GUIA DE C Y C++

PARA LAS COMPETENCIA ICPC

Jose Gonzalo Espejo Cuba

Este texto está dirigido a todos los programadores que compiten en la ACM-ICPC como una guía para utilizar gran parte de las herramientas que nos brinda los Lenguajes C y C++. Se advierte que para leer este texto se debe tener un conocimiento básico de lo que es programación, algoritmia y estructuras de control.

TIPOS DE DATOS

TIPO	Palabra reservada C++	Palabra reservada C	Tamaño en bits	Rango	
				Min	Max
Lógico	**bool**		8 bits	0, false	1, true
Caractér	**char**	**char**	8 bits	-128	127
Entero short con signo	**short**	**short**	16 bits	-32768	32767
Entero short sin signo	**unsigned short**	**unsigned short**	16 bits	0	65535
Entero int con signo	**int**	**int**	32 bits	-2147483648	2147483647
Entero int sin signo	**unsigned int**	**unsigned int**	32 bits	0	4294967295
Entero long con signo	**long int**	**long int**	32 bits	-2147483648	2147483647
Entero long sin signo	**unsigned**	**unsigned long int**	32 bits	0	4294967295
Entero long long con signo	**long long**	**long long**	64 bits	-9223372036854775808	9223372036854775807
Entero long long sin signo	**unsigned long long**	**unsigned long long**	64 bits	0	18446744073709551615
Real float	**float**	**float**	32 bits	1.17549 e-38	3.40282e+38
Real double	**double**	**double**	64 bits	2.22507e-308	1.79769e+308
Real double grande	**long double**	**long double**	96 bits	3.3621e-4932	1.18973e+4932
Cadena	**string**	***char**			

CONTENIDO

1 ENTRADA Y SALIDA DE DATOS EN C/C++

1.1 ENTRADA Y SALIDA EN C++

1.1.1 ENTRADA Y SALIDA SIMPLE DE DATO

```
#include<iostream>//Librería para la entrada y salida de datos
using namespace std;
int main(){
    //DECLARANDO VARIABLES
    //NUMEROS ENTEROS
    short int num1;
    unsigned short num2;
    int num3;
    unsigned int num4;
    long long num5;
    unsigned long long num6;
    //NUMEROS REALES
    double reald;
    float realf;
    //CADENAS Y CARACTERES
    string cadena;
    char caracter;
    char buffer[100];

    //LECTURA DE DATOS
    cout<<"Introduzca seis numeros enteros"<<endl;//Impresion de mensaje
    cin>>num1>>num2>>num3>>num4>>num5>>num6;//Lectura de seis números enteros
    cout<<"Introduzca dos numeros reales"<<endl;//Impresion de mensaje
    cin>>reald>>realf;//Lectura de datos reales
    cout<<"Introduzca una cadena y un caracter"<<endl;//Impresion de mensaje
    cin>>cadena>>caracter;//Lectura de una cadena y un caractér
    cout<<"Introduzca una cadena"<<endl;//Impresion de mensaje
    cin>>buffer; //Lectura de una cadena

    //MOSTRANDO LOS DATOS POR PANTALLA
    cout<<"El dato entero pequeño introducido es: "<<num1<<endl;
    cout<<"El dato entero pequeño sin signo introducido es: "<<num2<<endl;
    cout<<"El dato entero introducido es: "<<num3<<endl;
    cout<<"El dato entero sin signo introducido es: "<<num4<<endl;
    cout<<"El dato entero grande introducido es: "<<num5<<endl;
    cout<<"El dato entero grande sin signo introducido es:" <<num6<<endl;
    cout<<"El dato real double introducido es: "<<reald<<endl;
    cout<<"El dato real float introducido es: "<<realf<<endl;
    cout<<"La cadena introducida es: "<<cadena<<endl;
    cout<<"El caracter introducido es: "<<caracter<<endl;
    cout<<"La cadena introducida es: "<<buffer<<endl;
    return 0;
}
```

Tipo_dato nombrevar	Declaración de variables, se declara el **Tipo_dato**, y se nombra a la variable.
cin>>nombrevar; **cin>>**var1>>var2>>…	**cin** es el comando para la introducción de datos por teclado, este comando esta seguido por el operador **>>**, a continuación la(s) variable(s) a leerse, es importante notar que esta manera de lectura atrapa un solo dato por variable. Este comando pertenece a la librería <iostream>
cout<<"hola"**<<**var<<…	**cout** es el comando para la impresión de datos por consola, este comando esta seguido por el operador **<<**, a continuación la(s) variable(s). Este comando pertenece a la librería <iostream>
endl	Salto de línea.

1.1.2 ENTRADA Y SALIDA SIMPLE DE CARÁCTER

```
#include<iostream>//Libreria para la entrada y salida de datos
using namespace std;
int main(){
    char c1,c2;
    c1=cin.get();//Lectura del carácter c1
    cin.get(c2);//Lectura del carácter c2
    cout.put(c1);//Mostrando el caracter c1
    cout<<endl; //Salto de linea
    cout.put(c2);//Mostrando el caracter c2
    return 0;
}
```

`c1=cin.get()`	Donde c1 es de tipo char. La función cin.get() lee un solo carácter y la almacena en el carácter c1, si se ha introducido una cadena, esta función leerá únicamente el primer carácter de la cadena.
`cin.get(c2)`	Donde c2 es de tipo char. La función cin.get(char) lee un solo carácter y la almacena en el carácter c2, si se ha introducido una cadena, esta función leerá únicamente el primer carácter de la cadena.
`cout.put(c1)`	Donde c1 es de tipo char. La función cout.put(char) imprime un solo carácter sin salto de línea.

1.1.3 LECTURA DE LINEAS

```
#include<iostream>//Libreria para la entrada y salida de datos
using namespace std;
int main(){
    const int dim=500;
    char linea[dim];
    cin.get(linea,dim); //Lectura de una linea
    cout<<"La linea leida es: "<<linea<<endl;
    return 0;
}
```

```
#include<iostream>//Libreria para la entrada y salida de datos
using namespace std;
int main(){
    const int dim=500;
    char linea[dim];
    cin.getline(linea,dim);//Lectura de una linea(aconsejo este método)
    cout<<"La linea leida es: "<<linea<<endl;
    return 0;
}
```

`cin.get(linea,dim)`	Donde línea es de tipo char[] y dim es un entero positivo. Se realiza la lectura de una línea completa de tamaño dim.
`cin.getline(linea,dim)`	Donde línea es de tipo char[] y dim es un entero positivo. Se realiza la lectura de una línea completa de tamaño dim.

1.1.4 MANEJO DE LA LECTURA DE LINEAS

```
//Dado un numero entero N se lee a continuación N líneas
#include<iostream>//Libreria para la entrada y salida de datos
using namespace std;
int main(){
    const int dim=500;
    char linea[dim];
    int n;
    cout<<"Numero de lineas a leer"<<endl;
    cin>>n;
    for(int i=0;i<=n;i++){
```

```
        cin.getline(linea,dim);
        if(i>0){
            cout<<"La linea "<<i<<" es: "<<linea<<endl;
            cout<<"Numero de caracteres leidos: "<<cin.gcount()<<endl;
        }
    }
    return 0;
}
```

```
//Dado un numero entero N se lee N líneas
#include<iostream>//Libreria para la entrada y salida de datos
using namespace std;
int main(){
    const int dim=500;
    char linea[dim];
    int n;
    cout<<"Numero de lineas a leer"<<endl;
    cin>>n;
    cin.get();//Interrumpimos el flujo de entrada
    for(int i=1;i<=n;i++){
        cin.getline(linea,dim);
        cout<<"La linea "<<i<<" es: "<<linea<<endl;
        cout<<"Numero de caracteres leidos: "<<cin.gcount()<<endl;
    }
    return 0;
}
```

```
//Lectura de una linea seguida de una lectura de dos enteros
//Se imprime la linea leida y la suma de dos enteros
//La entrada termina cuando la línea leída diga "fin"
#include<iostream>//Libreria para la entrada y salida de datos
#include<cstring>//Libreria para la manipulación de cadenas
using namespace std;
int main(){
    const int dim=500;
    char linea[dim];
    int a,b;
    while(cin.getline(linea,dim)){
        if(strcmp(linea,"fin")==0)//Comparación de linea y la cadena "fin"
            break;
        cin>>a>>b;
        cin.get();//Interrumpimos el flujo de entrada
        cout<<linea<<" | "<<a+b<<endl;
    }
    return 0;
}
```

`cin.gcount()`	Esta función devuelve un número entero positivo que representa la longitud de la línea leida.

1.1.5 MANIPULACIONES DE FLUJO

```
#include<iostream>//Libreria para la entrada y salida de datos
using namespace std;
int main(){
    const int dim=20;
    char buffer[dim];
    cin.read(buffer,dim);
    cout.write(buffer,dim);
    return 0;
}
```

El comando **cin.read(buffer,dim)** lee una secuencia de exactamente el número de dim caracteres, en este ejemplo solo leerá 20 caracteres, asi mismo **cin.write(buffer,dim)**imprime una secuencia de 20 caracteres.

1.1.6 ENTRADA DE DATOS HASTA EL FIN DE ARCHIVO

```
while(cin>>dato){
    //codigo
}
```

```
while(cin>>dato1>>dato2>>dato3){
    //codigo
}
```

```
char c;
while((c=cin.get())!='\0'){
    //codigo
}
```

```
char linea[100];
while(cin.getline(linea,100)){
    //codigo
}
```

1.1.7 PRECISION DE COMA FLOTANTE

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<cmath>//Libreria para las funciones matematicas
using namespace std;
int main(){
    double raiz=sqrt(2);//raiz cuadrada de 2
    //redondeando el numero de 1 a 10 decimales
    for(int i=1;i<=10;i++){
        cout.precision(i);
        cout<<raiz<<endl;
    }
    return 0;
}
```

```
//Utilizando la librería iomanip
#include<iostream>//Librería para la entrada y salida de datos
#include<iomanip>//Librería para la manipulación de entrada y salida
#include<cmath>//Librería para las funciones matematicas
using namespace std;
int main(){
    double raiz=sqrt(2);
    for(int i=1;i<=10;i++)
        cout<<setprecision(i)<<raiz<<endl;
    return 0;
}
```

```
Ambos programas imprimen:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
```

1.1.8 CAMPO DE IMPRESIÓN

```
#include<iostream>//Librería para la entrada y salida de datos
using namespace std;
int main(){
    cout<<"Campo de impresion de tamaño 7"<<endl;
    cout.width(7);
    cout<<128<<endl;//Imprime: '    128'
    cout<<"Campo de impresion de tamaño 8"<<endl;
    cout<<"Rellenando los espacios vacios con  el caracter x"<<endl;
    cout.width(8);
    cout.fill('x');
    cout<<128<<endl;//Imprime: 'xxxxx128'
    return 0;
}
```

```
//Campo de impresión utilizando la librería iomanip
#include<iostream>//Librería para la entrada y salida de datos
#include<iomanip>//Librería para la manipulación entrada y salida
using namespace std;
int main(){
    cout<<"campo de impresion de longitud 7"<<endl;
    cout<<setw(7)<<128<<endl;//Imprime:   -1258
    cout<<"\nCampo de impresion de longitud 8"<<endl;
    cout<<"rellenando los espacios vacios con  el caracter x"<<endl;
    cout<<setw(8)<<setfill('x')<<128<<endl;//Imprime: xxxxx-1258
    return 0;
}
```

1.1.9 ALINEACION

```
#include<iostream>
using namespace std;
int main(){
    cout<<"Alineacion a la derecha"<<endl;
    cout.width(7);
    cout<<right<<128<<endl; //Imprime:'    128'
    cout<<"Alineacion a la izquierda"<<endl;
    cout.width(7);
    cout<<left<<128<<endl; //Imprime:'128     '
    cout<<"Alineacion justificada"<<endl;
    cout.width(7);
    cout<<internal<<-128<<endl; //Imprime:'-   128'
    return 0;
}
```

```
//Alineación utilizando flags
#include<iostream>
using namespace std;
int main(){
    cout<<"Alineacion a la derecha"<<endl;
    cout.width(7);
    cout.flags(ios::right);
    cout<<128<<endl;//Imprime:'    128'
    cout<<"Alineacion a la izquierda"<<endl;
    cout.width(7);
    cout.flags(ios::left);
    cout<<128<<endl;//Imprime:'128     '
    cout<<"Alineacion justificada"<<endl;
    cout.width(7);
    cout.flags(ios::internal);
    cout<<-128<<endl;//Imprime:'-   128'
    return 0;
```

```
}
```

```
//Alineación utilizando la librería iomanip
#include<iostream>//Librería para la entrada y salida de datos
#include<iomanip>//Librería para la manipulación entrada y salida
using namespace std;
int main(){
    cout<<"Alineacion a la derecha"<<endl;
    cout<<setw(7);
    cout<<setiosflags(ios::right)<<128<<endl;//Imprime:'    128'
    cout<<"Alineacion a la izquierda"<<endl;
    //quitamos el anterior formato ios::right
    cout<<setw(7)<<resetiosflags(ios::right);
    cout<<setiosflags(ios::left)<<128<<endl;//Imprime:'128    '
    cout<<"Alineacion justificada"<<endl;
    //quitamos el anterior formato ios::left
    cout<<setw(7)<<resetiosflags(ios::left);
    cout<<setiosflags(ios::internal)<<-128<<endl;//Imprime:'-   128'
    return 0;
}
```

1.1.10 BASES NUMERICAS (8, 10 Y 16) EN LA ENTRADA Y SALIDA DE DATOS

```
#include<iostream>//Librería para la entrada y salida de datos
using namespace std;
int main(){
    int num=27;
    cout<<"Base 8:  "<<oct<<num<<endl;//33
    cout<<"Base 10: "<<dec<<num<<endl;//27
    cout<<"Base 16: "<<hex<<num<<endl;//1b
    return 0;
}
```

```
//Utilizando la librería iomanip
#include<iostream>//Librería para la entrada y salida de datos
#include<iomanip>//Librería para la manipulación de entrada y salida
using namespace std;
int main(){
    int num=27;
    cout<<"Base 8:  "<<setbase(8)<<num<<endl;//33
    cout<<"Base 10: "<<setbase(10)<<num<<endl;//27
    cout<<"Base 16: "<<setbase(16)<<num<<endl;//1b
    return 0;
}
```

1.1.11 MANIPULACION DE LA COMA FLOTANTE Y EL SIGNO

```
#include<iostream>//Librería para la entrada y salida de datos
#include<cmath>//Librería para las funciones matemáticas
using namespace std;
int main(){
    //Obtenemos las raices cuadradas de 2 y 9
    double raiz1=sqrt(2);
    double raiz2=sqrt(9);

    cout<<"Default"<<endl;
    cout<<"Raiz cuadrada de 2: "<<raiz1<<endl;//Imprime: 1.41421
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;//Imprime: 3

    cout<<"\nShowpoint"<<endl;
    cout<<"Raiz cuadrada de 9: "<<showpoint<<raiz2<<endl;//Imprime: 3.00000
```

```
    cout<<"Raiz cuadrada de 9: "<<noshowpoint<<raiz2<<endl;//Imprime: 3

    cout<<"\nFixed"<<endl;
    cout<<"Raiz cuadrada de 2: "<<fixed<<raiz1<<endl;//Imprime: 1.414214
    cout<<"Raiz cuadrada de 9: "<<fixed<<raiz2<<endl;//Imprime: 3.000000

    cout<<"\nScientific"<<endl;
    cout<<"Raiz cuadrada de 2: "<<scientific<<raiz1<<endl;//Imprime: 1.414214e+000
    cout<<"Raiz cuadrada de 9: "<<scientific<<raiz2<<endl;//Imprime: 3.000000e+000

    cout<<"\nShowpos"<<endl;
    cout<<"Raiz cuadrada de 9: "<<showpos<<raiz2<<endl;//Imprime: +3.000000e+000
    cout<<"Raiz cuadrada de 9: "<<noshowpos<<raiz2<<endl;//Imprime: 3.000000e+000
    return 0;
}
```

```
//Utilizando flags
#include<iostream>//Librería para la entrada y salida de datos
#include<cmath>//Librería para las funciones matemáticas
using namespace std;
int main(){
    double raiz1=sqrt(2);
    double raiz2=sqrt(9);

    cout<<"Deault"<<endl;
    cout<<"Raiz cuadrada de 2: "<<raiz1<<endl;//Imprime: 1.41421
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;//Imprime: 3

    cout<<"\nShowpoint"<<endl;
    cout.setf(ios::showpoint);
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;//Imprime: 3.00000
    cout.unsetf(ios::showpoint);
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;//Imprime: 3

    cout<<"\nFixed"<<endl;
    cout.flags(ios::fixed);
    cout<<"Raiz cuadrada de 2: "<<raiz1<<endl;//Imprime: 1.41421
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;//Imprime: 3.00000

    cout<<"\nScientific"<<endl;
    cout.flags(ios::scientific);
    cout<<"Raiz cuadrada de 2: "<<raiz1<<endl;//Imprime: 1.414214e+000
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;//Imprime: 3.000000e+000

    cout<<"\nShowpos"<<endl;
    cout.setf(ios::showpos);
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;//Imprime: +3.000000e+000
    cout.unsetf(ios::showpos);
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;//Imprime: 3.000000e+000
    return 0;
}
```

```
//Utilizando la biblioteca iomanip
#include<iostream>//Librería para la entrada y salida de datos
#include<iomanip>//Librería para la manipulación entrada y salida
#include<cmath>//Librería para las funciones matemáticas
using namespace std;
int main(){
    double raiz1=sqrt(2);
    double raiz2=sqrt(9);

    cout<<"Deault"<<endl;
    cout<<"Raiz cuadrada de 2: "<<raiz1<<endl;
    cout<<"Raiz cuadrada de 9: "<<raiz2<<endl;
```

```
    cout<<"\nShowpoint"<<endl;
    cout<<setiosflags(ios::showpoint)<<"Raiz cuadrada de 9: " <<raiz2<<endl;
    cout<<resetiosflags(ios::showpoint)<<"Raiz cuadrada de 9: " <<raiz2<<endl;

    cout<<"\nFixed"<<endl;
    cout<<setiosflags(ios::fixed)<<"Raiz cuadrada de 2: "<<raiz1<<endl;
    cout<<setiosflags(ios::fixed)<<"Raiz cuadrada de 9: "<<raiz2<<endl;

    cout<<"\nScientific"<<endl;
    cout<<setiosflags(ios::scientific)<<"Raiz cuadrada de 2: "<<raiz1<<endl;
    cout<<setiosflags(ios::scientific)<<"Raiz cuadrada de 9: "<<raiz2<<endl;

    cout<<"\nShowpos"<<endl;
    cout<<setiosflags(ios::showpos)<<"Raiz cuadrada de 9: "<<raiz2<<endl;
    cout<<resetiosflags(ios::showpos)<<"Raiz cuadrada de 9: "<<raiz2<<endl;
    return 0;
}
```

1.1.12 MANIPULADORES ALPHA

