

Guía 1: Repaso y aplicación de funciones.

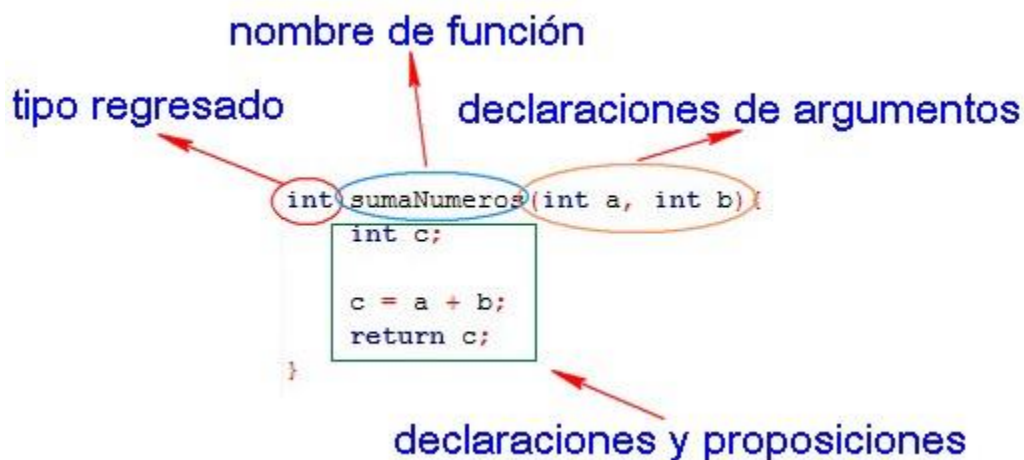
Objetivos

- Conocer los diferentes tipos de funciones que se pueden crear e implementar en **c++**.
- Utilizar funciones para optimizar el código.
- Aprender a utilizar el tipo de datos “struct” y visualizar sus aplicaciones.

Desarrollar una solución al problema planteado utilizando funciones y estructuras propias:

El uso de funciones convierte una pieza de código en una parte reutilizable y fácilmente adaptable sin tener que modificar todo el contenido de nuestro código, estas son solo algunas de las ventajas de dividir en módulos nuestro código convirtiéndolo así en pequeños segmentos.

A continuación un pequeño ejemplo:



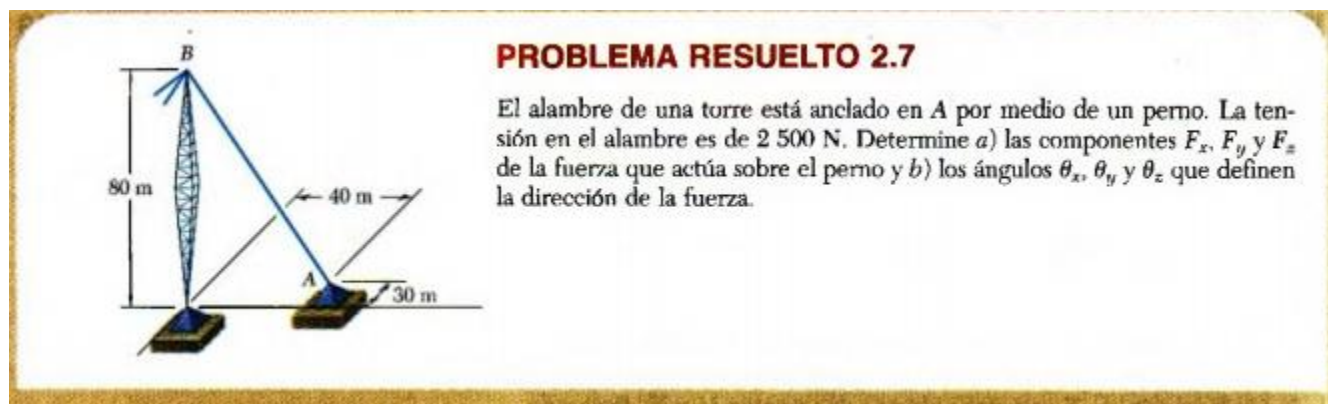
En la imagen se observa el cómo se compone una función.

Programación II

Desarrollo de un software capaz de encontrar el vector resultante en el espacio tridimensional dado n vectores:

En el desarrollo de la materia de estática se estudian las cargas, el equilibrio de fuerzas en sistemas estáticamente equilibrados.

