Definición

```
ostream & operator <<(ostream &sortie, vector const & z)
{
    sortie<<' ('<<z.get_x()<<","<<z.get_y()<<","<<z.get_z()<<") ";
    return sortie;
}

```

Funciones (main.cpp)

Todas las funciones fueron declaradas en el archivo main.cpp.

Función Principal

La función principal se encargará de llamar al menú presentado al usuario, que mediante un switch basado en la opción elegida, invocará las distintas funciones tanto las vinculadas a las operaciones matemáticas como a las otras.

```
int main()
{

```

```
int eleccion;

do
{
    eleccion=menu();
    switch(eleccion)
    {
        case 1:
            suma();
            break;

        case 2:
            resta();

        case 3:
            ProductoEscalar();
            break;

        case 4:
            ProductoVectorial();
            break;

        case 5:
            norma();
            break;

        case 6:
            versor();
            break;

        case 7:
            MultiplicarporEscalar();
            break;

        case 8:
            DividirporEscalar();
            break;

        case 9:
```

```

        ayuda();
        break;

    case 10:
        salir();
        break;

    default:
        invalido();
        break;
}

system("pause");
cout<<endl;

}while (eleccion);

return EXIT_SUCCESS;
}

```

Función Menú

La función Menú muestra al usuario de manera escalonada las distintas operaciones que se pueden realizar con uno o dos vectores.

Además, retorna un entero que servirá para la elección de alguna de las operaciones.

Declaración

```
int menu(void);
```

Definición

```

int menu(void)
{
    int v;

    cout<<"-----MENU-----"
        <<endl

```

```
<<"Operaciones vectoriales"
<<endl
<<endl
<<"Elija una de las siguientes operaciones"
<<endl
<<"1-Suma"
<<endl
<<"2-Resta"
<<endl
<<"3-Producto Escalar"
<<endl
<<"4-Producto Vectorial"
<<endl
<<"5-Norma"
<<endl
<<"6-Versor"
<<endl
<<"7-Multiplicación por un Escalar"
<<endl
<<"8-Dividir por un Escalar"
<<endl
<<"9-Ayuda"
<<endl
<<"10-Salir"
<<endl;

cin>>v;

return v;
}
```

Ejemplo:

Se muestra a continuación el menú, que aparece al inicio del programa, por pantalla.

```
practicass_TP0 — main.exe — 78x24
muratorela@Lucianos-MacBook-Pro practicas_TP0 % ./main.exe
-----MENU-----
Operaciones vectoriales

Elija una de las siguientes operaciones:
1-Suma
2-Resta
3-Producto Escalar
4-Producto Vectorial
5-Norma
6-Versor
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
█
```

Función inválido

La misma se ejecuta cuando el usuario ingresa erróneamente una opción no declarada en el menú.

Declaración

```
void invalido(void);
```

Definición

```
void invalido(void) {
    cout<<"Cometio un error ingresando los datos"<<endl;
    cout<<"Hagalo de nuevo"<<endl;
}
```

Función ayuda

La misma se encarga de especificar la manera en que deben ser ingresados los datos.