```
#include<iostream>
using namespace std;
int main(){
    //Manipulacion de valores booleanos
    bool sw=1;
    cout<<sw<<endl; //Imprime: 1
    cout<<boolalpha<<sw<<endl; //Imprime: true
    //Manipulacion de numeros hexadecimales
    int num=2381;
    cout<<hex<<num<<endl;//Imprime: 94d
    cout<<hex<<uppercase<<num<<endl;//Imprime: 94D
    cout<<hex<<nouppercase<<num<<endl;//Imprime: 94d
    return 0;
}
```

1.1.13 MANIPULADORES DE ENTRADA Y SALIDA IOS

`boolalpha`	Cuando se utiliza valores de verdad (true) o falso (false), se imprime textualmente: “true” y “false”. Este manipulador también se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`noboolalpha`	Cuando se utiliza valores de verdad (true) o falso (false), se imprime numéricamente: 1 y 0. Si se ha activado boolalpha, lo desactivamos con noboolalpha.
`fixed`	El dato real mejora su apariencia de la coma decimal. Este manipulador también se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`showpoint`	Se muestra el punto flotante de un dato real. Este manipulador también se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`noshowpoint`	Se oculta el punto flotante de un dato real. Este manipulador también desactiva showpoint.
`showpos`	Se muestra el signo positivo de un dato numérico. Este manipulador tambien se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`noshowpos`	Se oculta el signo positivo de un dato numérico. Este manipulador también desactiva showpos.
`right`	Alineción a la derecha. También se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`left`	Alineación a la izquierda. También se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`internal`	Alineación justificada. También se puede utilizar con las funciones cout.flags(…) y cout.setf(…).

`oct`	Establece el número a base 8. También se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`dec`	Establece el número a base 10. También se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`hex`	Establece el número a base 16. También se puede utilizar con las funciones cout.flags(…) y cout.setf(…).
`uppercase`	Imprime los caracteres alfabéticos como mayúsculas.
`nouppercase`	Imprime los caracteres alfabéticos como minúsculas. Si el manipulador uppercase está activado, entonces nouppercase lo desactivara.

1.1.14 FUNCIONES DEL COMANDO COUT

`width(n)`	Donde n es un entero positivo, la función width reserva n espacios para la impresión de un dato, por defecto el dato sera impreso alineado a la izquierda.
`fill(c)`	Donde c es un carácter, la función fill rellena los espacios de impresión con el carácter c que no son utilizados en la impresión de un dato.
`precision(n)`	Donde n es un entero positivo, la función precision redondea a n decimales.
`flags(…)`	Establece u obtiene el formato de flags. Sus parámetros son manipuladores **ios: flags(ios::manip1 | ios::manip2 | ios::manip3 | …)**
`setf(…)`	Establece el formato de flags. Sus parámetros son manipuladores **ios: setf(ios::manip1 | ios::manip2 | ios::manip3 | …)**
`unsetf(…)`	Limpia el formato de flags. Sus parámetros son manipuladores **ios: unseft(ios::manip1 | ios::manip2 | ios::manip3 | …)**

1.1.15 FUNCIONES DE LA LIBRERÍA IOMANIP

`setw(n)`	Donde n es un entero positivo, la función setw reserva n espacios para la impresión de un dato, por defecto el dato será impreso alineado a la izquierda.
`setfill(c)`	Donde c es un carácter, la función setfill rellena los espacios de impresión con el carácter c que no son utilizados en la impresión de un dato.
`setprecision(n)`	Donde n es un entero positivo, la función set precisión redondea a n decimales.
`setbase(n)`	Donde n es un entero positive, establece la base numérica n, pero tan solo se permiten las bases 8, 10 y 16.
`setiosflags(…)`	Establece u obtiene el formato de flags. Sus parámetros son manipuladores **ios: setiosflags(ios::manip1 | ios::manip2 | …)**
`resetiosflags(…)`	Resetea el formato de flags. Sus parámetros son manipuladores **ios: resetiosflags(ios::manip1 | ios::manip2 | …)**

1.2 ENTRADA Y SALIDA EN C

1.2.1 ENTRADA Y SALIDA SIMPLE DE DATO

En el lenguaje C la librería para leer datos e imprimirlos es <stdio.h> mientras que en el lenguaje de C++ la librería puede ser: <stdio.h> o <cstdio>.

```
#include<stdio.h>//Librería para la entrada y salida de datos
int main(){
    //DECLARANDO VARIABLES
    //NUMEROS ENTEROS
    int num1;
    unsigned int num2;
    long long num3;
```

```
    unsigned long long num4;
    //NUMEROS REALES
    double reald;
    float realf;
    //CADENAS Y CARACTERES
    char caracter;
    char cadena[20];

    //LECTURA DE DATOS
    puts("Introduzca cuatro numeros enteros");//Impresión de mensaje
    scanf("%i%u%lld%llu",&num1,&num2,&num3,&num4);//Lectura de cuatro enteros
    puts("Introduzca dos numeros reales");//Impresión de mensaje
    scanf("%lf %f",&reald,&realf);//Lectura de números reales
    puts("Introduzca una cadena y un caracter");//Impresion de mensaje
    scanf("%s %c",cadena,&caracter);//Lectura de una cadena y un caractér

    //MOSTRANDO LOS DATOS POR PANTALLA
    printf("El dato entero introducido es: %d\n",num1);
    printf("El dato entero sin signo introducido es: %u\n",num2);
    printf("El dato entero grande introducido es: %lld\n",num3);
    printf("El dato entero grande sin signo introducido es: %llu\n",num4);
    printf("El dato real double introducido es: %lf\n",reald);
    printf("El dato real float introducido es: %f\n",realf);
    printf("La cadena introducida es: %s\n",cadena);
    printf("El caracter introducido es: %c\n",caracter);
    return 0;
}
```

Tipo_dato nombrevar	Declaración de variables, se declara el **Tipo_dato**, y se nombra a la variable.
scanf("%especificador", nombrevar); **scanf("%especificador1%especificador2"**, **&**var1,&var2)	**scanf** es el comando para la introducción de datos por teclado, este comando comprende especificadores para cada tipo de dato, seguida por la variable a leerse, es importante notar que esta manera de lectura atrapa un solo dato por variable. Este comando pertenece a la librería <stdio.h> en C. En C++ es lo mismo utilizar <stdio.h> o <cstdio>
printf("hola %espe", nombrevar)	**printf** es el comando para la impresión de datos por consola, este comando comprende especificadores para cada tipo de datos, también se pueden imprimir mensajes, a continuación la variable o las variables a imprimirse o bien uno o más datos. Este comando pertenece a la librería <stdio.h> en C. En C++ es lo mismo utilizar <stdio.h> o <cstdio>
puts("hola"**)**	Imprime directamente un mensaje o una cadena y automáticamente se da el salto de línea.
printf("\n")	Salto de línea.

1.2.2 ESPECIFICADORES DE ENTRADA Y SALIDA

ESPECIFICADOR	TIPO	DESCRIPCION
%c	char	Carácter.
%s	Cadena	Cadena.
%i	int	Entero.
%d	int	Entero en base 10 (es lo mismo que %i).
%o	int	Entero en base 8.
%x	int	Entero en base 16.

%u	unsigned int	Entero sin signo.
%f	float	Número con punto flotante en notación decimal.
%e	float	Número con punto flotante en notación científica.
%g	float	Puede ser %e ó %f, lo que decida printf.
%lf	double	Número con punto flotante en notación decimal.
%ld	long int	Entero.
%lu	unsigned long int	Entero sin signo.
%lld	long long int	Entero.
%llu	unsigned long long int	Entero sin signo.

1.2.3 ENTRADA Y SALIDA SIMPLE DE CARÁCTER

```
#include<stdio.h>//Librería para la entrada y salida de datos
int main(){
    char c;
    c=getchar();//Lectura del carácter c
    putchar(c);//Mostrando el carácter c
    return 0;
}
```

c = getchar()	Donde c es de tipo char. La función getchar lee un solo carácter y la almacena en la variable c, si se ha introducido una cadena, esta función leerá únicamente el primer carácter de la cadena.
putchar(c)	Donde c es de tipo char. La función putchar imprime un solo carácter sin salto de línea.

1.2.4 LECTURA DE LINEAS

```
#include<stdio.h>//Librería para la entrada y salida de datos
int main(){
    const int dim=500;
    char linea[dim];
    gets(linea);//Lectura de la linea
    printf("La linea leida es: ");
    puts(linea);//Impresión de la linea
    return 0;
}
```

gets(x)	Donde x es de tipo char[]. Se realiza la lectura de una línea completa.

1.2.5 MANEJO DE LA LECTURA DE LINEAS

```
//Dado un numero entero N se lee N lineas que contienen 3 numeros
#include<stdio.h>//Librería para la entrada y salida de datos
int main(){
    const int dim=500;
    char linea[dim];
    int n,i,a,b,c;
    puts("Numero de lineas a leer");
       scanf("%d",&n);
    for(i=0;i<=n;i++){
        gets(linea);
        if(i>0)
            printf("La linea %d es: %s\n",i,linea);
```

```
    }
    return 0;
}
```

```
//Dado un numero entero N se lee N lineas que contienen 3 numeros
#include<stdio.h>//Librería para la entrada y salida de datos
int main(){
    const int dim=500;
    char linea[dim];
    int n,i,a,b,c;
    puts("Numero de lineas a leer");
    scanf("%d\n",&n);
    for(i=1;i<=n;i++){
        gets(linea);
        printf("La linea %d es: %s\n",i,linea);
    }
    return 0;
}
```

```
//Dado un numero entero N se lee N líneas
#include<stdio.h>//Librería para la entrada y salida de datos
int main(){
    const int dim=500;
    char linea[dim];
    int n,i,a,b,c;
    puts("Numero de lineas a leer");
    scanf("%d",&n);
    //Interrumpimos el flujo de entrada
    getchar();
    for(i=1;i<=n;i++){
        gets(linea);
        printf("La linea %d es: %s\n",i,linea);
    }
    return 0;
}
```

```
//Lectura de una línea seguida de una lectura de dos enteros
//Se imprime la linea leida y la suma de dos enteros
//La entrada termina cuando la linea leida diga "fin"
#include<stdio.h>//Librería para la entrada y salida de datos
#include<string.h>//Librería para el manejo de cadenas
int main(){
    const int dim=500;
    char linea[dim];
    int n,i,a,b,c;
    while(strcmp(gets(linea),"fin")!=0){
        scanf("%d%d",&a,&b);
        getchar();
        printf("La linea leida es: %s\n",linea);
        printf("La suma es: %d\n",a+b);
    }
    return 0;
}
```

1.2.6 FUNCIONES SSCANF Y SPRINTF

```
#include<stdio.h>
int main(){
    char cadena1[]="12 56 34";
    char cadena2[500];
    int a,b,c,n;
    //FUNCION SSCANF
    sscanf(cadena1,"%d %d %d",&a,&b,&c);
```

```
    printf("La suma es: %d\n",a+b+c);//Imprime: La suma es: 102
    //FUNCION SPRINTF
    n=sprintf(cadena2,"%d + %d + %d = %d",a,b,c,a+b+c);
    printf("la cadena %s tiene una longitud de %d\n",cadena2,n);
    //Imprime: la cadena 12 + 56 + 34 = 102 tiene una longitud de 18
    return 0;
}
```

La función sscanf realiza una lectura de datos desde una secuencia de caracteres, mientras que la función sprintf almacena en una secuencia de caracteres una impresión.

1.2.7 LECTURA HASTA EL FIN DE ARCHIVO

```
int dato;
while(scanf("%d",&dato)==1){
    //codigo
}
```

```
int dato1, dato2, dato3;
while(scanf("%d%d%d",&dato1,&dato2,&dato3)==3){
    //codigo
}
```

```
char c;
while((c=getchar())!='\0'){
    //codigo
}
```

```
char linea[100];
while(gets(linea)){
    //codigo
}
```

1.2.8 PRECISION DE COMA FLOTANTE

```
#include<stdio.h>//Librería para la entrada y salida de datos
#include<math.h>//Librería para las funciones matemáticas
int main(){
    double raiz=sqrt(2);
    //redondeando el número a 4 decimales
    printf("%.4f\n",raiz);//Imprime: 1.4142
    //redondeando el número a 7 decimales
    printf("%.7f",raiz);//Imprime: 1.4142136
    return 0;
}
```

1.2.9 BASES NUMERICAS (8, 10 Y 16) EN LA ENTRADA Y SALIDA DE DATOS

```
#include<stdio.h>//Librería para la entrada y salida de datos
int main(){
    int num=27;
    printf("Base 8: %o\n",num);//33
    printf("Base 8: %#o\n",num);//033
    printf("Base 10: %d\n",num);//27
    printf("Base 16: %x\n",num);//1b
    printf("Base 16: %X\n",num);//1B
    printf("Base 16: %#X\n",num);//0X1B
    return 0;
}
```

1.2.10 MANIPULACION DE LA COMA FLOTANTE

```
#include<stdio.h>//Librería para la entrada y salida de datos
#include<math.h>//Librería para las funciones matemáticas
int main(){
    float num=sqrt(2);
    printf("Float: %f\n",num);
    printf("Scientific: %e\n",num);
    printf("G: %g\n",num);
    return 0;
}
```

1.2.11 CAMPO DE IMPRESIÓN Y ALINEACION

```
#include<stdio.h>
int main(){
       int n=128;
       puts("Alineacion a la derecha");
       puts("Campo de impresion de tamaño 7");
       printf("%7d\n",n);//Imprime: '    128'

       puts("\nAlineacion a la izquierda");
       puts("Campo de impresion de tamaño 7");
       printf("%-7d\n",n);//Imprime: '128   '

       puts("\nColocando ceros a la izquierda de un numero entero");
       puts("Campo de impresion de tamaño 7");
       printf("%07d\n",n);//Imprime: '0000128'

       puts("\nUn espacio en blanco antes de imprimir un entero");
       printf("% d\n",n);//Imprime: ' 128'

       puts("\nMostrando el signo positivo");
       printf("%+d\n",n);//Imprime: '+128'
       return 0;
}
```

1.3 LECTURA Y ESCRITURA DE ARCHIVOS

1.3.1 LECTURA Y ESCRITURA DE ARCHIVOS EN C++

```
#include<iostream>//Librería para la entrada y salida de datos
#include<fstream>//Librería para el manejo de archivos
#include<cmath>//Librería para las funciones matemáticas
using namespace std;
int main(){
    ifstream fin("entrada_c++.txt");//Archivo de entrada
    ofstream fout("salida_c++.txt");//Archivo de entrada
    if(fin!=NULL){ //Preguntamos si el archivo existe
        int n,a,b;
        fin>>n;
        while(n--){
            fin>>a>>b;
            fout<<"Suma: "<<a+b<<endl;
            fout<<"Diferencia absoluta: "<<fabs(a-b)<<endl;
            fout<<"Producto: "<<a*b<<endl;
            if(b!=0)
                fout<<"Division: "<<a/(double)b<<endl;
            else
                fout<<"Error! division entre 0"<<endl;
            fout<<'\n';
```

```
        }
    }
    else
        cout<<"Error, el archivo de lectura no existe"<<endl;
    return 0;
}
```

Para la lectura y escritura de archivos es necesario utilizar la librería <fstream>, para la lectura utilizamos el objeto ifstream, para la escritura el objeto ofstream.

1.3.2 LECTURA Y ESCRITURA DE ARCHIVOS EN C

```
#include<cstdio>//Librería para la entrada y salida de datos
#include<cmath>//Librería para las funciones matematicas
#include<iostream>//Librería para la entrada y salida de datos
using namespace std;
int main(){
    //Abriendo el archivo entrada_c++.txt
    freopen("entrada_c++.txt","r",stdin);
    //Abriendo el archivo entrada_c.txt
    freopen("salida_c.txt","w",stdout);
    int n,a,b;
    scanf("%i",&n);
    while(n--){
        cin>>a>>b;
        cout<<"Suma: "<<a+b<<endl;
        cout<<"Diferencia absoluta: "<<fabs(a-b)<<endl;
        printf("Producto: %i\n",a*b);
        if(b!=0)
            printf("Division: %f\n",a/(double)b);
        else
            cout<<"Error! division entre 0"<<endl;
        cout<<'\n';
    }
    return 0;
}
```

1.4 ADICIONALES DE ENTRADA

1.4.1 OPTIMIZACION DE IOSTREAM

```
#include<iostream>
using namespace std;
int main(){
    ios_base::sync_with_stdio(false);
    int num;
    cin>>num;
    cout<<"Numero: "<<num<<endl;
    return 0;
}
```

1.4.2 MAS DE SCANF

`scanf("%[^\n]\n",cad)`	Donde cad es char**[].** Lee hasta fin de linea y descarta \n
`scanf("%10s",cad)`	Donde cad es char**[].** Lee 10 caracteres, si se introducen mas, entonces solo se toma en cuenta los 10 primeros.
`scanf("%[abc]\n",cad)`	Donde cad es char**[].** Lee caracteres 'a', 'b' y 'c', la entrada finaliza cuando se introduce un character diferente.

2 FUNCIONES MATEMATICAS

2.1 FUNCIONES MATEMATICAS CON LA LIBRERÍA cmath (math.h en C)

2.1.1 FUNCIONES TRIGONOMETRICAS

```
#include<iostream>//Librería para la entrada y salida de datos
#include<cmath>//Librería para las funciones matemáticas
#define PI 3.14159265
using namespace std;
int main(){
    double num=45;
    cout.setf(ios::fixed);
    cout<<"FUNCION COSENO"<<endl;
    cout<<"coseno en grados: "<<cos(num*PI/180.0)<<endl;//0.707107
    cout<<"coseno en radianes: "<<cos(num)<<endl;//0.525322

    cout<<"\nFUNCION SENO"<<endl;
    cout<<"seno en grados: "<<sin(num*PI/180.0)<<endl;//0.707107
    cout<<"seno en radianes: "<<sin(num)<<endl;//0.850904

    cout<<"\nFUNCION TANGENTE"<<endl;
    cout<<"tangente en grados: "<<tan(num*PI/180.0)<<endl;//1.000000
    cout<<"tangente en radianes: "<<tan(num)<<endl;//1.619775

    num=0.1224;
    cout<<"\nFUNCION ARCO COSENO"<<endl;
    cout<<"arco-coseno en grados: "<<acos(num)*180/PI<<endl;//82.969366
    cout<<"arco-coseno en radianes: "<<acos(num)<<endl;//1.448089

    cout<<"\nFUNCION ARCO SENO"<<endl;
    cout<<"arco-seno en grados: "<<asin(num)*180/PI<<endl;//7.030634
    cout<<"arco-seno en radianes: "<<asin(num)<<endl;//0.122708

    cout<<"\nFUNCION ARCO TANGENTE"<<endl;
    cout<<"arco-tangente en grados: "<<atan(num)*180/PI<<endl;//6.978293
    cout<<"arco-tangente en radianes: "<<atan(num)<<endl;//0.121794
    return 0;
}
```

2.1.2 FUNCIONES HIPERBOLICAS

```
#include<iostream>//Librería para la entrada y salida de datos
#include<cmath>//Librería para las funciones matemáticas
using namespace std;
int main(){
    double num=0.45;
    cout.setf(ios::fixed);
    cout<<"Coseno Hiperbolico: "<<cosh(num)<<endl;//1.102970
    cout<<"Seno Hiperbolico: "<<sinh(num)<<endl;//0.465342
    cout<<"Tangente Hiperbolico: "<<tanh(num)<<endl;//0.421899
    return 0;
}
```

2.1.3 FUNCIONES EXPONENCIALES Y LOGARITMICAS

```
#include<iostream>//Librería para la entrada y salida de datos
#include<cmath>//Librería para las funciones matemáticas
using namespace std;
int main(){
    double num=8.0,m;
```

```
    int n;
    cout<<"FUNCION EXP"<<endl;
    cout<<"La exponencial: "<<exp(num)<<endl;//2980.96

    cout<<"\nFUNCION FREXP"<<endl;
    double a=frexp(num,&n);
    cout<<num<<" = "<<a<<" * 2^"<<n<<endl;//8 = 0.5 * 2^4

    cout<<"\nFUNCION LDEXP"<<endl;
    a=ldexp(num,n);
    cout<<num<<" * 2^"<<n<<" = "<<a<<endl;//8 * 2^4 = 128

    cout<<"\nFUNCION LOG"<<endl;
    cout<<"El logaritmo natural: "<<log(num)<<endl;//2.07944

    cout<<"\nFUNCION LOG10"<<endl;
    cout<<"El logaritmo en base 10: "<<log10(num)<<endl;//0.90309

    cout<<"\nFUNCION MODF"<<endl;
    a=modf(log(num),&m);
    cout<<log(num)<<" = "<<m<<" + "<<a<<endl;//2.07944 = 2 + 0.0794415
    return 0;
}
```

2.1.4 FUNCIONES DE POTENCIA