Por lo que se analizan las fuerzas que generan n vectores con su respectiva tensión sobre un punto determinado a continuación se muestra un ejemplo:



El problema corresponde al libro: “Mecánica Vectorial para Ingenieros Estática”, Octava edición, Editorial McGraw Hill, página 50-51.

El cual muestra un sistema tridimensional estático, el cual tiene 2 puntos que son A, B. En el ejercicio se desea conocer los componentes F_x , F_y , F_z de la fuerza resultante sobre el perno (Punto A) y los ángulos θ_x , θ_y , θ_z los cuales son la dirección, esto dado la tensión de 2500 Newton que existen en el vector AB.

Análisis de la solución:

Se procede a determinar las variables que intervienen en la solución:

Cantidad Puntos = 2

Cantidad de fuerzas = 1

Tensión AB = 2500 N

Coordenadas : A(40, 0 , -30) , B (0 , 80 , 0)

Se necesita conocer las componentes del vector AB por lo que se utilizan las ecuaciones para definir la trayectoria en i j k.

Programación II

Formula:

$$\overrightarrow{AB} = d_x i + d_y j + d_z k$$

Donde

$$d_x = x_2 - x_1 \quad d_y = y_2 - y_1 \quad d_z = z_2 - z_1$$

Aplicación:

$$\overrightarrow{AB} = ((0) - (40))i + ((80) - (0))j + ((0) - (-30))k$$

$$\overrightarrow{AB} = -40i + 80j + 30k$$

La magnitud se define por:

$$AB = \sqrt{d_x^2 + d_y^2 + d_z^2} = \sqrt{(-40)^2 + (80)^2 + (30)^2} = 94.3$$

Encontrando el vector resultante **F**:

$$\mathbf{F} = F\lambda = F \frac{\overrightarrow{AB}}{AB} = \frac{2500}{94.3} [-40i + 80j + 30k]$$

Por lo que operando queda como resultado:

$$\mathbf{F} = (-1060)i + (2120)j + (795)k$$

Para obtener la dirección se despeja el coseno del ángulo correspondiente a la componente de la fuerza **F**.

$$\theta_x = \cos^{-1}\left(\frac{F_x}{F}\right) = \cos^{-1}\left(\frac{-1060}{2500}\right) = 115.1^\circ$$

$$\theta_y = \cos^{-1}\left(\frac{F_y}{F}\right) = \cos^{-1}\left(\frac{2120}{2500}\right) = 32.1^\circ$$

$$\theta_z = \cos^{-1}\left(\frac{F_z}{F}\right) = \cos^{-1}\left(\frac{795}{2500}\right) = 71.5^\circ$$

La función acos representa la función arco coseno.

Con esto concluye el desarrollo del ejercicio del cual se tienen la solución de **F** también su dirección expresada en grados.

Programación II

Implementación en código:

Una vez completado el análisis del problema se procede a codificar una solución genérica para este tipo de problemas, por lo que se definen estructuras básicas que contendrán nuestros datos definidos de la siguiente manera:

```
1  #include <iostream> //LIBRERIA ESTANDAR DE SALIDA
2  #include <cstring> //LIBRERIA DE MANEJO DE STRING
3  #include <math.h> //LIBRERIA DE FUNCIONES MATEMATICAS
4  /*
5   *solucion a problemas de encontrar vector resultante de n fuerzas en el espacio
6   *autor : jonathan geovany hernandez vasquez
7   *fecha : 22-02-2016
8   */
9  using namespace std;
10 #define PI 3.14159265 //definiendo PI
11 int cantidad_puntos, cantidad_fuerzas; //variables globales
12
13 struct punto { //estructura basica que define un punto en el espacio
14     float x=0;
15     float y=0;
16     float z=0;
17     char id;
18 };
19
20 struct componente { //estructura utilizada para contener componentes basicos de un vector
21     string identificador;
22     float i=0;
23     float j=0;
24     float k=0;
25     float magnitud=0;
26     float tension=0;
27     float lambda=0;
28 };
29
```

Se observan la inclusión de la librería <math.h> que nos aporta una serie de funciones matemáticas, además del uso de variables globales, así como las estructuras.