Declaracion

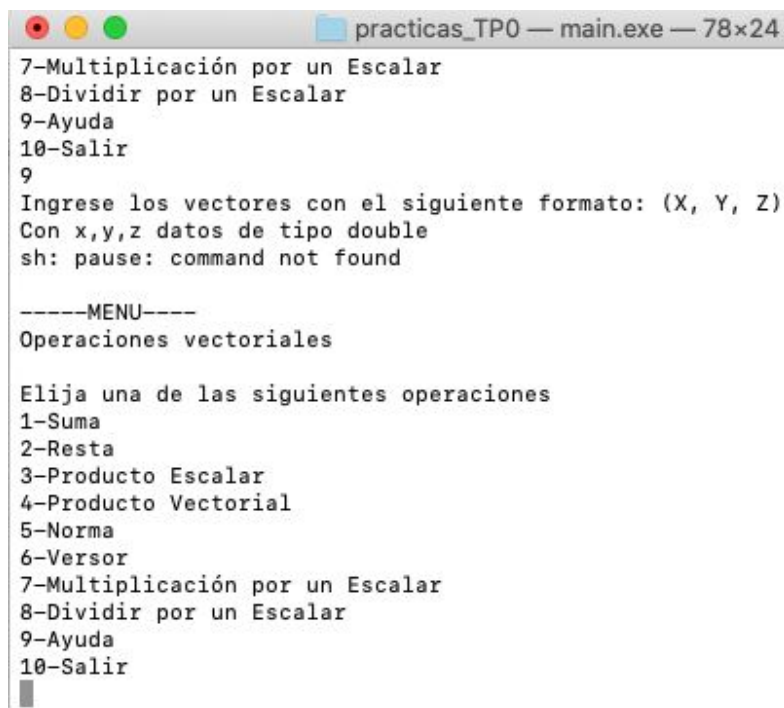
```
void ayuda(void);
```

Definición

```
void ayuda(void) {  
    cout << "Ingrese los vectores con el siguiente formato: (X, Y, Z)"<< endl;  
    cout<< "Con x,y,z datos de tipo double"<< endl;  
}
```

Ejemplo

Al ingresar el número 9, se imprime por pantalla cómo se deben ingresar los vectores.



```
practicass_TP0 — main.exe — 78x24  
7-Multiplicación por un Escalar  
8-Dividir por un Escalar  
9-Ayuda  
10-Salir  
9  
Ingrese los vectores con el siguiente formato: (X, Y, Z)  
Con x,y,z datos de tipo double  
sh: pause: command not found  
  
-----MENU-----  
Operaciones vectoriales  
  
Elija una de las siguientes operaciones  
1-Suma  
2-Resta  
3-Producto Escalar  
4-Producto Vectorial  
5-Norma  
6-Versor  
7-Multiplicación por un Escalar  
8-Dividir por un Escalar  
9-Ayuda  
10-Salir
```

Funcion Suma

La misma pide el ingreso de dos vectores y devuelve el vector resultante de la suma