```
#include<iostream>//Librería para la entrada y salida de datos
#include<cmath>//Librería para las funciones matematicas
using namespace std;
int main(){
    cout<<"3^4 = "<<pow(3,4)<<endl; //Imprime: 3^4 = 81
    cout<<"Raiz cuadrada de 9 = "<<sqrt(9)<<endl;//Imprime: Raiz cuadrada de 9 = 3
    cout<<"Raiz cuarta de  3 = "<<pow(3,1/(double)4)<<endl;//Imprime:  3 = 1.31607
    return 0;
}
```

2.1.5 FUNCIONES DE REDONDEO

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
    double num=12/5.0;
    cout<<"El dato original: "<<num<<endl;//2.4
    cout<<"Redondeando al  primer numero entero mas grande: "<<ceil(num)<<endl;//3
    cout<<"Redondeando al  primer numero entero mas pequeño: "<<floor(num)<<endl;//2
    cout<<"Modulo decimal"<<endl;
    cout<<num<<" % 2 = "<<fmod(num,2)<<endl;//2.4 % 2 = 0.4
    cout<<num<<" % 3 = "<<fmod(num,3)<<endl;//2.4 % 3 = 2.4
    cout<<num<<" % 1.1 = "<<fmod(num,1.1)<<endl;//2.4 % 1.1 = 0.2
    return 0;
}
```

2.1.6 VALOR ABSOLUTO

```
#include<iostream>//Librería para la entrada y salida de datos
#include<cmath>//Librería para las funciones matematicas
using namespace std;
```

```
int main(){
    int a=7,b=12;
    cout<<"FUNCION FABS"<<endl;
    cout<<"Valor absoluto: "<<fabs(a-b)<<endl;//5

    cout<<"\nFUNCION ABS"<<endl;
    cout<<"Valor absoluto: "<<abs((double)(a-b))<<endl;//5
    return 0;
}
```

2.2 FUNCIONES MATEMATICAS CON LA LIBRERÍA <cstdlib> (<stdlib.h> en C)

2.2.1 NUMEROS ALEATORIOS

```
#include<iostream>//Librería para la entrada y salida de datos
#include<cstdio>//Librería para la entrada y salida de datos
#include<cstdlib>//Librería para generar números aleatorios
using namespace std;
int main(){
    int num;
    puts("generando 100 numeros aleatorios del 1 al 10");
    for(int i=0;i<100;i++){
        num=rand()%10+1;
        cout<<i<<": "<<num<<endl;
    }
    puts("generando 100 numeros aleatorios del 1 al 100");
    for(int i=0;i<100;i++){
        num=rand()%100+1;
        cout<<i<<": "<<num<<endl;
    }
    return 0;
}
```

2.2.2 FUNCIONES ARITMETICAS CON NUMEROS ENTEROS

```
#include<iostream>>//Librería para la entrada y salida de datos
#include<cstdlib>//Librería para operar números enteros
using namespace std;
int main(){
    int a=7,b=12;
    cout<<"Valor absoluto: "<<abs(a-b)<<endl;//5
    cout<<"Division entera: "<<div(a,b).quot<<endl;//0
    cout<<"Residuo de la division: "<<div(a,b).rem<<endl;//7
    return 0;
}
```

3 CADENAS Y CARACTERES EN C/C++

3.1 FUNCIONES BASICAS DE CADENAS EN C++ LIBRERÍA <string>

3.1.1 CONSTRUCTORES DE CADENAS

```
#include<iostream>
using namespace std;
int main(){
    //crea una cadena1 vacia
    string cadena1;
    //crea una cadena2 con un contenido
    string cadena2("Esta es mi primera cadena");
    //crea una cadena3 con el contenido de la cadena2
    string cadena3(cadena2);
    //crea una cadena4 con 9 caracteres de la cadena2 desde el carácter 3
    string cadena4(cadena2,3,9);
    //crea una cadena5 con todos los caracteres de la cadena2
    string cadena5(cadena2,11);
    //crea una cadena6 de 10 caracteres 'h'
    string cadena6(10,'h');
    //crea una cadena7 de 10 caracteres 'z' (ASCII)
    string cadena7(10,122);
    //crea una cadena8 con el contenido de la cadena2
    //a partir del primer caracter hasta el caracter 7
    string cadena8(cadena2.begin(),cadena2.begin()+7);
    //crea una cadena9 con el contenido de la cadena2
    //a partir del tercer caracter hasta el caracter 12
    string cadena9(cadena2.begin()+3,cadena2.begin()+12);
    //crea una cadena10 con el contenido de la cadena2
    string cadena10(cadena2.begin(),cadena2.end());
    //crea una cadena11 con el contenido de la cadena6
    string cadena11=cadena6;

    cout<<"cadena1: "<<cadena1<<endl;//Imprime:
    cout<<"cadena2: "<<cadena2<<endl;//Imprime: Esta es mi primera cadena
    cout<<"cadena3: "<<cadena3<<endl;//Imprime: Esta es mi primera cadena
    cout<<"cadena4: "<<cadena4<<endl;//Imprime: a es mi p
    cout<<"cadena5: "<<cadena5<<endl;//Imprime: primera cadena
    cout<<"cadena6: "<<cadena6<<endl;//Imprime: hhhhhhhhhh
    cout<<"cadena7: "<<cadena7<<endl;//Imprime: zzzzzzzzzz
    cout<<"cadena8: "<<cadena8<<endl;//Imprime: Esta es
    cout<<"cadena9: "<<cadena9<<endl;//Imprime: a es mi p
    cout<<"cadena10: "<<cadena10<<endl;//Imprime: Esta es mi primera cadena
    cout<<"cadena11: "<<cadena11<<endl;//Imprime: hhhhhhhhhh
    return 0;
}
```

`string w`	Declarando el tipo de dato string, crea una cadena vacia w.
`string x(w)`	Donde x y w son de tipo string, crea una cadena x con el contenido de la cadena w.
`string x(w,i,n)`	Donde x y w son de tipo string, i y n son enteros no negativos, toma n caracteres desde el i-esimo carácter de la cadena w, y crea la cadena x.
`string x(w,i)`	Donde w es de tipo string, i es un entero positivo, crea una cadena x con el contenido de la cadena w a partir del i-esimo carácter hasta el final.
`string w(n,c)`	Donde n es un entero positivo y c es de tipo carácter. Crea una cadena w con n copias del carácter c.
`string A(itB1,itB2)`	Donde A y B son de tipo string, itB1 e itB2 son iteradores de la cadena B. Toma caracteres de B desde la posición de itB1 hasta la posición itB2, y crea la cadena A con los caracteres de B.

3.1.2 ASIGNACION DE CADENAS

```
#include<iostream>//Librería para la entrada y salida de datos
#include<string>//Librería para las funciones de cadena
using namespace std;
int main(){
    string cadena1,cadena2,cadena3,cadena4,cadena5;
    cout<<"OPERADOR DE ASIGNACION ="<<endl;
    cadena1="Hola";//Asigna "Hola" a la cadena1
    cout<<"cadena1: "<<cadena1<<endl;//Imprime: Hola

    cout<<"\nFUNCION ASSIGN"<<endl;
    cadena2.assign("Universidad");//Asigna "Universidad" a la cadena2
    cout<<"cadena2: "<<cadena2<<endl;//Imprime: Universidad
    cadena3.assign(cadena2); //Asignando el valor de la cadena2 a la cadena3
    cout<<"cadena3: "<<cadena3<<endl;//Imprime: Universidad
    //Asignando el contenido de la cadena2 a la cadena4
    //A partir del carácter 2 tomamos 5 caracteres de la cadena2
    cadena4.assign(cadena2,2,5);
    cout<<"cadena4: "<<cadena4<<endl;//Imprime: ivers
    //Asignando el contenido de la cadena2 a la cadena5
    //A partir del carácter 2 tomamos caracteres hasta el caracter 10 de la cadena5
    cadena5.assign(cadena2.begin()+2,cadena2.begin()+10);
    cout<<"cadena5: "<<cadena5<<endl;//Imprime: iversida

    cout<<"\nFUNCION SWAP"<<endl;
    cout<<"cadena1: "<<cadena1<<endl;//Imprime: Hola
    cout<<"cadena3: "<<cadena3<<endl;//Imprime: Universidad
    cout<<"Intercambio de valores"<<endl;
    cadena1.swap(cadena3);
    cout<<"cadena1: "<<cadena1<<endl;//Imprime: Universidad
    cout<<"cadena3: "<<cadena3<<endl;//Imprime: Hola
    return 0;
}
```

`x=w`	Donde x y w son de tipo string, asigna el valor de la cadena w a la cadena x.
`x.assign(y)`	Donde x y w son de tipo string, asigna el valor de la cadena w a la cadena x.
`x.assign(w,i,j)`	Donde x y w son de tipo string, i y j son enteros no negativos, toma j caracteres a partir de la posición del i-esimo carácter de la cadena w y los asigna a la cadena x.
`A.assing(itB1,itB2)`	Donde A y B son de tipo string, itB1 e itB2 son iteradores de B. Toma los caracteres de B a partir de la posición itB1 hasta la posición itB2 y los asigna a la cadena A.
`x.swap(w)`	Donde x y w son de tipo string, se intercambia el valor de la cadena x con el valor de la cadena y.

3.1.3 CONCATENACION DE CADENAS

```
#include<iostream>//Librería para la entrada y salida de datos
#include<string>//Librería para las funciones de cadena
using namespace std;
int main(){
    string cadena1,cadena2,cadena3,cadena4,cadena5;
    cadena1="Hola";
    cadena2="Universidad";
    cadena3="Buenos dias";

    cout<<"\nOPERADOR DE CONCATENACION +"<<endl;
    cadena1=cadena1+" Mundo";//concatenando con otra cadena
    cadena1=cadena1+'!';//concatenando con un carácter
```

```
    cout<<"cadena1: "<<cadena1<<endl;//Imprime: Hola Mundo!

    cout<<"\nFUNCION APPEND"<<endl;
    cadena2.append(" Mayor de San Andres");//concatenando un valor a la cadena2
    cout<<"cadena2: "<<cadena2<<endl;//Imprime: Universidad Mayor de San Andres
    //Concatenando la cadena1 a la cadena3
    //A partir del carácter 1 tomamos 5 caracteres de la cadena1
    cadena3.append(cadena1,1,5);
    cout<<"cadena3: "<<cadena3<<endl;//Imprime: Buenos diasola M
    //Concatenando la cadena1 a la cadena4 a partir del carácter 1 hasta el carácter 9
    cadena4.append(cadena1.begin()+1,cadena1.begin()+9);
    cout<<"cadena4: "<<cadena4<<endl;//Imprime: ola Mund

    cout<<"\nFUNCION PUSH_BACK"<<endl;
    cout<<"cadena5: "<<cadena5<<endl;//No muestra nada
    cadena5.push_back('A');
    cout<<"cadena5: "<<cadena5<<endl;//Imprime: A
    cadena5.push_back('C');
    cout<<"cadena5: "<<cadena5<<endl;// Imprime: AC
    cadena5.push_back('M');
    cout<<"cadena5: "<<cadena5<<endl;// Imprime: ACM
    return 0;
}
```

x+y	Donde x,y son de tipo string, concatenación de las cadenas x, y.
x+c	Donde x es de tipo string y c es de tipo char, se concatena la cadena x con el carácter c.
x.append(y)	Donde x,y son de tipo string, se concatena el valor de la cadena y a la cadena x.
x.append(y,i,n)	Donde x,y son de tipo string, i,n son enteros positivos, se concatena el valor de la cadena x con n caracteres de la cadena y a partir de la posición del i-esimo carácter.
A.append(itB1,itB2)	Donde A y B son de tipo string, itB1 e itB2 son iteradores de B. Toma los caracteres de B a partir de la posición itB1 hasta la posición itB2 y los concatena a la cadena A.
x.push_back(c)	Donde x es de tipo string y c es de tipo char, esta función concatena el carácter c al final de la cadena x.

3.1.4 INSERCCION Y ELIMINACION DE CARACTERES DE UNA CADENA

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string cadena1="El importante";
    string cadena2=" estudio";
    string cadena3=" es muy";

    cout<<"FUNCION INSERT"<<endl;
    //Insertando la cadena2 en la posición 2 de la cadena1
    cadena1.insert(2,cadena2);
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El estudio importante
    //Insertando 2 caracteres desde la posición 1 de la cadena3
    //En la posición 10 de la cadena1
    cadena1.insert(10,cadena3,1,2);
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El estudioes importante
    //Insertando 1 copia del carácter ' ' en la posición 10 de la cadena1
    cadena1.insert(10,1,' ');
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El estudio es importante
    //Insertando 3 copias del carácter '.' en la posición final de la cadena1
    cadena1.insert(cadena1.end(),3,'.');
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El estudio es importante...
```

```
    //Insertando caracteres de la cadena3 desde la posición 3 hasta el final
    //En la posición 13 de la cadena1
    cadena1.insert(cadena1.begin()+13,cadena3.begin()+3,cadena3.end());
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El estudio es muy importante...

    cout<<"\nFUNCION ERASE"<<endl;
    //Borrando los caracteres de la cadena1 a partir de la posición 29
    cadena1.erase(29);
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El estudio es muy importante.
    //Borrando 3 caracteres de la cadena1 a partir de la posición 13
    cadena1.erase(13,3);
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El estudio esy importante.
    //Borrando el caracter de la posición 13 de la cadena1
    cadena1.erase(cadena1.begin()+13);
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El estudio es importante.
    //Borrando caracteres de la cadena1 a partir de la posición 3 hasta la posición 11
    cadena1.erase(cadena1.begin()+3,cadena1.begin()+11);
    cout<<"cadena1: "<<cadena1<<endl;//Muestra: El es importante.

    cout<<"\nFUNCION CLEAR"<<endl;
    //Borrando todos los caracteres de la cadena1
    cadena1.clear();
    cout<<"cadena1: "<<cadena1<<endl;//Muestra:
    return 0;
}
```

`cad1.insert(i,cad2)`	Donde cad1 y cad2 son de tipo string, i es un entero no negativo. Introduce la cadena cad2 en la i-esima posición de la cadena cad1.
`cad1.insert(j,cad2,i,n)`	Donde cad1 y cad2 son de tipo string, i, j y n son enteros no negativos. Introduce n caracteres desde el carácter i de la cadena cad2 en la j-esima posición de la cadena cad1.
`cad.insert(i,n,c)`	Donde cad es de tipo string, i,n son enteros no negativos y c es de tipo char. Introduce n copias del carácter c en la i-esima posición de la cadena cad.
`cad.insert(it,c)`	Donde cad es de tipo string, it es un iterador de cad y c es de tipo char, introduce el carácter c en la posición de it.
`A.insert(itA, itB1, itB2)`	Donde A y B son de tipo string, itA es un iterador de A, itB1 e itB2 son iteradores de B. Toma los caracteres de la cadena B a partir de la posición de itB1, hasta la posición itB2 y los introduce en la posición de itA de la A.
`cad.erase(i)`	Donde cad es de tipo string, i es un entero no negativo, borra caracteres de la cadena cad, desde la posición i-esima hasta el final.
`cad.erase(i,n)`	Donde cad es de tipo string, i y n son enteros no negativos, borra n caracteres de la cadena cad, desde la posición i-esima.
`cad.erase(it)`	Donde cad es de tipo string, it es un iterador de cad, borra el carácter que se encuentra en la posición it.
`cad.erase(it1,it2)`	Donde cad es de tipo string, it1 e it2 son iteradores de cad, borra los caracteres de la cadena cad, desde la posición de it1 hasta la posición it2.
`cad.clear()`	Donde cad es de tipo string, la función clear borra todos los caracteres de la cadena.

3.1.5 REEMPLAZO DE CADENAS

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string cad1,cad2,cad3,cad4,cad5;
```

```
    cad1="El estudio es muy importante.";
    cad2="util";
    cad3="estudiar mucho";
    cad4="super";
    cad5="demasiado";

    cout<<"FUNCION REPLACE"<<endl;
    //Reemplazando 10 caracteres desde la posicion 18 de la cad1 por la cadena cad2
    cad1.replace(18,10,cad2);
    cout<<"cad1: "<<cad1<<endl;//Muestra: El estudio es muy util.
    //Reemplazando 7 caracteres desde la posicion 3 de la cad1
    //Por 8 caracteres desde la posicion 0 de la cad3
    cad1.replace(3,7,cad3,0,8);
    cout<<"cad1: "<<cad1<<endl;//Muestra: El estudiar es muy util.
    //Reemplazando 3 caracteres desde la posicion 3 de la cad1
    //Por 2 copias del caracter '>'
    cad1.replace(15,3,2,'>');
    cout<<"cad1: "<<cad1<<endl;//Muestra: El estudiar es >> util.

    //UTILIZANDO ITERADORES
    //Reemplazando los caracteres de la cad1 de la posicion 15 hasta la 17
    //Por toda la cad4
    cad1.replace(cad1.begin()+15,cad1.begin()+17,cad4.begin(),cad4.end());
    cout<<"cad1: "<<cad1<<endl;//Muestra: El estudiar es super util.
    //Reemplazando los caracteres de la cad1 de la posicion 15 hasta la 20
    //Por toda la cad5
    cad1.replace(cad1.begin()+15,cad1.begin()+20,cad5);
    cout<<"cad1: "<<cad1<<endl;//Muestra: El estudiar es demasiado util.
    //Reemplazando los caracteres de la cad1 de la posicion 11 hasta el final
    //Por la secuencia de caracteres de la cad3
    //a partir de la posicion 8 hasta el final-1
    cad1.replace(cad1.begin()+11,cad1.end(),cad3.begin()+8,cad3.end()-1);
    cout<<"cad1: "<<cad1<<endl;//Muestra: El estudiar much
    //Reemplazando los caracteres de la cad1 de la posicion final-2
    //hasta el final
    //Por 3 copias del caracter 'y'
    cad1.replace(cad1.end()-2,cad1.end(),3,'y');
    cout<<"cad1: "<<cad1<<endl;//Muestra: El estudiar muyyy
    return 0;
}
```

`cad1.replace(i,n,cad2)`	Donde cad1 y cad2 son de tipo string, i y n son enteros no negativos, reemplaza n caracteres desde el i-esimo carácter de la cadena cad1 por toda la cadena cad2.
`cad1.replace(i,n,cad2,j,m)`	Donde cad1 y cad2 son de tipo string, i, n, j y m son enteros no negativos, reemplaza n caracteres desde el i-esimo carácter de la cadena cad1 por m caracteres de la cadena cad2 desde el j-esimo caracter.
`cad1.replace(i,n,m,c)`	Donde cad1 es de tipo string, i, n y m son enteros positivos y c es de tipo char, reemplaza n caracteres desde el i-esimo carácter de la cadena cad1 por m copias del carácter c.
`A.replace(itA1, itA2, itB1, itB2)`	Donde A y B son de tipo string, itA1 e itA2 son iteradores de A, it1B e itB2 son iterdadroes de B, reemplaza la secuencia de caracteres desde la posición de it1A hasta la posición itA2 de la cadena cad1 por caracteres de la cadena B desde la posición itB1 hasta la posición itB2.
`cad1.replace(it1, it2,cad2)`	Donde cad1 y cad2 son de tipo string, it1 e it2 son iteradores de cad1, reemplaza la secuencia de caracteres de la cadena cad1 desde la posición de it1 hasta la posición de it2 por toda la cadena cad2.
`cad.replace(it1, it2,n,c)`	Donde cad es de tipo string, it1 e it2 son iteradores de cad, n es un entero no negativo y c es de tipo char, reemplaza la secuencia de caracteres de

	cad desde la posición it1 hasta la posición it2 por n copias del carácter c.