Programación II

A continuación se crean una serie de funciones para simplificar las operaciones ya sea de búsqueda y validación de nuestros datos.

```
30 bool existe_vector(string id,punto p[]){
31     //retorna true si existe el Vector y false ya sea uno de los puntos no exista
32     bool p1=false,p2=false;//por defecto false
33     for(int i=0; i < cantidad_puntos ;i++){
34         if( p[i].id == id[0] ){
35             p1 = true;//solo cambia a estado true cuando encuentra la coincidencia
36         }
37         if( p[i].id == id[1]){
38             p2 = true;
39         }
40     }
41     return (p1 && p2);//retornamos el resultado de and de p1 y p2
42 }
43
44 int pos_punto(char id,punto puntos[]){
45     //retorna la posicion dado el identificador de el punto
46     for(int i=0;i< cantidad_puntos ;i++){
47         if(id == puntos[i].id){
48             return i;//una vez se encuentra se retorna inmediatamente
49         }
50     }
51     return 0;//si no la encontro retorna cero por defecto
52 }
53
```

La función `existe_vector` recibe como parámetro el id del vector, ejemplo AB luego descompone la palabra en dos partes para así buscar el id correspondiente en el vector de puntos que también se le pasa como parámetro, y es una función tipo bool por lo que nos retorna true si el id es válido y false si no lo es.

La función `pos_punto` busca dentro de un vector de puntos el punto correspondiente al id que se le envía como parámetro y nos retorna la posición inmediatamente una vez sea encontrada y si no lo encuentra nos retorna cero.

Programación II

La aplicación de funciones que implemente paso por referencia simplifica el uso de variables ya que copiamos la dirección de memoria en otra variable, es decir es una copia idéntica y si una de las variables es cambiada las demás también lo son.

```

55 void det_componentes(componente &c,punto p){//paso por referencia del componente c
56     //buscando la posicion correspondiente a los puntos
57     int p1=pos_punto(c.identificador[0],p);
58     int p2=pos_punto(c.identificador[1],p);
59     //encontrando la direccion
60     c.i = p[p2].x - p[p1].x;
61     c.j = p[p2].y - p[p1].y;
62     c.k = p[p2].z - p[p1].z;
63     //mostrando los componentes i j k
64     cout << "\n" << c.identificador << " (" << c.i <<")i + ("<<c.j<<")j + (" << c.k << ")k\n";
65     //encontrando la magnitud y lambda
66     c.magnitud = sqrt( pow(c.i,2) + pow(c.j,2) + pow(c.k,2) );
67     c.lambda = c.tension / c.magnitud;
68     //mostrando la magnitud
69
70     cout << "Magnitud " << c.identificador << " " << c.magnitud << endl;
71     cout << "Lambda " << c.identificador << " " << c.lambda << endl;
72
73     //encontrando la tension y almacenandola en los componentes
74     c.i = c.lambda * c.i;
75     c.j = c.lambda * c.j;
76     c.k = c.lambda * c.k;
77     cout << "Tension " << c.identificador << " (" << c.i <<")i + ("<<c.j<<")j + (" << c.k << ")k\n\n";
78 }
79

```

La función det_componentes corresponde a determinar los componentes básicos de un vector, dado la dirección de ese vector ejemplo AB del punto A al punto B, para ello se le pasa como referencia el componente y también el vector de puntos.

Realiza primeramente las operaciones de la ecuación:

$$\overrightarrow{AB} = d_x i + d_y j + d_z k$$

Luego la magnitud se calcula usando la función sqrt(número) y también la función pow(base, exponente) las cuales son raíz cuadrada y la función potencia, respectivamente.

$$AB = \sqrt{d_x^2 + d_y^2 + d_z^2}$$

Luego se sobre escribe los valores de c.i c.j c.k con la multiplicación de lambda en cada una de sus componentes

Programación II

Por último la función que sumara todas las componentes de las fuerzas involucradas para así retornar la fuerza resultante, también su dirección y magnitud.

```

80 void det_resultante(componente c[]){
81     float resultante=0;//magnitud del vector resultante de fuerzas
82     float R[3] = {0,0,0};//vector resultante y sus componetes i,j,k
83     float dir[3]);//en grados sexagesimales
84     for(int i=0; i< cantidad_fuerzas ; i++){//sumar todas las componentes de las fuerzas
85         R[0] += c[i].i;
86         R[1] += c[i].j;
87         R[2] += c[i].k;
88     }
89     resultante = sqrt( pow(R[0],2) + pow(R[1],2) + pow(R[2],2) );//calculando la magnitud resultante
90     cout << "\nFuerza resultante : (" << R[0] << ")i + (" << R[1] << ")j + (" << R[2] << ")k \n";
91     cout << "Magnitud resultante : " << resultante;
92     for(int i=0;i<3;i++){
93         dir[i] = acos( R[i]/resultante) * 180 / PI;//arco coseno y luego convertir a grados
94     }
95     cout << "\nDireccion de la fuerza (grados) : (" << dir[0] << ")x + (" << dir[1] << ")y + (" << dir[2] << ")z \n";
96 }
97

```

Se multiplica por 180 y se divide por PI, dado que el resultado de la función acos (arco coseno) se expresa en radianes, es necesario convertirlos a grados sexagesimales.