Declaración

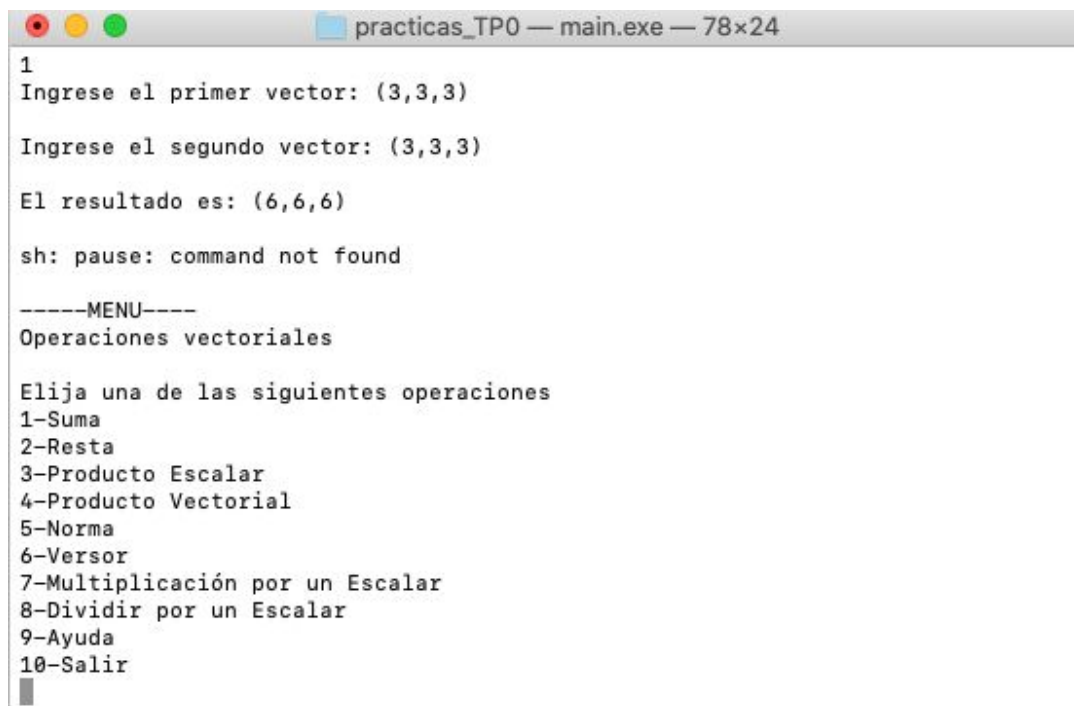

```
void suma(void);
```

Definición

```
void suma(void) {  
    vector  v1, v2;  
    cout << "Ingrese el primer vector: ";  
    cin >> v1;  
    cout << "Ingrese el segundo vector: ";  
    cin >> v2;  
    cout << "El resultado es: "  
        << v1 + v2  
        << endl  
        << endl;  
}
```

Ejemplo

El siguiente ejemplo muestra la suma entre los vectores (3,3,3) y (3,3,3).
Previamente, se ingresó 1 por teclado para seleccionar la opción Suma.



```
practicas_TP0 — main.exe — 78x24  
1  
Ingrese el primer vector: (3,3,3)  
Ingrese el segundo vector: (3,3,3)  
El resultado es: (6,6,6)  
sh: pause: command not found  
-----MENU-----  
Operaciones vectoriales  
Elija una de las siguientes operaciones  
1-Suma  
2-Resta  
3-Producto Escalar  
4-Producto Vectorial  
5-Norma  
6-Versor  
7-Multiplicación por un Escalar  
8-Dividir por un Escalar  
9-Ayuda  
10-Salir  
1
```

Funcion Resta

La misma pide el ingreso de dos vectores y devuelve el vector resultante de la resta.

Declaración

```
void resta(void);
```

Definicion

```
void resta(void) {  
    vector  v1, v2;  
    cout << "Ingrese el primer vector: ";  
    cin >> v1;  
    cout << "Ingrese el segundo vector: ";  
    cin >> v2;  
    cout << "El resultado es: "  
        << v1 - v2  
        << endl  
        << endl;  
}
```

Ejemplo

Se muestra la resta entre (1,1,1) y (22,2,2).

```
practicas_TP0 — main.exe — 80x24
2
Ingrese el primer vector: (1,1,1)

Ingrese el segundo vector: (22,2,2)

El resultado es: (23,3,3)

sh: pause: command not found

-----MENU-----
Operaciones vectoriales

Elija una de las siguientes operaciones
1-Suma
2-Resta
3-Producto Escalar
4-Producto Vectorial
5-Norma
6-Versor
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
```

Función Producto Escalar

Solicita al usuario dos vectores. Y luego se realiza el producto escalar.

Declaracion

```
void ProductoEscalar(void);
```

Definicion

```
void ProductoEscalar(void) {
    vector v1, v3;
    cout << "Ingrese el primer vector: ";
    cin >> v1;
    const vector v2(v1);
    cout << "Ingrese el segundo vector: ";
    cin >> v3;
    const vector v4(v3);
    cout << "El resultado es: "
    << v2 * v4
    << endl
    << endl;
```

```
}
```

Ejemplo

Se muestra el Producto escalar entre dos vectores.

Ambos vectores elegidos del ejemplo son (1,1,1).

```
practicass_TP0 — main.exe — 80x24
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
3
Ingrese el primer vector: (1,1,1)
Ingrese el segundo vector: (1,1,1)
El resultado es: 3
sh: pause: command not found
-----MENU-----
Operaciones vectoriales
Elija una de las siguientes operaciones
1-Suma
2-Resta
3-Producto Escalar
4-Producto Vectorial
5-Norma
6-Versor
7-Multiplicación por un Escalar
```

Función Producto Vectorial

La misma tiene como parámetros dos vectores. Y devuelve el vector resultante de la operación Producto Vectorial.