3.1.6 LONGITUD DE CADENAS

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string x,y;
    x.assign("Esta es una cadena");

    cout<<"LONGITUD DE LA CADENA REPRESENTADO EN NUMERO DE CARACTERES"<<endl;
    //Numero de caracteres en una cadena
    cout<<"Funcion length: "<<x.length()<<endl;//Muestra: 18
    cout<<"Funcion size: "<<x.size()<<endl;//Muestra: 18
    cout<<"Funcion max_size: "<<x.max_size()<<endl;//Muestra: 1073741820

    cout<<"\nMODIFICANDO LA LONGITUD DE LA CADENA"<<endl;
    //Modificando el tamaño de 18 a 13 caracteres
    x.resize(13);
    cout<<"Cadena x: "<<x<<endl;//Muestra: Esta es una c
    //Modificando el tamaño de 13 a 15 caracteres
    x.resize(15);
    cout<<"Cadena x: "<<x<<endl;//Muestra: Esta es una c
    //Modificando el tamaño de 15 a 20 caracteres
    x.resize(20,'!');
    cout<<"Cadena x: "<<x<<endl;//Muestra: Esta es una c  !!!!!

    cout<<"\nPROBANDO SI UNA CADENA ESTA VACIA"<<endl;
    cout<<"La cadena x esta vacia?"<<endl;
    cout<<((x.empty())?"Esta vacia":"No, tiene caracteres")<<endl;
    //Muestra: No, tiene caracteres
    cout<<"La cadena y esta vacia?"<<endl;
    cout<<((y.empty())?"Esta vacia":"No, tiene caracteres")<<endl;
    //Muestra: Esta vacia
    return 0;
}
```

`length()`	Devuelve un entero positivo que representa el tamaño (nro de caracteres) de una cadena.
`size()`	Devuelve un entero positivo que representa el tamaño (nro de caracteres) de una cadena.
`max_size()`	Devuelve un entero positivo que representa el máximo tamaño (nro de caracteres) de una cadena.
`resize(n)`	Donde n es un entero no negativo, modifica el tamaño o longitud de una cadena. Si n es un numero menor al tamaño actual de una cadena, esta cadena conserva los primeros n caracteres, mientras que los demás caracteres se borraran y la nueva longitud será n, pero si n es mayor al tamaño actual de una cadena, entonces esta cadena se rellenará con caracteres de espacio ' ' y la nueva longitud será n.
`resize(n,c)`	Donde n es un entero no negativo y c es de tipo char, modifica el tamaño o longitud de una cadena. Si n es un número menor al tamaño actual de una cadena, esta cadena conserva los primeros n caracteres, mientras que los demás caracteres se borraran y la nueva longitud será n, pero si n es mayor al tamaño actual de una cadena, entonces esta cadena aumentara rellenará la cadena con caracteres c y la nueva longitud será n.
`empty()`	Devuelve un valor de tipo falso/verdadero, si la cadena esta vacía (longitud 0) devuelve **true (verdadero),** si la cadena no esta vacía, devulve **false (falso).**

3.1.7 ACCESO A LOS ELEMENTOS DE UNA CADENA

```
#include<iostream>
#include<string>
```

```
using namespace std;
int main(){
    string cadena="Programacion";
    cout<<"FUNCION AT"<<endl;
    for(int i=0;i<cadena.length();i++)
        cout<<cadena.at(i)<<endl;//Muestra 'Programacion' caracter por caracter.

    cout<<"\nOPERADOR []"<<endl;
    for(int i=0;i<cadena.length();i++)
        cout<<cadena[i]<<endl; //Muestra 'Programacion' caracter por caracter.
    return 0;
}
```

`at(i)`	Donde i es un entero no negativo, la función at accede al i-esimo carácter de una cadena.
`[i]`	Donde i es un entero no negativo, el operador [] accede al i-esimo carácter de una cadena.

3.1.8 ITERADORES DE CADENA

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string cadena="Programacion";

    cout<<"ITERADORES HACIA ADELANTE"<<endl;
    //Creamos un iterador de cadena
    string::iterator it;
    for(it=cadena.begin();it!=cadena.end();it++)
        cout<<*it;//Muestra: Programacion
    cout<<endl;

    cout<<"\nITERADORES DE REVERSA"<<endl;
    //Creamos un iterador reverso de cadena
    string::reverse_iterator rit;
    for(rit=cadena.rbegin();rit!=cadena.rend();rit++)
        cout<<*rit;//Muestra: noicamargorP
    cout<<endl;

    //EJEMPLOS CON OTRAS FUNCIONES
    cout<<"\nINSERCCION DE CARACTERES EN UNA CADENA"<<endl;
    //El iterador apunta al princio de la cadena
    it=cadena.begin();
    string w="Viva la ";
    //Introduciendo la cadena w en la cadena
    cadena.insert(it,w.begin(),w.end());
    //Es lo mismo que hacer: cadena.insert(cadena.begin(),w.begin(),w.end());
    cout<<"cadena: "<<cadena<<endl;//Muestra: Viva la Programacion
    //El iterador apunta en la posicion 4
    it=cadena.begin()+4;
    cadena.insert(it,'!');
    //Es lo mismo que hacer: cadena.insert(cadena.begin()+4,'!');
    cout<<"cadena: "<<cadena<<endl;//Muestra: Viva! la Programacion

    cout<<"\nELIMINACION DE CARACTERES EN UNA CADENA"<<endl;
    //creamos dos iteradores
    string::iterator a,b;
    //El iterador a apunta al principio
    a=cadena.begin();
    //El iterador b apunta en la posicion 9
    b=cadena.begin()+9;
    cadena.erase(a,b);
```

```
    //Es lo mismo que hacer: cadena.erase(cadena.begin(),cadena.begin()+9);
    cout<<"cadena: "<<cadena<<endl;//Muestra: Programacion

    cout<<"\nREEMPLAZO DE CADENAS"<<endl;
    w="ndo en C++";
    //El iterador a apunta al final-4
    a=cadena.end()-4;
    //El iterador b apunta al final
    b=cadena.end();
    string::iterator x,y;
    //El iterador a apunta al principio de la cadena w
    x=w.begin();
    //El iterador a apunta a la posicion 3 de la cadena w
    y=w.begin()+3;
    cadena.replace(a,b,x,y);
    //Es lo mismo que hacer:
    //cadena.erase(cadena.end()-4,cadena.end(),w.begin(),w.begin()+3);
    cout<<"cadena: "<<cadena<<endl;//Muestra: Programando
    cadena.replace(a,b,w);
    //cadena.erase(cadena.end()-4,cadena.end(),w);
    cout<<"cadena: "<<cadena<<endl;//Muestra: Programando en C++
    return 0;
}
```

`string::iterator it`	Crea un iterator de cadena denominado it.
`string::reverse_iterator rit`	Crea un iterator reverso de cadena denominado rit.
`begin()`	Iterador que apunta al principio de la cadena.
`end()`	Iterador que apunta al final de la cadena.
`rbegin()`	Iterador reverso que apunta al principio de la cadena invertida.
`rend()`	Iterador reverso que apunta al final de la cadena invertida.

3.1.9 COMPARACION DE CADENAS

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string cad1,cad2,cad3,cad4,cad5,cad6;
    cad1="Universidad";
    cad2="Universidad";
    cad3="cadena";
    cad4="CADENA";
    cad5="Esta es una cadena";
    cad6="madera";

    cout<<"OPERADORES RELACIONALES"<<endl;
    cout<<"Operador == (igual)"<<endl;
    if(cad1==cad2)
        cout<<"cad1 y cad2 son iguales"<<endl;
    else
        cout<<"cad1 y cad2 son diferentes"<<endl;
    //Muestra: cad1 y cad2 son iguales

    cout<<"Operador != (diferente)"<<endl;
    if(cad1!=cad3)
        cout<<"cad1 y cad3 son diferentes"<<endl;
    else
        cout<<"cad1 y cad3 son iguales"<<endl;
```

```
    //Muestra: cad1 y cad3 son diferentes

    cout<<"Operador > (mayor)"<<endl;
    if(cad3>cad4)
        cout<<"cad3 es mayor que cad4"<<endl;
    else
        cout<<"cad3 NO es mayor que cad4"<<endl;
    //Muestra: cad3 es mayor que cad4

    cout<<"Operador < (menor)"<<endl;
    if(cad4<cad3)
         cout<<"cad4 es menor que cad3"<<endl;
    else
         cout<<"cad4 NO es mayor que cad3"<<endl;
    //Muestra: cad4 es menor que cad3

    cout<<"\nFUNCION COMPARE"<<endl;
    if(cad1.compare(cad2)==0)
        cout<<"cad1 y cad2 son iguales"<<endl;
    else
        cout<<"cad1 y cad2 son diferentes"<<endl;
    //Muestra: cad1 y cad2 son iguales

    //Compara la secuencia de 6 caracteres de cad5 a partir del caracter 12
    if(cad5.compare(12,6,cad3)==0)
        cout<<"cad5 contiene a cad3"<<endl;
    else
        cout<<"cad5 NO contiene a cad3"<<endl;
    //Muestra: cad5 contiene a cad3

    //Compara la secuencia de 3 caracteres de cad3 a partir del caracter 1
    //con la secuencia de 3 caracteres de cad6 a partir del caracter 1
    if(cad3.compare(1,3,cad6,1,3)==0)
        cout<<"Secuencia de tres caracteres iguales en cad3 y cad6"<<endl;
    else
        cout<<"Secuencia de tres caracteres diferentes en cad3 y cad6"<<endl;
    //Muestra: La secuencia de tres caracteres son iguales en cad3 y cad6

    if(cad3.compare(cad4)>0)
        cout<<"cad3 es mayor que cad4"<<endl;
    else
        cout<<"cad3 NO es mayor que cad4"<<endl;
    //Muestra: cad3 es mayor que cad4

    if(cad4.compare(cad3)<0)
        cout<<"cad4 es menor que cad3"<<endl;
    else
        cout<<"cad4 NO es mayor que cad3"<<endl;
    //Muestra: cad4 es menor que cad3
    return 0;
}
```

a==b	Donde a y b son de tipo string. El operador == compara las cadenas a y b y devuelve **true** (verdad) si ambas cadenas son iguales, es decir que contienen los mismos caracteres. Caso contrario devuelve **false** (falso).
a!=b	Donde a y b son de tipo string. El operador != compara las cadenas a y b y devuelve **true** (verdad) si ambas cadenas son diferentes, es decir que no contienen los mismos caracteres. Caso contrario devuelve **false** (falso).
a>b	Donde a y b son de tipo string. El operador > compara las cadenas a y b y devuelve **true** (verdad) si la cadena a es mayor que b, es decir que los caracteres de a son mayores que los de b (orden del codigo ASCII). Caso contrario devuelve **false** (falso).

a<b	Donde a y b son de tipo string. El operador < compara las cadenas a y b y devuelve **true** (verdad) si la cadena a es menor que b, es decir que los caracteres de a son menores que los de b (orden del codigo ASCII). Caso contrario devuelve **false** (falso).
a.compare(b)	Donde a y b son de tipo string. La función compare, compara las cadenas a y b. Devuelve un número entero. 0: Si las cadenas a y b son iguales. >0: Si la cadena a es mayor que b <0: Si la cadena a es menor que b
a.compare(i,n,b)	Donde a y b son de tipo string, i y n son enteros no negativos. La función compare compara las cadenas a y b. Devuelve **0** si la secuencia de n caracteres de la cadena a desde el carácter i son iguales a toda la cadena b.
a.compare(i,n,b,j,m)	Donde a y b son de tipo string, i, j, n y m son enteros no negativos. La función compare compara las cadenas a y b. Devuelve **0** si la secuencia de n caracteres de la cadena a desde el carácter i son iguales a la secuencia de m caracteres de la cadena b desde el carácter j.

3.1.10 SUBCADENAS

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string cad="Esta es una subcadena";
    cout<<"Cadena completa: "<<cad<<endl;
    cout<<"Subcadena desde la posicion 5"<<endl;
    cout<<cad.substr(5)<<endl;//Muestra: es una subcadena
    cout<<"Subcadena desde la posicion 12 hasta la posicion 18"<<endl;
    cout<<cad.substr(12,6)<<endl;
    return 0;
}
```

substr(i)	Donde i es un entero no negativo, la función substr obtiene una subcadena de una cadena a partir de la i-esima posición.
substr(i,n)	Donde i y n son enteros no negativos, la función substr obtiene una subcadena tomando la secuencia de n caracteres a partir de la i-esima posición de una cadena.

3.2 FUNCIONES BASICAS DE CADENAS EN C, LIBRERÍA<string.h> (<cstring> en C++)

3.2.1 ASIGNACION Y CONCATENCACION DE CADENAS

```
#include<stdio.h>//Libreria para la entrada y salida de datos
#include<string.h>//Libreria para las funciones de cadenas
int main(){
    char cad1[]={"Esto es"};
    char cad2[]={" un ejemplo"};
    char cad3[]={" de cadenas"};
    char cad4[]={" en C, solo en C"};

    puts("FUNCION STRCAT");
    strcat(cad1,cad2); //Concatenando cad2 en cad1
    puts(cad1);//Muestra: Esto es un ejemplo
    strcat(cad1,cad3);//Concatenando cad3 en cad1
    puts(cad1);//Muestra: Esto es un ejemplo de cadenas

    puts("\nFUNCION STRNCAT");
    strncat(cad1,cad4,5);//Concatenando los 5 primeros caracteres de cad4 en cad1
    puts(cad1);//Muestra: Esto es un ejemplo de cadenas en C
    return 0;
}
```

`strcat(cad1,cad2)`	Donde cad1 y cad2 son de tipo char[], la función strcat concatena la cadena cad2 en la cadena cad1.
`strncat(cad1,cad2,n)`	Donde cad1 y cad2 son de tipo char[] y n es un entero no negativo. La función strncat concatena los primeros n caracteres de la cadena cad2 en la cadena cad1.

3.2.2 COPIA DE CADENAS

```
#include<stdio.h>//Libreria para la entrada y salida de datos
#include<string.h>//Libreria para las funciones de cadenas
int main(){
    char cad1[]={"Cadenas en C"};
    char cad2[50], cad3[50], cad[50];
    puts("FUNCION STRCPY");
    strcpy(cad2,cad1); //Copiando cad1 a cad2
    puts(cad2);//Muestra: Cadenas en C

    puts("\nFUNCION MEMCPY");
    memcpy(cad5,cad1,sizeof(cad1));//Copiando cad1 a cad2
    puts(cad5);//Muestra: Cadenas en C

    puts("\nFUNCION STRNCPY");
    strncpy(cad3,cad1,sizeof(cad1));//Copiando cad1 a cad3
    puts(cad3);//Muestra: Cadenas en C
    strncpy(cad4,cad1,7);//Copiando los primeros 7 caracteres de cad1 en cad4
    cad4[7]='\0';//Caracter nulo añadido manualmente
    puts(cad4);//Muestra: Cadenas
    return 0;
}
```

`strcpy(cad2,cad1)`	Donde cad1 y cad2 son de tipo char[], la función strcpy copia los caracteres de la cadena cad1 en la cadena cad2.
`memcpy(cad2,cad1,n)`	Donde cad1 y cad2 son de tipo char[], n es un entero no negativo, la función memcpy copia los primeros n caracteres de la cadena cad1 en la cadena cad2.
`strncpy(cad1,cad2,n)`	Donde cad1 y cad2 son de tipo char[] y n es un entero no negativo. La función strncpy copia los primeros n caracteres de la cadena cad2 en la cadena cad1.

3.2.3 LONGITUD DE CADENAS

```
#include<stdio.h>//Libreria para la entrada y salida de datos
#include<string.h>//Libreria para las funciones de cadenas
int main(){
    char cad[]={"Cadenas en C"};
    printf("La longitud de la cadena es: %d\n",strlen(cad));//12
    return 0;
}
```

`strlen(cad)`	Donde cad es de tipo char[], la función strlen devuelve un entero no negativo que representa la longitud de una cadena, es decir el numero de caracteres que contiene una cadena.

3.2.4 COMPARACION DE CADENAS

```
#include<stdio.h>
#include<string.h>
int main(){
    char cad1[]="Universidad";
    char cad2[]="Universidad";
```

```
    char cad3[]="cadena";
    char cad4[]="CADENA";
    char cad5[]="Universitario";

    puts("FUNCION MEMCMP");
    if(memcmp(cad1,cad2,sizeof(cad1))==0)
        puts("Las cadenas cad1 y cad2 son iguales");
    else
        puts("Las cadenas cad1 y cad2 son diferentes");
    //Muestra: Las cadenas cad1 y cad2 son iguales

    if(memcmp(cad1,cad3,sizeof(cad1))!=0)
        puts("Las cadenas cad1 y cad3 son diferentes");
    else
        puts("Las cadenas cad1 y cad3 son iguales");
    //Muestra: Las cadenas cad1 y cad3 son diferentes

    if(memcmp(cad3,cad4,sizeof(cad3))>0)
        puts("cad3 es mayor que cad4");
    else
        puts("cad3 NO es mayor que cad4");
    //Muestra: cad3 es mayor que cad4

    if(memcmp(cad4,cad3,sizeof(cad3))<0)
        puts("cad4 es menor que cad3");
    else
        puts("cad4 NO es menor que cad3");
    //Muestra: cad4 es menor que cad3

    puts("\nFUNCION STRCMP");
    if(strcmp(cad1,cad2)==0)
        puts("Las cadenas cad1 y cad2 son iguales");
    else
        puts("Las cadenas cad1 y cad2 son diferentes");
    //Muestra: Las cadenas cad1 y cad2 son iguales

    if(strcmp(cad1,cad3)!=0)
        puts("Las cadenas cad1 y cad3 son diferentes");
    else
        puts("Las cadenas cad1 y cad3 son iguales");
    //Muestra: Las cadenas cad1 y cad3 son diferentes

    if(strcmp(cad3,cad4)>0)
        puts("cad3 es mayor que cad4");
    else
        puts("cad3 NO es mayor que cad4");
    //Muestra: cad3 es mayor que cad4

    if(strcmp(cad4,cad3)<0)
        puts("cad4 es menor que cad3");
    else
        puts("cad4 NO es menor que cad3");
    //Muestra: cad4 es menor que cad3

    puts("\nFUNCION STRNCMP");
    //Comparando los primeros 8 caracteres de cad1 y ccad5
    if(strncmp(cad1,cad5,8)==0)
        puts("Las cadenas cad1 y cad5 son iguales");
    else
        puts("Las cadenas cad1 y cad2 son diferentes");

    //Comparando los primeros 9 caracteres de cad1 y ccad5
    //Muestra: Las cadenas cad1 y cad5 son iguales
```

```
    if(strncmp(cad1,cad5,9)==0)
        puts("Las cadenas cad1 y cad5 son iguales");
    else
        puts("Las cadenas cad1 y cad5 son diferentes");
    //Muestra: Las cadenas cad1 y cad5 son diferentes
    return 0;
}
```

`memcmp(cad1,cad2,n)`	Donde cad1 y cad2 son de tipo char[] y n es un entero no negativo, la función memcmp compara los bloques de memoria de la cadena cad1 y la cadena cad2 los primeros n caracteres, devolviendo un numero entero. 0: si los bloques de memoria son iguales. >0: si cad1 es mayor que cad2 (código ASCII). <0: si cad1 es menor que cad2 (código ASCII).
`strcmp(cad1,cad2)`	Donde cad1 y cad2 son de tipo char[], la función strcmp compara las cadenas cad1 y cad2, devolviendo un numero entero. 0: si las cadenas son iguales. >0: si cad1 es mayor que cad2 (código ASCII). <0: si cad1 es menor que cad2 (código ASCII).
`strncmp(cad1,cad2,n)`	Donde cad1 y cad2 son de tipo char[] y n es un entero no negativo, la función strncmp compara los primeros n caracteres de las cadenas cad1 y cad2, devolviendo un numero entero. 0: si las cadenas son iguales. >0: si cad1 es mayor que cad2 (código ASCII). <0: si cad1 es menor que cad2 (código ASCII).