Implementación en la función main:

Para la implementación en la función main se declaran dos variables necesarias para poder desarrollar el ejercicio las cuales son : el identificador una variable tipo string que contiene el identificador del vector y un vector de tipo componente el cual se define el tamaño mediante una información proporcionada por el usuario.

```

97
98 int main()
99 {
100     string identificador;
101     cout << "Calculo de vector resultante en el espacio" << endl;
102     cout << "Digite la cantidad de puntos que inciden en el problema (Ej: A , B son 2 )\n : ";
103     cin >> cantidad_puntos;
104
105     punto puntos[cantidad_puntos];
106
107     for(int i=0;i < cantidad_puntos;i++){
108         cout << "Digite el identificador de el punto " << (i+1) << " (Ej: A)\n : ";
109         cin >> puntos[i].id;
110         cout << "Coordenada x de " << puntos[i].id << "\n : ";
111         cin >> puntos[i].x;
112         cout << "Coordenada y de " << puntos[i].id << "\n : ";
113         cin >> puntos[i].y;
114         cout << "Coordenada z de " << puntos[i].id << "\n : ";
115         cin >> puntos[i].z;
116     }//fin for

```

Programación II

A continuación se capturan los datos de las tensiones y también cuantos vectores inciden en el problema.

```
118
119     cout << "Cantidad de fuerzas o tensiones que inciden (Ej : AB , AC son 2 ) \n : ";
120     cin >> cantidad_fuerzas;
121     componente componentes[cantidad_fuerzas];
122     for(int i=0;i<cantidad_fuerzas;i++){
123
124         do{
125             cout << "Digita la direccion de la fuerza "<< (i+1) << " (Ej : AB ) \n : ";
126             cin >> identificador;
127             if(existe_vector(identificador,puntos) != true)
128             {
129                 cout << "Direccion invalida vuelve a intentarlo \n";
130             }
131
132         }while(existe_vector(identificador,puntos)==false); //mientras no exista el vector definido (Ej AB)
133         // si la direccion esta bien sale del bucle
134         componentes[i].identificador = identificador;
135         cout << "Digita la tension o fuerza de "<< identificador<< " \n : ";
136         cin >> componentes[i].tension;
137         det_componentes(componentes[i],puntos);
138     } //fin for
139     det_resultante(componentes);
140     return 0;
141 }
```

El ciclo do while que se encuentra dentro del ciclo for sirve para validar el ingresos de un vector correcto es decir que existan los puntos, de lo contrario se pedirá nuevamente datos correctos, una vez se digite una dirección correcta se procede a salir del ciclo do while y así procedemos a calcular las componentes dentro del ciclo for.