Definición

```
void ProductoVectorial(void);
```

Declaración

```

void ProductoVectorial(void) {
    vector  v1, v2;
    cout << "Ingrese el primer vector: ";
    cin >> v1;
    cout << "Ingrese el segundo vector: ";
    cin >> v2;
    cout << "El resultado es: "
    << v1 * v2
    << endl
    << endl;
}

```

Ejemplo

```

practicas_TP0 — main.exe — 80x24
4
Ingrese el primer vector: (1,1,1)
Ingrese el segundo vector: (2,2,2)
El resultado es: (0,0,0)
sh: pause: command not found
-----MENU-----
Operaciones vectoriales
Elija una de las siguientes operaciones
1-Suma
2-Resta
3-Producto Escalar
4-Producto Vectorial
5-Norma
6-Versor
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
█

```

Se muestra el resultado de realizar el producto vectorial entre (1,1,1) y (2,2,2).

Función Norma

Declaración

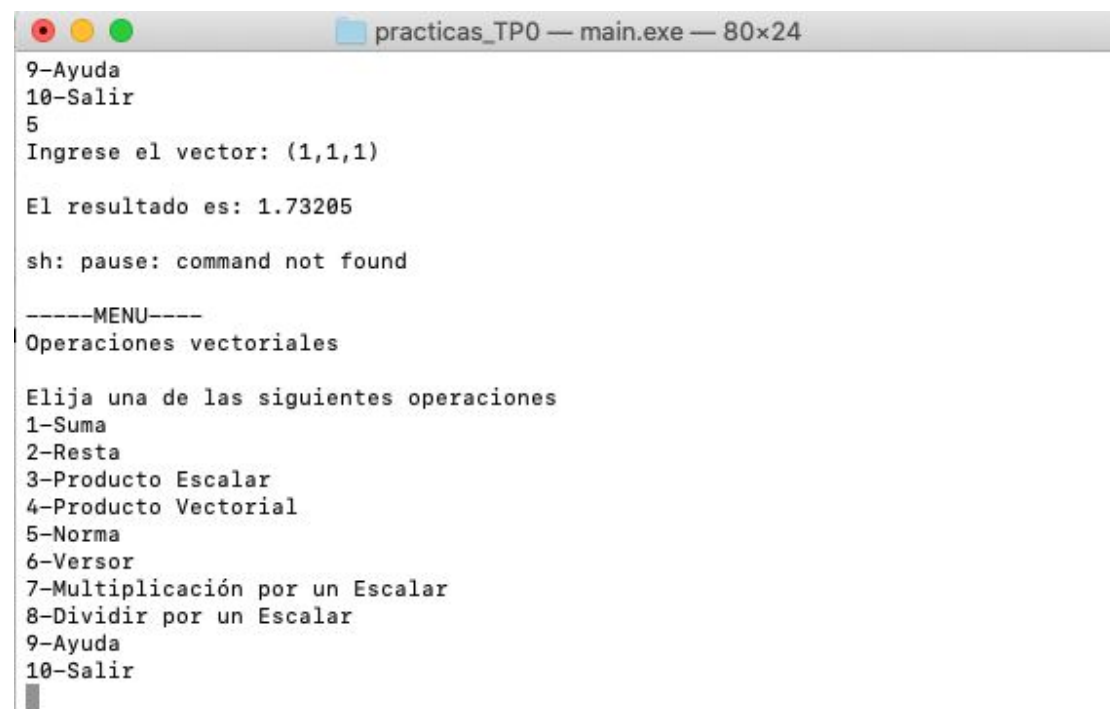
```
void norma(void);
```

Definición

```
void norma(void) {  
    vector v1;  
    cout << "Ingrese el vector: ";  
    cin >> v1;  
    cout << "El resultado es: "  
        << v1.norma()  
        << endl  
        << endl;  
}
```

En esta función se implementó el método `norma()` de la clase `vector`.