3.2.5 BUSQUEDA DE CARACTERES

```
#include<stdio.h>
#include<string.h>
int main(){
    char cadena[]="La programacion es genial, ACM ICPC";
    char vocales[]="aeiouAEIOU";
    char x='a';
    char *ptr;//Creando un puntero
    printf("cadena: %s\n",cadena);//Muestra: La programacion es genial, ACM ICPC

    puts("\nFUNCION STRCHR");
    puts("Buscando las posiciones del caracter 'a' en la cadena");
    ptr=strchr(cadena,x);
    while(ptr!=NULL){
        printf("'a' en la posicion %d\n",ptr-cadena);
        ptr=strchr(ptr+1,x);
    }

    puts("\nFUNCION STRCSPN");
    int n=strcspn(cadena,vocales);
    printf("La primera vocal se encuentra en la posicion %d\n",n);

    puts("\nFUNCION STRRCHR");
    ptr=strrchr(cadena,x);
    printf("La ultima posicion de 'a' se encuentra en: %d\n",ptr-cadena);

    puts("\nFUNCION STRPBRK");
    puts("Buscando las posiciones de las vocales en la cadena");
    ptr=strpbrk(cadena,vocales);
    while(ptr!=NULL){
        printf("Vocal localizada en la posicion %d\n",ptr-cadena);
        ptr=strpbrk(ptr+1,vocales);
    }
```

```
    return 0;
}
```

`strchr(cad,x)`	Donde cad es de tipo char[] y x es de tipo char, la función strchr devuelve un puntero que nos inidica la posición en la que se encuentra el carácter x en la cadena cad si es que x esta en cad, pero cuando x no se encuentra en cad, devulve una posición NULL.
`strcspn(A,B)`	Donde A y B son de tipo char[], la función strcspn devuelve un número entero no negativo que representa la primera posición en la cadena A, de algún carácter que pertenece a B.
`strrchr(cad,x)`	Donde cad es de tipo char[] y x es de tipo char, la función strrchr devuelve un puntero que nos inidica la ultima posición en la que se encuentra el carácter x en la cadena cad si es que x esta en cad, pero cuando x no se encuentra en cad, devulve una posición NULL.
`strpbrk(A,B)`	Donde A y B son de tipo char[], la función strpbrk devuelve un puntero que indica posición en la cadena A, de algún carácter que pertenece a B.

3.2.6 TOKENS

```
#include<stdio.h>
#include<string.h>
int main(){
    char cad1[]="Este es el primer ejemplo";
    char cad2[]="Este!!!es, el segundo?. Ejemplo-";
    char *token;
    token=strtok(cad1," ");
    while(token!=NULL){
        printf("Palabra: %s \nLongitud: %d\n",token,strlen(token));
        token=strtok(NULL," ");
    }
    puts("");
    puts(cad2);
    token=strtok(cad2," ,.!?");
    while(token!=NULL){
        printf("Palabra: %s \nLongitud: %d\n",token,strlen(token));
        token=strtok(NULL," ,.!?-");
    }
    return 0;
}
```

```
//Se muestra por pantalla
Longitud: 4
Palabra: es
Longitud: 2
Palabra: el
Longitud: 2
Palabra: primer
Longitud: 6
Palabra: ejemplo
Longitud: 7

Este!!!es, el segundo?. Ejemplo-
Palabra: Este
Longitud: 4
Palabra: es
Longitud: 2
Palabra: el
Longitud: 2
Palabra: segundo
Longitud: 7
Palabra: Ejemplo
Longitud: 7
```

`strtok(cad1,cad2)`	Donde cad1 y cad2 son de tipo char[],la función strtok devuleve un puntero que apunta al final de una secuencia de caracteres de cad1 que no contiene un elemento de cad2. Es muy útil para dividir una oración.

3.2.7 MEMSET

```
#include<stdio.h>//Librería de entrada y salida
#include<string.h>//Librería para la herramienta memset
int main(){
    int i,n=7;
    int v[n];//Un vector de enteros
    char cad[]="Esta es una cadena";//Una cadena

    memset(v,0,sizeof(v));//Llenamos el vector enteros con 0
    memset(cad,'*',6);//Llenamos los primeros 9 caracteres de la con el caracter '*'

    puts("El contenido del vector de entero es:");
    for(i=0;i<n;i++)
        printf("%d ",v[i]);//Muestra: 0 0 0 0 0 0 0
    puts("");

    puts("\nEl contenido de la cadena es:");
    puts(cad); //Muestra: **********a cadena
    return 0;
}
```

`memset(v,0,n)`	Donde v es un vector numerico y n es un entero no negativo, la función memset llena los primeros n elementos del vector v con el elemento 0 (Unicamente este numero).
`memset(cad,x,n)`	Donde cad es de tipo char[], x es de tipo char y n es un entero no negativo, la función memset llena los primeros n elementos de la cadena cad con el carácter x.

3.3 COMPATIBILIDAD ENTRE CADENAS DE C++ Y CADENAS DE C

3.3.1 TRANSFORMANDO UN STRING A UNA SECUENCIA DE CARACTERES char[]

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
using namespace std;
int main(){
    string cad1="Esta es una cadena en C++";
    char cad2[100];
    char cad3[100];
    char cad4[100];
    puts("FUNCION C_STR");
    strcpy(cad2,cad1.c_str());
    puts(cad2);//Muestra: Esta es una cadena en C++

    puts("\nFUNCION DATA");
    strcpy(cad3,cad1.data());
    puts(cad3);//Muestra: Esta es una cadena en C++

    puts("\nFUNCION COPY");
    cad1.copy(cad4,13,5);
    puts(cad4);//Muestra: es una cadena
    return 0;
}
```

`c_str()`	Dada una cadena de tipo string, la función c_str obtiene una secuencia de caracteres char[]

	equivalente a la cadena string.
`data()`	Dada una cadena de tipo string, la función data obtiene una secuencia de caracteres char[] equivalente a la cadena string.
`A.copy(B,n,m)`	Donde A es de tipo string, B es de tipo char[], n y m son números enteros no negativos. La función copy, copia n caracteres desde la posición m de la cadena A a la cadena B.

3.3.2 TRANSFORMANDO UNA SECUENCIA DE CARACTERES char[] A UN STRING

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int main(){
    char cad1[]="Esta es una cadena en C";
    string cad2;
    puts("OPERADOR DE ASIGNACION =");
    cad2=cad1;
    cout<<cad2<<endl;//Muestra: Esta es una cadena en C
          return 0;
}
```

`A=B`	Donde A es de tipo string, B es de tipo char[], el operador '=' asigna el contenido de B a la cadena string A, como se puede observar, es sencilla llevar una char[] a un string.

3.4 MANEJO DE CARACTERES

3.4.1 CLASIFICACION DE CARACTERES CON LA LIBRERÍA <cctype> (<ctype.h> en C)

```
//Programa que cuenta los distintos tipos de carácter
#include<iostream>//Librería para la salida de datos
#include<cctype>//Librería para la clasificación de caracteres
using namespace std;
int main(){
    string cad="11 de Agosto, de 2013!\nEl Octavo Mes.\n";
    int nalnum,nnum,nmayus,nminus,nletras,nsigno,nesp,ncontrol,nimp;
    nalnum=nnum=nletras=nsigno=nesp=ncontrol=nimp=nmayus=nminus=0;
    for(int i=0;i<cad.length();i++){
        //El caracter es imprimible?
        if(isprint(cad[i]))
            nimp++;
        //El caracter es de control?
        if(iscntrl(cad[i]))
            ncontrol++;
        //El caracter es numero o letra?
        if(isalnum(cad[i]))
            nalnum++;
        //El caracter es letra?
        if(isalpha(cad[i])){
            nletras++;
            //La letra es mayuscula?
            if(isupper(cad[i]))
                nmayus++;
            //La letra es minuscula?
            if(islower(cad[i]))
                nminus++;
        }
        //El caracter es numero?
        if(isdigit(cad[i]))
            nnum++;
        //El caracter es signo de puntuacion?
```

```
        if(ispunct(cad[i]))
            nsigno++;
        //El caracter es espacio?
        if(isspace(cad[i]))
            nesp++;
    }
    cout<<"Numero total de caracteres: "<<cad.length()<<endl;//38
    cout<<"Numero de caracteres imprimibles: "<<nimp<<endl;//36
    cout<<"Numero de caracteres de control: "<<ncontrol<<endl;//2
    cout<<"Numero de caracteres alfanumericas: "<<nalnum<<endl;//27
    cout<<"Numero de letras: "<<nletras<<endl;//21
    cout<<"Numero de letras mayusculas: "<<nmayus<<endl;//4
    cout<<"Numero de letras minusculas: "<<nminus<<endl;//17
    cout<<"Numero de digitos: "<<nnum<<endl;//6
    cout<<"Numero de signos de puntuacion: "<<nsigno<<endl;//3
    cout<<"Numero de espacios en blanco: "<<nesp<<endl;//8
    return 0;
}
```

`isprint(x)`	Donde x es de tipo char, la función isprint devuelve **true** (verdadero) si el carácter es imprimible, caso contrario devuelve **false** (falso).
`iscntrl(x)`	Donde x es de tipo char, la función iscntrl devuelve **true** (verdadero) si el carácter es de control, caso contrario devuelve **false** (falso).
`isalnum(x)`	Donde x es de tipo char, la función isalnum devuelve **true** (verdadero) si el carácter es letra o numero, caso contrario devuelve **false** (falso).
`isalpha(x)`	Donde x es de tipo char, la función isalpha devuelve **true** (verdadero) si el carácter es letra, caso contrario devuelve **false** (falso).
`isupper(x)`	Donde x es de tipo char, la función isupper devuelve **true** (verdadero) si el carácter es letra mayúscula, caso contrario devuelve **false** (falso).
`islower(x)`	Donde x es de tipo char, la función islower devuelve **true** (verdadero) si el carácter es letra minúscula, caso contrario devuelve **false** (falso).
`isdigit(x)`	Donde x es de tipo char, la función isdigit devuelve **true** (verdadero) si el carácter es numero, caso contrario devuelve **false** (falso).
`ispunct(x)`	Donde x es de tipo char, la función ispunct devuelve **true** (verdadero) si el carácter es signo de puntuación, caso contrario devuelve **false** (falso).
`isspace(x)`	Donde x es de tipo char, la función isspace devuelve **true** (verdadero) si el carácter es espacio en blanco, caso contrario devuelve **false** (falso).

3.5 CONVERSION DE CADENAS

3.5.1 CONVERSION DE MAYUSCULAS A MINUSCULAS Y VICEBERSA CON LA LIBRERÍA <cctype> (<ctype.h> en C)

```
#include<iostream>//Libreria para la salida de datos
#include<cctype>//Libreria para la conversion de caracteres letras
using namespace std;
int main(){
    string cad="Programacion En Lenguaje C/C++";
    for(int i=0;i<cad.size();i++){
        if(isalpha(cad[i]))
            cad[i]=(isupper(cad[i]))?tolower(cad[i]):toupper(cad[i]);
    }
    cout<<cad<<endl;//Muestra: pROGRAMACION eN lENGUAJE c/c++
    return 0;
}
```

`toupper(x)`	Donde x es de tipo char, la función toupper devuelve el carácter x transformado en mayúscula.

`tolower(x)`	Donde x es de tipo char, la función tolower devuelve el carácter x transformado en minúscula.

3.5.2 CONVERSION DE CADENAS A NUMEROS CON LA LIBRERÍA <cstdlib> (<stdlib.h> en C)

```
#include<iostream>//Libreria para la salida de datos
#include<cstdlib>//Libreria para la conversion de numeros
using namespace std;
int main(){
    int num1;
    long num2;
    double num3;
    string cad1="2147483647";
    string cad2="123.056";
    num1=atoi(cad1.data());//Transformando una cadena a un numero entero
    cout<<"Numero tipo int: "<<num1<<endl;
    num2=atol(cad1.data());//Transformando una cadena a un numero long
    cout<<"Numero tipo long: "<<num2<<endl;
    num3=atof(cad2.data());//Transformando una cadena a un real double
    cout<<"Numero tipo double: "<<num3<<endl;
    return 0;
}
```

`atoi(x)`	Donde x es de tipo char[] (cadena en C), la función atoi transforma la cadena en un numero entero **int**.
`atol(x)`	Donde x es de tipo char[] (cadena en C), la función atol transforma la cadena en un numero entero **long**.
`atof(x)`	Donde x es de tipo char[] (cadena en C), la función atof transforma la cadena en un numero real **double**.

3.5.3 CONVERSION DE NUMEROS A CADENAS CON LA LIBRERÍA <sstream>

```
#include<iostream>
#include<sstream>//Libreria para transformar el numero a cadena
using namespace std;
int main(){
    int num1=1153;
    long long num2 = 9223372036854775807;
    double num3=123.435;
    string cad1,cad2,cad3;
    //Creamos dos objetos stringstream
    stringstream conv1,conv2;

    //Transformando un solo numero a cadena
    conv1<<num1;
    cad1=conv1.str();
    cout<<"cadena1: "<<cad1<<endl;//1153

    //Transformando dos numeros a cadenas
    conv2<<num2<<' '<<num3;
    conv2>>cad2>>cad3;
    cout<<"cadena2: "<<cad2<<endl;//9223372036854775807;
    cout<<"cadena3: "<<cad3<<endl;//123.435
    return 0;
}
```

4 VECTORES, MATRICES Y PUNTEROS EN C/C++

4.1 MANEJO DE VECTORES

4.1.1 CREACION DE VECTORES

```
#include<iostream>
using namespace std;
void llenar(int x[],int m){
    for(int i=0;i<m;i++)
        x[i]=i*10;
}
void mostrar(int x[],int m){
    for(int i=0;i<m;i++)
        cout<<x[i]<<" ";
    cout<<endl;
}
int* crear_vector(int dim){
    int *x=new int[dim];
    for(int i=0;i<dim;i++)
        x[i]=(i*2);
    return x;
}
int main(){
    int n=5;//Variable que sera la dimension de los vectores
    int vec1[n];//Creando el array estatico vec1 de manera sencilla
    llenar(vec1,n);//Metodo para llenar el vector
    mostrar(vec1,n);//Muestra: 0 10 20 30 40
    int *vec2; //Creando el array dinamico vec2
    vec2 = new int[n];//Dimensionando vec2
    llenar(vec2,n); //Metodo para llenar el vector
    mostrar(vec2,n);//Muestra: 0 10 20 30 40
    int *vec3=crear_vector(n);//Crea el array vec3 mediante el método crear_vector
    mostrar(vec3,n);//Muestra: 0 2 4 6 8
    return 0;
}
```

`Tipo_dato v[m]`	Donde v es un vector estatico, m es un entero positivo, crea un vector de m bloques donde se pueden almacenar datos del mismo **Tipo de dato.**
`Tipo_dato *v = new Tipo_dato [m]`	Donde v es un vector, m es un entero positivo, crea un vector de m bloques donde se pueden almacenar datos del mismo **Tipo de dato.**

4.1.2 CREACION DE MATRICES

```
#include<iostream>
using namespace std;
int** crear_matriz(int rows,int cols){
    int **x=new int*[rows];
    int c=1;
    for(int i=0;i<rows;i++){
        x[i]=new int[cols];
        for(int j=0;j<cols;j++){
            x[i][j]=c++;
        }
    }
    return x;
}
void show(int **w,int n,int m){
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++)
            cout<<w[i][j]<<" ";
```

```
        cout<<endl;
    }
}
int main(){
    int filas, columnas;
    filas=2;
    columnas=3;
    //Primera manera de crear una matriz: sencillo
    int mat1[filas][columnas];
    //Llenando la matriz mat1
    for(int i=0;i<filas;i++){
        for(int j=0;j<columnas;j++)
            mat1[i][j]=filas*i+j;
    }
    //Mostrando la matriz mat1
    for(int i=0;i<filas;i++){
        for(int j=0;j<columnas;j++)
            cout<<mat1[i][j]<<" ";
        cout<<endl;
    }
    /*
    Muestra:
    0 1 2
    3 4 5
    */
    //Segunda manera de crear una matriz
    //Mediante punteros
    int **mat2;
    mat2=new int*[filas];
    //Llenando la matriz mat2
    for(int i=0;i<filas;i++){
        mat2[i]=new int[columnas];
        for(int j=0;j<columnas;j++)
            mat2[i][j]=(filas*i+j)*2;
    }
    //Mostrando la matriz mat2
    cout<<endl;
    show(mat2,filas,columnas);
    /*
    Muestra:
    0 2 4
    6 8 10
    */
    //Tercera manera de crear una matriz
    //Mediante el metodo crear_matriz
    int **mat3=crear_matriz(filas,columnas);
    //Mostrando la matriz mat2
    cout<<endl;
    show(mat3,filas,columnas);
    /*
    Muestra:
    1 2 3
    4 5 6
    */
    return 0;
}
```

`Tipo_dato w[n][m]`	Donde w es una matriz estatica, n y m son enteros positivos, crea una matriz de n filas por m columnas donde se pueden almacenar datos del mismo **Tipo de dato.**
`Tipo_dato **w = new Tipo_dato* [n]` `Tipo_dato w[i] = new Tipo_dato [m]`	Donde w es una matriz, i, m y n son enteros positivos, crea una matriz de n filas. A continuación por cada i-esima fila se debe crear un array de m columnas dentro de la matriz.