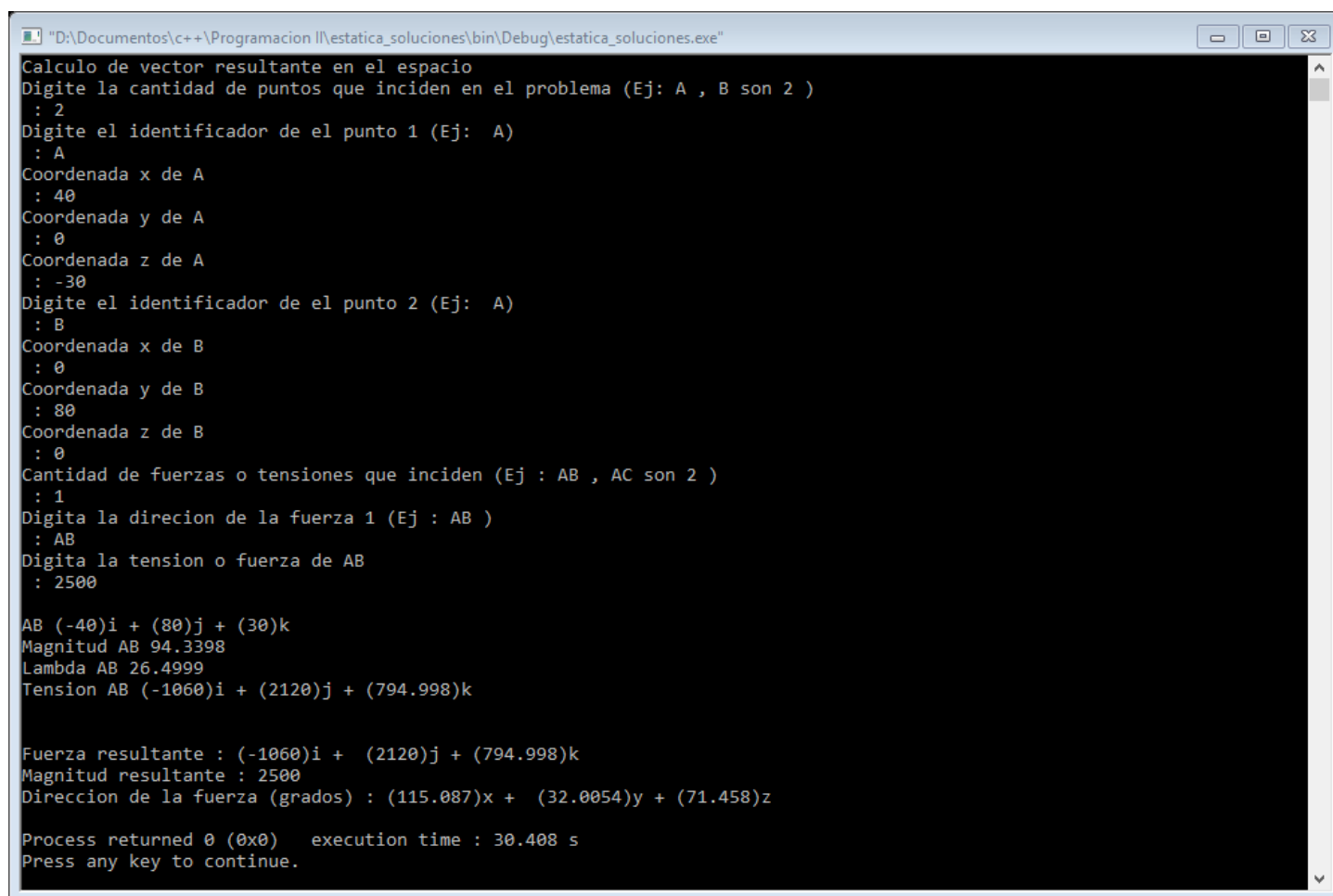
Una vez finalizado el ciclo for y evaluado los componentes de manera individual se procede a invocar la función det_resultante para mostrar los resultados finales.

Programación II

Evaluando el software:

Una vez finalizado el código y ya compilado sin errores procedemos a evaluar el software:

Utilizando como ejemplo el ejercicio 2.7 y comparando los resultados verificamos si cumple con lo deseado.



```
"D:\Documentos\c++\Programacion II\estatica_soluciones\bin\Debug\estatica_soluciones.exe"
Calculo de vector resultante en el espacio
Digite la cantidad de puntos que inciden en el problema (Ej: A , B son 2 )
: 2
Digite el identificador de el punto 1 (Ej: A)
: A
Coordenada x de A
: 40
Coordenada y de A
: 0
Coordenada z de A
: -30
Digite el identificador de el punto 2 (Ej: A)
: B
Coordenada x de B
: 0
Coordenada y de B
: 80
Coordenada z de B
: 0
Cantidad de fuerzas o tensiones que inciden (Ej : AB , AC son 2 )
: 1
Digita la direccion de la fuerza 1 (Ej : AB )
: AB
Digita la tension o fuerza de AB
: 2500

AB (-40)i + (80)j + (30)k
Magnitud AB 94.3398
Lambda AB 26.4999
Tension AB (-1060)i + (2120)j + (794.998)k

Fuerza resultante : (-1060)i + (2120)j + (794.998)k
Magnitud resultante : 2500
Direccion de la fuerza (grados) : (115.087)x + (32.0054)y + (71.458)z

Process returned 0 (0x0)   execution time : 30.408 s
Press any key to continue.
```

Como resultado el software proporciona las operaciones de manera correcta.

Ejercicio:

Dado el código y explicación de las funciones:

- Implemente una función que abarque el contenido de la función main.
- Crear un menú con la operaciones básicas (nuevo problema y salir del programa)