Ejemplo



```
practicas_TP0 — main.exe — 80x24  
9-Ayuda  
10-Salir  
5  
Ingrese el vector: (1,1,1)  
  
El resultado es: 1.73205  
  
sh: pause: command not found  
  
-----MENU-----  
Operaciones vectoriales  
  
Elija una de las siguientes operaciones  
1-Suma  
2-Resta  
3-Producto Escalar  
4-Producto Vectorial  
5-Norma  
6-Versor  
7-Multiplicación por un Escalar  
8-Dividir por un Escalar  
9-Ayuda  
10-Salir  
█
```

Se ingresó el vector (1,1,1). Y se calculó su norma.

Función Versor

También se volvió a usar el método `norma()` de la clase `vector`, dado que para calcular el versor de un vector se divide a cada coordenada por su norma, respectivamente.

Definición

```
void versor(void);
```

Declaración

```
void versor(void) {  
    vector v1;  
    cout << "Ingrese el vector: ";  
    cin >> v1;  
  
    if (v1.norma() == 1) {  
        cout << v1  
            << " ya es un versor."  
            << endl  
            << endl;  
    }  
    else {  
        cout << "El resultado es: "  
            << v1.versor()  
            << endl  
            << endl;  
    };  
}
```

Ejemplo

```
practicas_TP0 — main.exe — 80x24
9-Ayuda
10-Salir
6
Ingrese el vector: (11,1,1)

El resultado es: (0.991837,0.090167,0.090167)

sh: pause: command not found

-----MENU-----
Operaciones vectoriales

Elija una de las siguientes operaciones
1-Suma
2-Resta
3-Producto Escalar
4-Producto Vectorial
5-Norma
6-Versor
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
█
```

Luego de ingresar el vector (11,1,1), se calculo el versor.
Función Multiplicación por un Escalar

Definición

```
void MultiplicarporEscalar(void);
```

Declaración

```
void MultiplicarporEscalar(void)
{
    double a;
    vector v1;
    cout<<"Ingrese el vector: "<<endl;
    cin>>v1;
    cout<<"Ingrese el escalar: "<<endl;
    cin>>a;
    cout<<endl<<"El resultado es: "<<v1*a<<endl<<endl;
}
```

Ejemplo

Se multiplica 8 por el vector (1,1,1).


```
practicas_TPO — main.exe — 80x24
(1,1,1)

Ingrese el escalar:
8

El resultado es: (8,8,8)

sh: pause: command not found

-----MENU-----
Operaciones vectoriales

Elija una de las siguientes operaciones
1-Suma
2-Resta
3-Producto Escalar
4-Producto Vectorial
5-Norma
6-Versor
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
█
```

Función División por una Escalar

Definición

```
void DividirporEscalar(void);
```

Declaración

```
void DividirporEscalar(void) {
    double a;
    vector v1;
    cout << "Ingrese el vector: ";
    cin >> v1;
    cout << "Ingrese el escalar: ";
    cin >> a;
    cout << endl
        << "El resultado es: "
        << v1 / a
        << endl
        << endl;
}
```

Ejemplo

```
practicas_TP0 — main.exe — 80x24
8
Ingrese el vector: (2,2,2)

Ingrese el escalar: 2

El resultado es: (1,1,1)

sh: pause: command not found

-----MENU-----
Operaciones vectoriales

Elija una de las siguientes operaciones
1-Suma
2-Resta
3-Producto Escalar
4-Producto Vectorial
5-Norma
6-Versor
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
_
```

Se dividió a cada una de las coordenadas del vector (2,2,2) por 2.

Función Salir

La función salida se encarga de despedir al usuario, y enviarle el número de salida a la función principal para que sepa que debe cerrar el programa.

Definición

```
int salir(void);
```

Declaración

```
int salir(void) {
    cout<< "Adios"<< endl<< endl;
    return 0;
}
```

Ejemplo

Al elegir la opción 10, se imprime por pantalla "Adios"

```
practicas_TP0 — main.exe — 80x24
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
10
Adios

sh: pause: command not found

-----MENU-----
Operaciones vectoriales

Elija una de las siguientes operaciones
1-Suma
2-Resta
3-Producto Escalar
4-Producto Vectorial
5-Norma
6-Versor
7-Multiplicación por un Escalar
8-Dividir por un Escalar
9-Ayuda
10-Salir
```