4.1.3 PLANTILLAS DE FUNCION

```
#include<iostream>
using namespace std;
//Metodo para llenar un vector de cualquier tipo de dato
template <class R>
void llenar_vector(R vec[],int m){
    for(int i=0;i<m;i++)
        cin>>vec[i];
}
//Metodo para mostrar por pantalla un vector de cualquier tipo de dato
template <class R>
void mostrar_vector(R vec[],int m){
    for(int i=0;i<m;i++)
        cout<<vec[i]<<" ";
    cout<<endl;
}
//Metodo para sumar dos vectores de dos tipos de dato
template <class R, class S>
void sumar(R a[], S b[],int m){
    for(int i=0;i<m;i++){
        cout<<a[i]<<"+"<<b[i]<<"="<<(S)(a[i]+b[i])<<endl;
    }
}
int main(){
    int n=7;
    //Declarando vectores
    int v1[n];
    double v2[n];
    string v3[n];
    char v4[n];
    //Lectura de vectores
    cout<<"Introduzca siete numeros enteros"<<endl;
    llenar_vector(v1,n);
    cout<<"Mostrando el contenido del vector de enteros"<<endl;
    mostrar_vector(v1,n);

    cout<<"\nIntroduzca siete numeros reales"<<endl;
    llenar_vector(v2,n);
    cout<<"Mostrando el contenido del vector de reales"<<endl;
    mostrar_vector(v2,n);

    cout<<"\nIntroduzca siete cadenas"<<endl;
    llenar_vector(v3,n);
    cout<<"Mostrando el contenido del vector de cadenas"<<endl;
    mostrar_vector(v3,n);

    cout<<"\nIntroduzca siete caracteres"<<endl;
    llenar_vector(v4,n);
    cout<<"Mostrando el contenido del vector de caracteres"<<endl;
    mostrar_vector(v4,n);

    cout<<"\nSumando el vector de enteros con el de reales"<<endl;
    sumar(v1,v2,n);
    return 0;
}
```

4.1.4 PUNTEROS

```
#include<iostream>
using namespace std;
int main(){
    int* p; //Declaramos un puntero de entero
```

```
    int a,b,c; //Declaramos variables enteras
    p=&a;//Apuntamos el puntero p hacia la direccion en memoria de la variable a

    cout<<p<<endl;//0x22ff18
    p=&b; //Apuntamos el puntero p hacia la direccion en memoria de la variable b
    cout<<p<<endl;//0x22ff14
    p=&c; //Apuntamos el puntero p hacia la direccion en memoria de la variable c
    cout<<p<<endl;//0x22ff10
    return 0;
}
```

4.1.5 PUNTEROS EN VECTORES

```
#include<iostream>
using namespace std;
int main(){
    int n=7;
    int v[n];
    for(int i=0;i<n;i++)
        v[i]=2*i;
    int* p;
    for(p=v;p!=v+n;p++){
        cout<<*p<<" ";
    }//Muestra:0 2 4 6 8 10 12
    return 0;
}
```

4.2 METODOS DE ORDENAMIENTO

4.2.1 METODO BURBUJA

```
void bubblesort(int x[],int m){
    int aux;
    for(int i=0;i<m-1;i++){
        for(int j=i+1;j<m;j++){
            if(x[i]>x[j]){
                aux=x[i];
                x[i]=x[j];
                x[j]=aux;
            }
        }
    }
}
```

4.2.2 METODO DE SELECCIÓN

```
void selectionsort(int x[],int m){
    int min,aux;
    for(int i=0;i<m-1;i++){
        min=i;
        for(int j=i+1;j<m;j++){
            if(x[j]<x[min])
                min=j;
        }
        aux=x[i];
        x[i]=x[min];
        x[min]=aux;
    }
}
```

4.2.3 METODO DE INSERCCION

```
void insertionsort(int x[],int m){
    for(int i=1;i<m;i++){
        int aux=x[i];
        int j=i;
        while(j>0 and x[j-1]>=aux){
            x[j]=x[j-1];
            j--;
        }
        x[j]=aux;
    }
}
```

4.2.4 METODO POR INTERCALACION

El metodo por intercalación (merge) es muy rápido pero consume mucha memoria,

```
#include<iostream>
#define MAX 10000
using namespace std;
int v[MAX];
void mergesort(int a,int b){
    if(b-a>1){
        int medio=(a+b)/2;
        mergesort(a,medio);
        mergesort(medio,b);
        int x,y,z,result[b];
        x=y=a;
        z=medio;
        while(x<b){
            if(y==medio)
                result[x++]=v[z++];
            else if(z==b)
                result[x++]=v[y++];
            else if(v[y]<v[z])
                result[x++]=v[y++];
            else
                result[x++]=v[z++];
        }
        while(a<b)
            v[a++]=result[a];
    }
}
int main(){
    int n=7;
    //Llenando los datos
    for(int i=0;i<n;i++)
        cin>>v[i];
    mergesort(0,n);
    //Mostrando los datos
    for(int i=0;i<n;i++)
        cout<<v[i]<<" ";
    return 0;
}
```

4.2.5 METODO RAPIDO

```
void quicksort(int x[],int m){
    int i,a,b;
    if(m>1){
```

```
        int centro=x[m/2];
        int der[m];
        int izq[m];
        a=b=0;
        for(i=0;i<m;i++){
            if(i!=m/2){
                if(x[i]<=centro)
                    izq[a++]=x[i];
                else
                    der[b++]=x[i];
            }
        }
        i=0;
        quicksort(izq,a);
        for(int j=0;j<a;j++)
            x[i++]=izq[j];
        x[i++]=centro;
        quicksort(der,b);
        for(int j=0;j<b;j++)
            x[i++]=der[j];
    }
}
```

4.3 METODOS DE BUSQUEDA

4.3.1 BUSQUEDA LINEAL

```
int linearsearch(int x[],int m,int dato){
    for(int i=0;i<m;i++){
        if(dato==x[i])
            return i;
    }
    return -1;
}
```

4.3.2 BUSQUEDA BINARIA

```
int binarysearch(int x[],int dato,int a,int b){
    int centro;
    while(a<=b){
        centro=(a+b)/2;
        if(dato==x[centro])
            return centro;
        else{
            if(dato<x[centro])
                b=centro-1;
            else
                a=centro+1;
        }
    }
    return -1;
}
```

5 OBJETOS Y CLASES

5.1 DEFINICION DE ESTRUCTURAS Y CLASES

5.1.1 DEFINICION DE ESTRUCTURAS

```
struct persona{
    //Atributos
    string nombre;//nombre
    string ap_paterno;//apellido paterno
    string ap_materno;//apellido materno
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
};
```

5.1.2 DEFINICION DE CLASES

```
class persona{
    //Atributos
    string nombre;
    string ap_paterno;
    string ap_materno;
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
};
```

5.1.3 CONSTRUCTORES Y DESTRUCTORES

```
#include<iostream>
using namespace std;
struct persona{
    string nombre;
    string ap_paterno;
    string ap_materno;
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
    //CONSTRUCTORES: Inicialización de datos
    //Constructor de persona
    persona();
    //Constructor de persona con tres parametros (nombre y apellidos)
    persona(string,string,string);
    //Constructor de persona con tres parametros (la fecha de nacimiento)
    persona(int,int,int);
    //Constructor de persona con seis parametros (todos los atributos)
    persona(string,string,string,int,int,int);
    //DESTRUCTOR
    ~persona();
};
persona::persona(){
    nombre="";
    ap_paterno="";
    ap_materno="";
    d=m=a=0;
}
```

```
persona::persona(string n,string ap,string am){
    nombre=n;
    ap_paterno=ap;
    ap_materno=am;
    d=m=a=0;
}
persona::persona(int x,int y,int z){
    nombre="";
    ap_paterno="";
    ap_materno="";
    d=x;
    m=y;
    a=z;
}
persona::persona(string n,string ap,string am,int x,int y,int z){
    nombre=n;
    ap_paterno=ap;
    ap_materno=am;
    d=x;
    m=y;
    a=z;
}
persona::~persona(){
    cout<<"Destruyendo el objeto persona"<<endl;
}
int main(){
    //CREACION DE PERSONAS
    //Constructor sin 50arámetros
    persona A;
    //Constructor de persona con tres 50arámetros (nombre y apellidos)
    persona B("Jose Gonzalo","Espejo","Cuba");
    //Constructor de persona con tres 50arámetros (la fecha de nacimiento)
    persona C(6,10,1996);
    //Constructor de persona con seis 50arámetros (todos los atributos)
    persona D("Juan","Perez","Torrico",24,9,1999);
    return 0;
}
```

```
//Esta es otra manera de utilizar los constructores
#include<iostream>
using namespace std;
struct persona{
    string nombre;
    string ap_paterno;
    string ap_materno;
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
    //CONSTRUCTORES: Inicialización de datos
    //Constructor de persona, los datos se colocan en valor nulo
    persona(){}
    //Constructor de persona con tres parametros (nombre y apellidos)
    persona(string n,string ap,string am){
        nombre=n;
        ap_paterno=ap;
        ap_materno=am;
        d=m=a=0;
    }
    //Constructor de persona con tres parametros (la fecha de nacimiento)
    persona(int x,int y,int z){
        nombre="";
        ap_paterno="";
        ap_materno="";
```

```
        d=x;
        m=y;
        a=z;
    }
    //Constructor de persona con seis parametros (todos los atributos)
    persona(string n,string ap,string am,int x,int y,int z){
        nombre=n;
        ap_paterno=ap;
        ap_materno=am;
        d=x;
        m=y;
        a=z;
    }
};
int main(){
    //CREACION DE PERSONAS
    //Constructor sin parametros
    persona A;
    //Constructor de persona con tres parametros (nombre y apellidos)
    persona B("Jose Gonzalo","Espejo","Cuba");
    //Constructor de persona con tres parametros (la fecha de nacimiento)
    persona C(6,10,1996);
    //Constructor de persona con seis parametros (todos los atributos)
    persona D("Juan","Perez","Torrico",24,9,1999);
    return 0;
}
```

5.1.4 CONSTRUCTORES CON CLASE

```
#include<iostream>
using namespace std;
class persona{
    private:
    string nombre;
    string ap_paterno;
    string ap_materno;
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
    public:
    //CONSTRUCTORES: Inicialización de datos
    //Constructor de persona
    persona();
    //Constructor de persona con tres parametros (nombre y apellidos)
    persona(string,string,string);
    //Constructor de persona con tres parametros (la fecha de nacimiento)
    persona(int,int,int);
    //Constructor de persona con seis parametros (todos los atributos)
    persona(string,string,string,int,int,int);
};
persona::persona(){
    nombre="";
    ap_paterno="";
    ap_materno="";
    d=m=a=0;
}
persona::persona(string n,string ap,string am){
    nombre=n;
    ap_paterno=ap;
    ap_materno=am;
    d=m=a=0;
```

```
}
persona::persona(int x,int y,int z){
    nombre="";
    ap_paterno="";
    ap_materno="";
    d=x;
    m=y;
    a=z;
}
persona::persona(string n,string ap,string am,int x,int y,int z){
    nombre=n;
    ap_paterno=ap;
    ap_materno=am;
    d=x;
    m=y;
    a=z;
}
int main(){
    //CREACION DE PERSONAS
    //Constructor sin parametros
    persona A;
    //Constructor de persona con tres parametros (nombre y apellidos)
    persona B("Jose Gonzalo","Espejo","Cuba");
    //Constructor de persona con tres parametros (la fecha de nacimiento)
    persona C(6,10,1996);
    //Constructor de persona con seis parametros (todos los atributos)
    persona D("Juan","Perez","Torrico",24,9,1999);
    return 0;
}
```

5.2 FUNCIONES MIEMBRO

5.2.1 EMPLEO DE LAS FUNCIONES MIEMBRO

```
#include<iostream>
using namespace std;
struct semaforo{
    bool foco_rojo;
    bool foco_amarillo;
    bool foco_verde;
    semaforo();
    void encender_rojo();
    void encender_amarillo();
    void encender_verde();
    bool puede_avanzar();
};
semaforo::semaforo(){
    foco_rojo=foco_amarillo=foco_verde=0;
}
void semaforo::encender_rojo(){
    //Encendemos el foco rojo y apagamos los demas
    foco_rojo=1;
    foco_amarillo=0;
    foco_verde=0;
}
void semaforo::encender_amarillo(){
    //Encendemos el foco amarillo y apagamos los demas
    foco_rojo=0;
    foco_amarillo=1;
    foco_verde=0;
}
void semaforo::encender_verde(){
    //Encendemos el foco verde y apagamos los demas
```

```
    foco_rojo=0;
    foco_amarillo=0;
    foco_verde=1;
}
bool semaforo::puede_avanzar(){
    //Si el foco verde esta encendido devuelve verdadero
    return(foco_verde);
}
int main(){
    semaforo S;
     S.encender_rojo();
    if(S.puede_avanzar())
        cout<<"Luz verde encendida, puede avanzar"<<endl;
    else
        cout<<"Luz verde apagada, debe detenerse"<<endl;

    S.encender_amarillo();
    if(S.puede_avanzar())
        cout<<"Luz verde encendida, puede avanzar"<<endl;
    else
        cout<<"Luz verde apagada, debe detenerse"<<endl;

    S.encender_verde();
    if(S.puede_avanzar())
        cout<<"Luz verde encendida, puede avanzar"<<endl;
    else
         cout<<"Luz verde apagada, debe detenerse"<<endl;
   return 0;
}
```

5.2.2 SOBRECARGA DE FUNCIONES

```
#include<iostream>
using namespace std;
class persona{
    private:
    string nombre;
    string ap_paterno;
    string ap_materno;
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
    public:
    persona();
    void colocar_dato(string,string,string);
    void colocar_dato(int,int,int);
    void mostrar();
};
persona::persona(){
    nombre="";
    ap_paterno="";
    ap_materno="";
    d=m=a=0;
}
void persona::colocar_dato(string n,string ap,string am){
    nombre=n;
    ap_paterno=ap;
    ap_materno=am;
}
void persona::colocar_dato(int x,int y,int z){
    d=x;
```

```
    m=y;
    a=z;
}
void persona::mostrar(){
    cout<<"Nombre: "<<nombre<<endl;
    cout<<"Apellido Paterno: "<<ap_paterno<<endl;
    cout<<"Apellido Materno: "<<ap_materno<<endl;
    cout<<"Fecha de nacimiento: ";
    if(d<10)
        cout<<"0";
    cout<<d<<"/";
    if(m<10)
        cout<<"0";
    cout<<m<<"/";
    cout<<a<<endl;
}
int main(){
    //CREACION DE PERSONAS
    //Constructor sin parametros
    persona A;
    A.mostrar();
    /*
    Nombre:
    Apellido Paterno:
    Apellido Materno:
    Fecha de nacimiento: 00/00/0
    */
    //Colocando el nombre y los apellidos
    cout<<endl;
    A.colocar_dato("Jose Goonzalo","Espejo","Cuba");
    A.mostrar();
    /*
    Nombre: Jose Goonzalo
    Apellido Paterno: Espejo
    Apellido Materno: Cuba
    Fecha de nacimiento: 00/00/0
    */
    //Colocando la fecha de nacimiento
    cout<<endl;
    A.colocar_dato(25,11,1986);
    A.mostrar();
    /*
    Nombre: Jose Goonzalo
    Apellido Paterno: Espejo
    Apellido Materno: Cuba
    Fecha de nacimiento: 25/11/1986
    */
    return 0;
}
```

5.3 ACCESO A LOS ATRIBUTOS DE UNA CLASE O UNA ESTRUCTURA

5.3.1 ACCESO A LOS MIEMBROS DONDE LOS ATRIBUTOS SON PRIVADOS

```
#include<iostream>//Libreria para la impresion de datos
#include<iomanip>//Libreria para la presentacion de los datos al imprimirlos
using namespace std;
class persona{
    //ATRIBUTOS
    private:
    string nombre;
    string ap_paterno;
    string ap_materno;
```

```
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
    //FUNCIONES MIEMBRO
    public:
    //Constructor que inicializa los datos en valor nulo
    persona(){}
    void set_nombre(string);
    void set_ap_paterno(string);
    void set_ap_materno(string);
    void set_fecnac(int,int,int);
    string get_nombre();
    string get_ap_paterno();
    string get_ap_materno();
    int get_dia_nac();
    int get_mes_nac();
    int get_year();
    void mostrar_datos();
};
//FUNCIONES PARA MODIFICAR DATOS PRIVADOS
void persona::set_nombre(string n){
    nombre=n;
}
void persona::set_ap_paterno(string ap){
    ap_paterno=ap;
}
void persona::set_ap_materno(string am){
    ap_materno=am;
}
void persona::set_fecnac(int x,int y,int z){
    d=x;
    m=y;
    a=z;
}
//FUNCIONES PARA OBTENER DATOS PRIVADOS
string persona::get_nombre(){
    return nombre;
}
string persona::get_ap_paterno(){
    return ap_paterno;
}
string persona::get_ap_materno(){
    return ap_materno;
}
int persona::get_dia_nac(){
    return d;
}
int persona::get_mes_nac(){
    return m;
}
int persona::get_year(){
    return a;
}
void persona::mostrar_datos(){
    cout<<setw(12)<<left<<"Nombre"<<": "<<nombre<<endl;
    cout<<setw(12)<<left<<"Ap. paterno"<<": "<<ap_paterno<<endl;
    cout<<setw(12)<<left<<"Ap. materno"<<": "<<ap_materno<<endl;
    cout<<setw(12)<<left<<"Fecha nac"<<": "<<d<<"/"<<m<<"/"<<a<<endl;
}
int main(){
    //creamos el objeto persona
    persona p;
```

```
    //Llenamos los datos del objeto persona
    p.set_nombre("Fermin");
    p.set_ap_paterno("Suarez");
    p.set_ap_materno("Gala");
    p.set_fecnac(6,8,2003);
    cout<<"Mostramos los datos mediante la funcion mostrar_datos"<<endl;
    p.mostrar_datos();

    cout<<"\nMostramos los datos mediante las funciones get"<<endl;
    cout<<p.get_nombre()<<endl;
    cout<<p.get_ap_paterno()<<endl;
    cout<<p.get_ap_materno()<<endl;
    cout<<p.get_dia_nac()<<'/'<<p.get_mes_nac()<<'/'<<p.get_year()<<endl;
    return 0;
}
```

5.3.2 ACCESO A LOS MIEMBROS DE MANERA DIRECTA

```
#include<iostream>//Libreria para la impresion de datos
#include<iomanip>//Libreria para la presentacion de los datos al imprimirlos
using namespace std;
struct persona{
    //ATRIBUTOS
    string nombre;
    string ap_paterno;
    string ap_materno;
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
    //FUNCIONES MIEMBRO
    //Constructor que inicializa los datos en valor nulo
    persona(){}
    void mostrar_datos();
};
void persona::mostrar_datos(){
    cout<<setw(12)<<left<<"Nombre"<<": "<<nombre<<endl;
    cout<<setw(12)<<left<<"Ap. paterno"<<": "<<ap_paterno<<endl;
    cout<<setw(12)<<left<<"Ap. materno"<<": "<<ap_materno<<endl;
    cout<<setw(12)<<left<<"Fecha nac"<<": "<<d<<"/"<<m<<"/"<<a<<endl;
}
int main(){
    //creamos el objeto persona
    persona p;
    //Llenamos los datos del objeto persona
    p.nombre="Fermin";
    p.ap_paterno="Suarez";
    p.ap_materno="Gala";
    p.d=6;
    p.m=8;
    p.a=2003;
    cout<<"Mostramos los datos mediante la funcion mostrar_datos"<<endl;
    p.mostrar_datos();

    //Modificando los datos
    p.nombre="Juan";
    p.ap_paterno="Perez";
    cout<<"\nMostramos los datos mediante la funcion mostrar_datos"<<endl;
    p.mostrar_datos();

    cout<<"\nImprimiendo directamente los datos"<<endl;
    cout<<p.nombre<<endl;
```

```
    cout<<p.ap_paterno<<endl;
    cout<<p.ap_materno<<endl;
    cout<<p.d<<'/'<<p.m<<'/'<<p.a<<endl;
    return 0;
}
```

5.3.3 ACCESO A LOS MIEMBROS MEDIANTE PUNTEROS

```
#include<iostream>//Libreria para la impresion de datos
#include<iomanip>//Libreria para la presentacion de los datos al imprimirlos
using namespace std;
struct persona{
    //ATRIBUTOS
    string nombre;
    string ap_paterno;
    string ap_materno;
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año
    //FUNCIONES MIEMBRO
    persona(){} //Constructor que inicializa los datos en valor nulo
    void mostrar_datos();
};
void persona::mostrar_datos(){
    cout<<setw(12)<<left<<"Nombre"<<": "<<nombre<<endl;
    cout<<setw(12)<<left<<"Ap. paterno"<<": "<<ap_paterno<<endl;
    cout<<setw(12)<<left<<"Ap. materno"<<": "<<ap_materno<<endl;
    cout<<setw(12)<<left<<"Fecha nac"<<": "<<d<<"/"<<m<<"/"<<a<<endl;
}
int main(){
    persona *p=new persona();    //creamos el objeto persona
    //Llenamos los datos del objeto persona
    p->nombre="Fermin";
    p->ap_paterno="Suarez";
    p->ap_materno="Gala";
    p->d=6;
    p->m=8;
    p->a=2003;
    cout<<"Mostramos los datos mediante la funcion mostrar_datos"<<endl;
    p->mostrar_datos();
    //Modificando los datos
    p->nombre="Juan";
    p->ap_paterno="Perez";
    cout<<"\nMostramos los datos mediante la funcion mostrar_datos"<<endl;
    p->mostrar_datos();
    cout<<"\nImprimiendo directamente los datos"<<endl;
    cout<<p->nombre<<endl;
    cout<<p->ap_paterno<<endl;
    cout<<p->ap_materno<<endl;
    cout<<p->d<<'/'<<p->m<<'/'<<p->a<<endl;
    return 0;
}
```

5.4 HERENCIA

5.4.1 HERENCIA SIMPLE

```
#include<iostream>//Libreria para la impresion de datos
#include<iomanip>//Libreria para la presentacion de los datos al imprimirlos
using namespace std;
```

```
class persona{
    //ATRIBUTOS
    protected:
    string nombre;
    string ap_paterno;
    string ap_materno;
    //Fecha de nacimiento
    int d;//dia
    int m;//mes
    int a;//año

    //FUNCIONES MIEMBRO
    public:
    //Constructor que inicializa los datos en valor nulo
    persona(){}
    void mostrar_datos();
};
void persona::mostrar_datos(){
    cout<<setw(12)<<left<<"Nombre"<<": "<<nombre<<endl;
    cout<<setw(12)<<left<<"Ap. paterno"<<": "<<ap_paterno<<endl;
    cout<<setw(12)<<left<<"Ap. materno"<<": "<<ap_materno<<endl;
    cout<<setw(12)<<left<<"Fecha nac"<<": "<<d<<"/"<<m<<"/"<<a<<endl;
}
class estudiante:public persona{
    private:
    string codigo_estudiante;
    public:
    estudiante(string,string,string,int,int,int,string);
    string getcodigo();
};
estudiante::estudiante(string n,string p,string m,int x,int y,int z,string c){
    nombre=n;
    ap_paterno=p;
    ap_materno=m;
    d=x;
    m=y;
    a=z;
    codigo_estudiante=c;
}
string estudiante::getcodigo(){
    return codigo_estudiante;
}
int main(){
    estudiante E("Fermin","Suarez","Gala",6,8,2003,"SGF060803");
    //Utilizando la funcion heredada mostrar_datos de persona
    E.mostrar_datos();
    //Utilizando la funcion propia de estudiante getcodigo
    cout<<"Codigo: "<<E.getcodigo()<<endl;
      return 0;
}
```

5.4.2 HERENCIA MULTIPLE

```
#include<iostream>
using namespace std;
class area1{
    protected:
    double ancho;
    double largo;
       public:
    area1(){}
    void mostrar_area1();
```

```
};
void area1::mostrar_area1(){
    cout<<"Ancho: "<<ancho<<endl;
    cout<<"Largo: "<<largo<<endl;
}

class area2{
    protected:
    double alto;
    public:
    area2(){}
    void mostrar_area2();
};
void area2::mostrar_area2(){
    cout<<"Alto: "<<alto<<endl;
}
class rectangulo: public area1, public area2{
    private:
    double area;
    public:
    rectangulo(){}
    void set_datos(double,double,double);
    double get_area();
};
void rectangulo::set_datos(double a,double b,double c){
    ancho=a;
    largo=b;
    alto=c;
    area=ancho*largo*alto;
}
double rectangulo::get_area(){
    return area;
}
int main(){
    rectangulo R;
    R.set_datos(25.0, 17.3, 10.03);
    R.mostrar_area1();
    R.mostrar_area2();
    cout<<R.get_area()<<endl;
    return 0;
}
```

5.5 SOBRECARGA DE OPERADORES

5.5.1 OPERADORES UNARIOS ++ Y --

```
//Operadores fuera de la clase
#include<iostream>
using namespace std;
struct par{
    int primer, segundo;
    par(){}
    par(int x,int y){primer=x;segundo=y;}
    void mostrar();
};
void par::mostrar(){
    cout<<primer<<" "<<segundo<<endl;
}
//SOBRECARGA DEL OPERADOR UNARIO ++
par& operator ++(par &P){
    P.primer++;
    P.segundo++;
    return P;
```

```
}
//SOBRECARGA DEL OPERADOR UNARIO --
par& operator --(par &P){
    P.primer--;
    P.segundo--;
    return P;
}
int main(){
    par p1(23,45);
    ++p1;//Operador unario ++
    p1.mostrar();//Imprime: 24 46
    --p1;//Operador unario --
    p1.mostrar();//Imprime: 23 45
    return 0;
}
```

```
//Encapsulando los datos y los operadores estan dentro de la clase
#include<iostream>
using namespace std;
struct par{
    private:
    int primer, segundo;
    public:
    par(){}
    par(int x,int y){primer=x;segundo=y;}
    void mostrar();
    par& operator ++();
    par& operator --();
};
void par::mostrar(){
    cout<<primer<<" "<<segundo<<endl;
}
//SOBRECARGA DEL OPERADOR UNARIO ++
par& par:: operator ++(){
    this->primer++;
    this->segundo++;
}
par& par:: operator --(){
    this->primer--;
    this->segundo--;
}
int main(){
    par p(23,45);
    ++p;//Operador unario ++
    p.mostrar();//Imprime: 24 46
    --p;//Operador unario --
    p.mostrar();//Imprime: 23 45
    return 0;
}
```

5.5.2 OPERADORES BINARIOS +, -, * Y /

```
//Operadores fuera de la clase
#include<iostream>
using namespace std;
struct par{
    int primer, segundo;
    par(int x,int y){primer=x;segundo=y;}
    void mostrar();
};
void par:: mostrar(){
    cout<<primer<<" "<<segundo<<endl;
```

```
}
//SOBRECARGA DEL OPERADOR +
par& operator +(par &A,par &B){
    return *(new par(A.primer+B.primer,A.segundo+B.segundo));
}
//SOBRECARGA DEL OPERADOR -
par& operator -(par &A,par &B){
    return *(new par(A.primer-B.primer,A.segundo-B.segundo));
}
//SOBRECARGA DEL OPERADOR *
par& operator *(par &A,par &B){
    return *(new par(A.primer*B.primer,A.segundo*B.segundo));
}
//SOBRECARGA DEL OPERADOR /
par& operator /(par &A,par &B){
    return *(new par(A.primer/B.primer,A.segundo/B.segundo));
}
int main(){
    par p1(19,86);
    par p2(25,11);
    par p3=p1+p2;//Operador +
    par p4=p1-p2;//Operador -
    par p5=p1*p2;//Operador *
    par p6=p1/p2;//Operador *
    p3.mostrar();//Imprime: 44 97
    p4.mostrar();//Imprime: -6 75
    p5.mostrar();//Imprime: 475 946
    p6.mostrar();//Imprime: 0 7
    return 0;
}
```

```
//Encapsulando los datos y los operadores estan dentro de la clase
#include<iostream>
using namespace std;
class par{
    private:
    int primer, segundo;
    public:
    par(int x,int y){primer=x;segundo=y;}
    void mostrar();
    par& operator +(par &p);
    par& operator -(par &p);
    par& operator *(par &p);
    par& operator /(par &p);
};
void par:: mostrar(){
    cout<<primer<<" "<<segundo<<endl;
}
//SOBRECARGA DEL OPERADOR +
par& par:: operator +(par &P){
    this->primer+=P.primer;
    this->segundo+=P.segundo;
    return *this;
}
//SOBRECARGA DEL OPERADOR +
par& par:: operator -(par &P){
    this->primer-=P.primer;
    this->segundo-=P.segundo;
    return *this;
}

//SOBRECARGA DEL OPERADOR *
par& par:: operator *(par &P){
    this->primer*=P.primer;
```

```
    this->segundo*=P.segundo;
    return *this;
}

//SOBRECARGA DEL OPERADOR /
par& par:: operator /(par &P){
    this->primer/=P.primer;
    this->segundo/=P.segundo;
    return *this;
}
int main(){
    par p1(19,86);
    par p2(25,11);
    p1+p2;//Operador +
    p1.mostrar();//Imprime: 44 97
    p1-p2;//Operador -
    p1.mostrar();//Imprime: 19 86
    p1*p2;
    p1.mostrar();//Imprime: 475 946
    p1/p2;
    p1.mostrar();//Imprime: 19 86
    return 0;
}
```

5.5.3 OPERADORES DE COMPARACION ==, >, >=, <, <= Y !=

```
//Operadores fuera de la clase
#include<iostream>
using namespace std;
struct par{
    int primer, segundo;
    par(int x,int y){primer=x;segundo=y;}
};
//SOBRECARGA DEL OPERADOR ==
bool operator ==(par &A,par &B){return (A.primer==B.primer && A.segundo==B.segundo);}
//SOBRECARGA DEL OPERADOR !=
bool operator !=(par &A,par &B){return (A.primer!=B.primer || A.segundo!=B.segundo);}
//SOBRECARGA DEL OPERADOR >
bool operator >(par &A,par &B){
    if(A.primer>B.primer)
        return 1;
    if(A.primer==B.primer)
        return (A.segundo>B.segundo);
    return 0;
}
//SOBRECARGA DEL OPERADOR >=
bool operator >=(par &A,par &B){
    if(A.primer>B.primer)
        return 1;
    if(A.primer==B.primer)
        return (A.segundo>=B.segundo);
    return 0;
}
//SOBRECARGA DEL OPERADOR <
bool operator <(par &A,par &B){
    if(A.primer<B.primer)
        return 1;
    if(A.primer==B.primer)
        return (A.segundo<B.segundo);
    return 0;
}
//SOBRECARGA DEL OPERADOR <=
```

```
bool operator <=(par &A,par &B){
    if(A.primer<B.primer)
        return 1;
    if(A.primer==B.primer)
        return (A.segundo<=B.segundo);
    return 0;
}

int main(){
    par p1(25,11);
    par p2(25,11);
    par p3(27,13);
    par p4(25,10);
    if(p1==p2)
        cout<<"Los pares p1 y p2 son iguales"<<endl;
    else
        cout<<"Los pares p1 y p2 son diferentes"<<endl;
    //Imprime: Los pares p1 y p2 son iguales
    if(p1!=p3)
        cout<<"Los pares p1 y p3 son diferentes"<<endl;
    else
        cout<<"Los pares p1 y p3 son iguales"<<endl;
    //Imprime: Los pares p1 y p3 son diferentes
    if(p4>p1)
        cout<<"p4 es mayor que p1"<<endl;
    else
        cout<<"p4 NO es mayor que p1"<<endl;
    //Imprime: p4 NO es mayor que p1
    if(p4<p1)
        cout<<"p4 es menor que p1"<<endl;
    else
        cout<<"p4 NO es menor que p1"<<endl;
    //Imprime: p4 es menor que p1
    if(p3>=p1)
        cout<<"p3 es mayor o igual que p1"<<endl;
    else
        cout<<"p3 es menor que p1"<<endl;
    //Muestra: p3 es mayor o igual que p1
    if(p1<=p2)
        cout<<"p1 es menor o igual que p2"<<endl;
    else
        cout<<"p1 es mayor que p2"<<endl;
    //Imprime: p1 es menor o igual que p2
    return 0;
}
```

```
//Encapsulando los datos y los operadores estan dentro de la clase
#include<iostream>
using namespace std;
class par{
    private:
    int primer, segundo;
    public:
    par(int x,int y){primer=x;segundo=y;}
    bool operator ==(par &p);
    bool operator !=(par &p);
    bool operator >(par &p);
    bool operator >=(par &p);
    bool operator <(par &p);
    bool operator <=(par &p);
};
//SOBRECARGA DEL OPERADOR ==
bool par::operator ==(par &P){return (this->primer==P.primer && this-
>segundo==P.segundo);}
```

```
//SOBRECARGA DEL OPERADOR !=
bool par::operator !=(par &P){return (this->primer!=P.primer || this-
>segundo!=P.segundo);}
//SOBRECARGA DEL OPERADOR >
bool par::operator >(par &P){
    if(this->primer>P.primer)
        return 1;
    if(this->primer==P.primer)
        return (this->segundo>P.segundo);
    return 0;
}
//SOBRECARGA DEL OPERADOR >=
bool  par::operator >=(par &P){
    if(this->primer>P.primer)
        return 1;
    if(this->primer==P.primer)
        return (this->segundo>=P.segundo);
    return 0;
}
//SOBRECARGA DEL OPERADOR <
bool par::operator <(par &P){
    if(this->primer<P.primer)
        return 1;
    if(this->primer==P.primer)
        return (this->segundo<P.segundo);
    return 0;
}
//SOBRECARGA DEL OPERADOR <=
bool par::operator <=(par &P){
    if(this->primer<P.primer)
        return 1;
    if(this->primer==P.primer)
        return (this->segundo<=P.segundo);
    return 0;
}

int main(){
    par p1(25,11);
    par p2(25,11);
    par p3(27,13);
    par p4(25,10);
    if(p1==p2)
        cout<<"Los pares p1 y p2 son iguales"<<endl;
    else
        cout<<"Los pares p1 y p2 son diferentes"<<endl;
    //Imprime: Los pares p1 y p2 son iguales
    if(p1!=p3)
        cout<<"Los pares p1 y p3 son diferentes"<<endl;
    else
        cout<<"Los pares p1 y p3 son iguales"<<endl;
    //Imprime: Los pares p1 y p3 son diferentes
    if(p4>p1)
        cout<<"p4 es mayor que p1"<<endl;
    else
        cout<<"p4 NO es mayor que p1"<<endl;
    //Imprime: p4 NO es mayor que p1
    if(p4<p1)
        cout<<"p4 es menor que p1"<<endl;
    else
        cout<<"p4 NO es menor que p1"<<endl;
    //Imprime: p4 es menor que p1
    if(p3>=p1)
        cout<<"p3 es mayor o igual que p1"<<endl;
```

```
    else
        cout<<"p3 es menor que p1"<<endl;
    //Muestra: p3 es mayor o igual que p1
    if(p1<=p2)
        cout<<"p1 es menor o igual que p2"<<endl;
    else
        cout<<"p1 es mayor que p2"<<endl;
    //Imprime: p1 es menor o igual que p2
    return 0;
}
```

5.5.4 OPERADORES >>(ISTREAM) Y <<(OSTREAM)

```
#include<iostream>
using namespace std;
struct par{
    int primer, segundo;
    par(){}
    par(int x,int y){primer=x;segundo=y;}
};
istream& operator >>(istream& i,par &P){
    i>>P.primer>>P.segundo;
}
ostream& operator <<(ostream& o,par &P){
    o<<P.primer<<" "<<P.segundo;
}
int main(){
    par p;
    cin>>p;//Lectura del par mediante el teclado
    cout<<p<<endl;//Impresion del par
    return 0;
}
```

5.6 COMPOSICION

5.6.1 EJEMPLO SIMPLE

```
#include<iostream>
using namespace std;
class oficina{
    private:
    int nro;
    int capacidad;
    string detalle;
    public:
    oficina(){}
    void set_oficina(int,int,string);
    void mostrar();
};
void oficina::set_oficina(int n,int c,string d){
    nro=n;
    capacidad=c;
    detalle=d;
}
void oficina::mostrar(){
    cout<<"Nro: "<<nro<<endl;
    cout<<"Capacidad: "<<capacidad<<endl;
    cout<<"Detalle: "<<detalle<<endl;
}
class edificio{
    private:
```

```
    int nro_oficinas;
    oficina *vec;
    public:
    edificio(){}
    edificio(int);
    void crear_oficinas();
    int get_nro_oficinas();
    void mostrar_oficinas();
};
edificio::edificio(int n){
    nro_oficinas=n+1;
    vec = new oficina[nro_oficinas];
}
void edificio::crear_oficinas(){
    int nr,c;
    string det;
    for(int i=0;i<nro_oficinas;i++){
        //Introduciendo datos
        cin>>nr>>c>>det;
        oficina f;
        f.set_oficina(nr,c,det);
        vec[i]=f;
    }
}
int edificio::get_nro_oficinas(){
    return nro_oficinas;
}
void edificio::mostrar_oficinas(){
    for(int i=0;i<nro_oficinas;i++){
        vec[i].mostrar();
        cout<<"*******"<<endl;
    }
}
int main(){
    edificio E(7);
    E.crear_oficinas();
    cout<<"Nro de oficinas: "<<E.get_nro_oficinas()<<endl;
    cout<<"LISTA DE OFICINAS"<<endl;
    E.mostrar_oficinas();
    return 0;
}
```

6 CONTENEDORES STL DE C++

6.1 VECTORS (VECTORES)

6.1.1 CONSTRUCTORES DE VECTOR

```
#include<iostream>//Librería para la entrada y salida de datos
#include<vector>//Librería para el manejo de vectores
using namespace std;
//Procedimiento para mostrar el contenido de un vector
void mostrar(vector<int> vec){
    for(int i=0;i<vec.size();i++)
        cout<<vec[i]<<" ";
    cout<<endl;
}
int main(){
    //construye un vector vacío
    vector<int>primero;
    //construye un vector con 5 elementos de valor 0
    vector<int>segundo(5);
    //construye un vector con 5 elementos de valor 7
    vector<int>tercero(5,7);
    //construye un vector con los elementos de un vector estatico
    int x[]={25,7,11,2,16};
    vector<int>cuarto(x,x+5);
    //construye un quinto vector con los elementos del segundo
    vector<int>quinto(segundo.begin(),segundo.end());

    //Mostramos resultados mediante mostrar:
    mostrar(primero);//No imprime nada
    mostrar(segundo);//Imprime: 0 0 0 0 0
    mostrar(tercero);//Imprime: 7 7 7 7 7
    mostrar(cuarto);//Imprime: 25 7 11 2 6
    mostrar(quinto);//Imprime: 0 0 0 0 0
    return 0;
}
```

`vector<tipo_dato>V`	Crea un vector vacío.
`vector<int>V(m)`	Donde m es un entero positivo. Crea un vector con m elementos, en el caso de este vector de enteros, las m casillas se inicializaran con 0.
`vector<tipo_dato>V(m,dato)`	Donde m es un entero. Crea un vector con m copias del dato.
`vector<tipo_dato>V(a, b)`	Donde a y b son iteradores. Crea un vector V con los elementos de otro contenedor STL desde la posición a hasta la posición b.
`x[m]` `vector<int>V(x+i,x+j)`	Donde i, j son enteros no negativos. Crea un vector vec con los elementos de x desde la posición i hasta la posición j.
`v.~vector()`	Destruimos el vector denominado vec.

6.1.2 COMPARACION DE VECTORES

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
   int x[]={25,7,11,2,19,86,14};
   int y[]={26,7,11,2,19,86,14};
   vector<int>vector1(x,x+7);
   vector<int>vector2(x,x+7);
   vector<int>vector3(y,y+7);
```

```
   cout<<"OPERADOR =="<<endl;
   if(vector1==vector2)
        cout<<"Los vectores 1 y 2 son iguales"<<endl;
   else
        cout<<"Los vectores 1 y 2 son diferentes"<<endl;
    //Muestra: Los vectores 1 y 2 son iguales

    cout<<"OPERADOR !="<<endl;
    if(vector1!=vector3)
        cout<<"Los vectores 1 y 3 son diferentes"<<endl;
    else
        cout<<"Los vectores 1 y 3 son iguales"<<endl;
    //Muestra: Los vectores 1 y 3 son diferentes

    cout<<"OPERADOR >"<<endl;
    if(vector1>vector3)
        cout<<"El vector 1 es mayor al vector 3"<<endl;
    else
        cout<<"El vector 1 NO es mayor al vector 3"<<endl;
   //Muestra: El vector 1 NO es mayor al vector 3

    cout<<"OPERADOR <"<<endl;
    if(vector1<vector3)
       cout<<"El vector 1 es menor al vector 3"<<endl;
    else
        cout<<"El vector 1 NO es menor al vector 3"<<endl;
    //Muestra: El vector 1 es menor al vector 3
    return 0;
}
```

==	Esta comparación devuelve **true** (verdadrero) si los vectores son iguales, es decir que los elementos de ambos vectores son iguales. En caso de que los vectores sean diferentes la comparación devolverá **false** (falso).
!=	Esta comparación devuelve **true** (verdadrero) si los vectores son diferentes, es decir que los elementos de ambos vectores son diferentes. En caso de que los vectores sean iguales la comparación devolverá **false** (falso).
>	Esta comparación devuelve **true** (verdadrero) si el vector vec1 es mayor lexicográficamente al vector vec2. Caso contrario devuelve **false** (falso).
<	Esta comparación devuelve **true** (verdadrero) si el vector vec1 es menor lexicográficamente al vector vec2. Caso contrario devuelve **false** (falso).
>=	Esta comparación devuelve **true** (verdadrero) si el vector vec1 es mayor o igual lexicográficamente al vector vec2. Caso contrario devuelve **false** (falso).
<=	Esta comparación devuelve **true** (verdadrero) si el vector vec1 es menor lexicográficamente al vector vec2. Caso contrario devuelve **false** (falso).

6.1.3 INSERCCION DE ELEMENTOS EN EL VECTOR

```
#include<iostream>
#include<vector>
using namespace std;
void mostrar_vector(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    vector<int>A,B,C,D,E;
    cout<<"FUNCION PUSH_BACK"<<endl;
    //Introducimos los elementos: 12 51 27 8
    A.push_back(12);
```

```
    A.push_back(51);
    A.push_back(27);
    A.push_back(8);
    cout<<"Vector A: ";
    mostrar_vector(A);//Muestra: 12 51 27 8

    cout<<"\nFUNCION INSERT"<<endl;
    //Insertando el numero 20 en la primera posicion del vector A
    A.insert(A.begin(),20);
    cout<<"Vector A: ";
    mostrar_vector(A);//Muestra: 20 12 51 27 8
    //Insertando el numero 13 en la posicion final del vector A
    A.insert(A.end(),13);
    cout<<"Vector A: ";
    mostrar_vector(A);//Muestra: 20 12 51 27 8 13
    //Insertamos el numero 39 en la tercera posicion del vector A
    A.insert(A.begin()+2,39);
    cout<<"Vector A: ";
    mostrar_vector(A);//Muestra: 20 12 39 51 27 8 13
    //Insertamos el numero 10 en la quinta posicion del vector A
    A.insert(A.end()-2,10);
    cout<<"Vector A: ";
    mostrar_vector(A);//Muestra: 20 12 39 51 27 10 8 13
    //Insertamos los elementos del vector A en el vector B
    B.insert(B.begin(),A.begin(),A.end());
    cout<<"Vector B: ";
    mostrar_vector(B);//Muestra: 20 12 51 27 8 13
    //Insertamos los elementos de un vector estatico en el vector C
    //Vector estatico
    int x[]={25,7,11,2,19,10};
    C.insert(C.begin(),x,x+6);
    cout<<"Vector C: ";
    mostrar_vector(C);//Muestra: 25 7 11 2 19 10
    //Insertamos los tres primeros elementos del vector A en el vector D
    D.insert(D.begin(),A.begin(),A.begin()+3);
    cout<<"Vector D: ";
    mostrar_vector(D);//Muestra: 20 12 39
    //Insertamos los elementos 2do, 3ro y 4to del vector C en la segunda
    //posicion del vector D
    D.insert(D.begin()+1,C.begin()+1,C.begin()+4);
    cout<<"Vector D: ";
    mostrar_vector(D);//Muestra: 20 7 11 2 12 39
    return 0;
}
```

`push_back(dato)`	Insertar un dato al final del vector.
`vec.insert(a,dato)`	Dónde a es un iterador del vector vec. Inserta un dato en la posición a.
`A.insert(itA, itB1, itB2)`	Dónde itA1 es un iterador del vector A, itB1 e itB2 son iteradores de otro contenedor B. La función insert inserta en la posición itA del vector A, los datos del vector B a partir de la posición itB1 hasta la posición itB2.

6.1.4 ASIGNACION E INTERCAMBIO DE VECTORES

```
#include<iostream>
#include<vector>
using namespace std;
void mostrar_vector(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
```

```
}
int main(){
    vector<int>vector1,vector2,vector3,vector4;
    int x[]={25,7,11,2,19,86,14};
    int y[]={26,7,11,2,19,86,14};

    cout<<"FUNCION ASSIGN"<<endl;
    //Asignando los elementos del array x al vector 1
    vector1.assign(x,x+7);
    cout<<"vector1: ";
    mostrar_vector(vector1);//Muestra: 25 7 11 2 19 86 14
    //Asignando los 3 primeros elementos del array x al vector 2
    vector2.assign(x,x+3);
    cout<<"vector2: ";
    mostrar_vector(vector2);//Muestra: 25 7 11
    //Asignando 4 elementos del 2do al 5to elemento del array x al vector 3
    vector3.assign(x+1,x+5);
    cout<<"vector3: ";
    mostrar_vector(vector3);//Muestra: 7 11 2 19
    //Asignando los elementos del vector 1 al vector 4
    vector4.assign(vector1.begin(),vector1.end());
    cout<<"vector4: ";
    mostrar_vector(vector4);//Muestra: 25 7 11 2 19 86 14
    //Asignando los tres primeros elementos del vector 1 al vector 4
    vector4.assign(vector1.begin(),vector1.begin()+3);
    cout<<"vector4: ";
    mostrar_vector(vector4);//Muestra: 25 7 11
    //Asignando los tres ultimos elementos del vector 1 al vector 4
    vector4.assign(vector1.end()-3,vector1.end());
    cout<<"vector4: ";
    mostrar_vector(vector4);//Muestra: 19 86 14
    //Asignando 4 elementos del 2do al 5to elemento del vector 1 al vector 4
    vector4.assign(vector1.begin()+1,vector1.begin()+5);
    cout<<"vector4: ";
    mostrar_vector(vector4);//Muestra: 7 11 2 19

    cout<<"\nOPERADOR DE ASSIGNACION ="<<endl;
    //Los elementos del vector 4
    //Seran introducidos al vector 2
    vector2=vector4;
    cout<<"Vector2: ";
    mostrar_vector(vector2);//Muestra: 7 11 2 19

    cout<<"\nFUNCION SWAP"<<endl;
    cout<<"vector1: ";
    mostrar_vector(vector1);//Muestra: 25 7 11 2 19 86 14
    cout<<"vector4: ";
    mostrar_vector(vector4);//Muestra: 7 11 2 19
    cout<<"Intercambio de elementos del vector1 con elementos del vector4"<<endl;
    vector1.swap(vector4);
    cout<<"vector1: ";
    mostrar_vector(vector1);//Muestra: 7 11 2 19
    cout<<"vector4: ";
    mostrar_vector(vector4);//Muestra: 25 7 11 2 19 86 14
    return 0;
}
```

`A.assign(itB1, itB2)`	Dónde itB1 e itB2 son iteradores de otro contenedor B. Asigna los valores de B a A, a partir de la posición itB1 hasta la posición itB2.
`vec1=vec2`	Dónde a, b son enteros. Asigna los valores de vec2 a vec1.
`vec1.swap(vec2)`	La función swap intercambia los elementos del vector vec1 con los elementos del vector 2.

6.1.5 ELIMINACION DE DATOS DEL VECTOR

```
#include<iostream>
#include<vector>
using namespace std;
void mostrar_vector(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    vector<int>vec;
    //Llenando el vector con numeros del 1 al 10
    for(int i=1;i<=15;i++)
        vec.push_back(i);
    mostrar_vector(vec);//Muestra: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

    cout<<"\nFUNCION POP BACK"<<endl;
    cout<<"Borrando los 3 ultimos elementos del vector"<<endl;
    vec.pop_back();
    vec.pop_back();
    vec.pop_back();
    mostrar_vector(vec);//Muestra: 1 2 3 4 5 6 7 8 9 10 11 12

    cout<<"\nFUNCION ERASE"<<endl;
    cout<<"Borrando el cuarto elemento del vector"<<endl;
    vec.erase(vec.begin()+3);
    mostrar_vector(vec);//Muestra: 1 2 3 5 6 7 8 9 10 11 12
    cout<<"Borrando los elementos 2do, 3ro, 4to y 5to del vector"<<endl;
    vec.erase(vec.begin()+1,vec.begin()+5);
    mostrar_vector(vec);//Muestra: 1 7 8 9 10 11 12

    cout<<"\nFUNCION CLEAR"<<endl;
    cout<<"Borrando todos los elementos vector"<<endl;
    vec.clear();
    mostrar_vector(vec);//No muestra nada
    return 0;
}
```

`pop_back()`	La función pop_back() elimina el dato que se encuentra en la posición final.
`vec.erase(it)`	Dónde it es ub iterador de vector. La función erase elimina el dato que se encuentra en la posición del iterador it.
`vec.erase(it1, it2)`	Dónde it1 e it2 son iteradores de vector. La función erase elimina los dato del vector que se encuentran desde la posición de it1 hasta la posición de it2.
`clear()`	Elimina todos los datos del vector.

6.1.6 CAPACIDAD Y TAMAÑO DEL VECTOR

```
#include<iostream>
#include<vector>
using namespace std;
void mostrar(vector<int> &vec){
    for(int i=0;i<vec.size();i++)
        cout<<vec[i]<<" ";
    cout<<endl;
}
int main(){
    vector<int>vec;
    for(int i=1;i<=10;i++)
        vec.push_back(i);
```

```
    cout<<"Contenido del vector: ";
    mostrar(vec);//Muestra: 1 2 3 4 5 6 7 8 9 10

    cout<<"\nLONGITUDES DEL VECTOR:"<<endl;
    cout<<"size: "<<vec.size()<<endl;//Muestra: 10
    cout<<"capacity: "<<vec.capacity()<<endl;//Muestra: 16
    cout<<"max_size: "<<vec.max_size()<<endl;//Muestra: 1073741823

    cout<<"\nFUNCION RESIZE:"<<endl;
    cout<<"Modificamos la longitud de 10 a 7"<<endl;
    vec.resize(7);
    cout<<"Contenido del vector: ";
    mostrar(vec);//Muestra: 1 2 3 4 5 6 7
    cout<<"Modificamos la longitud de 7 a 12"<<endl;
    vec.resize(12);
    cout<<"Contenido del vector: ";
    mostrar(vec);//Muestra: 1 2 3 4 5 6 7 0 0 0 0 0
    cout<<"Modificamos la longitud de 12 a 15 con copias del numero 27"<<endl;
    vec.resize(15,27);
    cout<<"Contenido del vector: ";
    mostrar(vec);//Muestra: 1 2 3 4 5 6 7 0 0 0 0 0 27 27 27

    cout<<"\nFUNCION EMPTY:"<<endl;
    cout<<"Eliminando todos los elementos del vector"<<endl;
    while(!vec.empty())//mientras el vector no este vacio
        vec.pop_back();//Elimando el ultimo elemento del vector
    cout<<"Contenido del vector: ";
    mostrar(vec);//Muestra:
    return 0;
}
```

`size()`	Devuelve un entero que representa la longitud del vector, es decir el número de datos del vector.
`capacity()`	Devuelve un entero que representa la capacidad del vector en potencias de 2.
`max_size()`	Devuelve un entero que representa el tamaño máximo del vector: 1073741823.
`resize(m)`	Donde m es un entero positivo, modifica el tamaño del vector a m. Si m es menor al tamaño actual del vector entonces se conservan los primeros m valores. Si m es mayor al tamaño actual del vector entonces se conservan los datos del vector añadiéndole elementos vacíos, como ejemplo: si el vector es de números entonces llenara los datos restantes con 0. Si el vector es de cadenas entonces llenara los datos restantes con cadenas vacías.
`resize(m,elem)`	Donde m es un entero positivo y elem es un elemento, modifica el tamaño del vector a m. Si m es menor al tamaño actual del vector entonces se conservan los primeros m valores. Si m es mayor al tamaño actual del vector entonces se conservan los datos del vector añadiéndole elementos elem.
`empty()`	Devuelve **true** (verdadero) si el vector está vacío, caso contrario devuelve **false** (falso).

6.1.7 ACCESO A LOS ELEMENTOS DEL VECTOR

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
    vector<int>vec;
    //Introducimos los numeros: 12 8 20 13 9
    vec.push_back(12);
    vec.push_back(8);
    vec.push_back(20);
    vec.push_back(13);
```

```
    vec.push_back(19);

    cout<<"OPERADORES [] Y AT"<<endl;
    //Operador []
    for(int i=0;i<vec.size();i++)
        cout<<vec[i]<<" ";//Muestra: 12 8 20 13 9
    cout<<endl;
    //Operador at
    for(int i=0;i<vec.size();i++)
        cout<<vec.at(i)<<" ";//Muestra: 12 8 20 13 9
    cout<<endl;

    cout<<"\nOPERADOR FRONT"<<endl;
    cout<<"el primer elemento del vector es: "<<vec.front()<<endl;
    cout<<"\nOPERADOR BACK"<<endl;
    cout<<"el ultimo elemento del vector es: "<<vec.back()<<endl;

    //Creamos un vector de cadenas
    vector<string>vec_cad;
    vec_cad.push_back("Jose");
    vec_cad.push_back("Fermin");
    vec_cad.push_back("Juan");
    vec_cad.push_back("Rolo");
    //Creamos un puntero
    string *p=vec_cad.data();
    cout<<"\nOPERADOR data"<<endl;
    cout<<"Mostramos los elementos de un vector de cadenas:"<<endl;
    for(int i=0;i<vec_cad.size();i++){
        cout<<"elemento "<<i<<" "<<*p<<endl;
        p++;
    }
    /*
    elemento 0 Jose
    elemento 1 Fermin
    elemento 2 Juan
    elemento 3 Rolo
    */
    return 0;
}
```

`[i]`	Donde i es un entero no negativo. Accede al i-esimo elemento del vector.
`at(i)`	Donde i es un entero no negativo. Accede al i-esimo elemento del vector.
`front()`	Accede al primer elemento del vector
`back()`	Accede al último elemento del vector
`data()`	Accede a cierto elemento de un vector mediante un puntero.

6.1.8 ITERADORES DEL VECTOR

```
#include<iostream>
#include<vector>
using namespace std;
void mostrar(vector<string> &vec){
    for(int i=0;i<vec.size();i++)
        cout<<vec[i]<<" ";
    cout<<endl;
}
int main(){
    vector<string>vec;
    vec.push_back("Jose");
```

```
    vec.push_back("Fermin");
    vec.push_back("Rolo");
    vec.push_back("Roger");
    vec.push_back("Juan");

    cout<<"ITERADORES"<<endl;
    vector<string>::iterator it;
    for(it=vec.begin();it!=vec.end();it++)
        cout<<*it<<" ";//Muestra: Jose Fermin Rolo Roger Juan
    cout<<endl;

    cout<<"\nITERADORES DE REVERSA"<<endl;
    vector<string>::reverse_iterator rit;
    for(rit=vec.rbegin();rit!=vec.rend();rit++)
        cout<<*rit<<" ";//Muestra: Juan Roger Rolo Fermin Jose
    cout<<endl;

    cout<<"\nIntroduciendo datos mediante la funcion: INSERT"<<endl;
    it=vec.begin();
    it=it+2;//avanzamos 2 posiciones
    cout<<"Introducimos la cadena mundo en la posicion 2"<<endl;
    vec.insert(it,"mundo");
    mostrar(vec);//Muestra: Jose Fermin mundo Rolo Roger Juan

    cout<<"\nEliminando datos mediante la funcion: ERASE"<<endl;
    cout<<"Eliminamos la cadena que se encuentra en la posicion 4"<<endl;
    it=vec.begin()+4;//el iterator apunta en la posicion 4
    vec.erase(it);
    mostrar(vec);//Muestra: Jose Fermin mundo Rolo Juan
    cout<<"Eliminamos los elementos desde la posicion 2 hasta la posicion 4"<<endl;
    vector<string>:: iterator a,b;
    a=vec.begin()+2;
    b=vec.begin()+4;
    vec.erase(a,b);
    mostrar(vec);//Muestra: Jose Fermin Juan
    return 0;
}
```

`vector<tipo_dato>:: iterator it;`	Crea un iterator de vector denominado it.
`vector<tipo_dato>:: reverse_iterator rit;`	Crea un iterator reverso de vector denominado rit.
`begin()`	Iterador que apunta al principio del vector.
`end()`	Iterador que apunta al final del vector.
`rbegin()`	Iterador reverso que apunta al principio del vector invertido.
`rend()`	Iterador reverso que apunta al final del vector invertido.

6.1.9 VECTOR BOOL

```
//Programa que genera primos con la criba de Eratóstenes
#include<iostream>
#include<vector>
#define max 1000000
using namespace std;
vector<bool>vec(max,true);
void criba(){
    vec[0]=vec[1]=false;
    for(int i=2;i<max;i++){
        if(vec[i]){
            for(int j=(i<<1);j<max;j=j+i)
```

```
                vec[j]=false;
        }
    }
}
int main(){
    criba();
    cout<<"Muestra los 100 primeros numeros primos"<<endl;
    int i=0;
    int j=0;
    while(j<100){
        if(vec[i]){
            cout<<i<<" ";
            j++;
        }
        i++;
    }
    cout<<endl;
    cout<<"\nInvertimos los valores con FLIP"<<endl;
    cout<<"Muestra los 100 primeros numeros no primos"<<endl;
    vec.flip();
    i=j=0;
    while(j<100){
        if(vec[i]){
            cout<<i<<" ";
            j++;
        }
        i++;
    }
    cout<<endl;
    return 0;
}
```

6.2 STACKS (PILAS)

6.2.1 CONSTRUCTORES Y FUNCIONES DE LA PILA

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<stack>//Libreria para el manejo de pilas
#include<vector>//Libreria para el manejo de vectores
using namespace std;
//Metodo para leer datos desde teclado
void llenar_pila(stack <int> &P,int n){
    int x;
    for(int i=1;i<=n;i++){
        cin>>x; //Lectura de datos
        P.push(x);//Introduciendo los datos leidos a la pila
    }
}
//Metodo para vaciar una pila origen a una pila destino.
//Este método nos sirve para salvar los datos después de usar el método mostrar
template <class R>
void vaciar_pila(R &P1,R &P2){//Se utiliza punteros para guarder modificaciones
    while(!P1.empty()){//Mientras P1 no este vacia
        //Obtiene el ultimo elemento de la pila P1 y lo introduce en la pila P2
        P2.push(P1.top());
        P1.pop();//Elimina el ultimo dato de la pila P1
    }
}
//Metodo para imprimir los datos de una pila de enteros
void mostrar_pila(stack<int> P){
    stack<int>aux; //Crea una pila auxiliar
    while(!P.empty()){//Mientras la pila no esta vacia
        cout<<P.top()<<" ";
```

```
        //Obtiene el ultimo elemento de la pila y lo introduce en la pila aux
        aux.push(P.top());
        P.pop();//Elimina el ultimo elemento de la pila
    }
    cout<<endl;
    vaciar_pila(aux,P); //vaciamos la pila aux para devolver el contenido a pila P
}
//Metodo para imprimir los datos de una pila de vectores de enteros
void mostrar_pila(stack<int,vector<int> > &P){
    while(!P.empty()){//Mientras la pila no esta vacia
        cout<<P.top()<<" ";//Obtiene el ultimo elemento de la pila y lo imprime
        P.pop();//Elimina el ultimo elemento de la pila
    }
    cout<<endl;
}
int main(){
    //Creamos un vector y lo cargamos de datos
    vector<int>vec;
    vec.push_back(11);
    vec.push_back(9);
    vec.push_back(2);
    vec.push_back(22);
    vec.push_back(13);

    //crea una pila vacia
    stack<int>pila1;
    //crea una pila con los elementos del vector
    stack<int, vector<int> >pila2(vec);
    //llenamos la pila1 con 7 datos mediante el procedimiento llenar
    cout<<"Introduzca 7 numeros enteros"<<endl;
    llenar_pila(pila1,7);
    //creamos la pila3 con los elementos de la pila1
    stack<int>pila3(pila1);

    //Mostramos los datos
    cout<<"\nPila1: ";
    mostrar_pila(pila1);//Imprime los datos leidos en orden inverso
    cout<<"\nPila2: ";
    mostrar_pila(pila2);//Imprime: 13 22 2 9 11
    cout<<"\nPila3: ";
    mostrar_pila(pila3);//Imprime los datos leidos en orden inverso

    //Tamaño de la pila
    cout<<"Longitud de la pila1: "<<pila1.size()<<endl;//7
    cout<<"Longitud de la pila2: "<<pila2.size()<<endl;
    return 0;
}
```

`stack<tipo_dato>pila`	Crea una pila vacía.
`stack<int, vector<int> >pila(vec)`	Crea una pila que contiene los elementos de un vector vec.
`stack<tipo_dato>pila2(pila1)`	Crea una pila2 que contiene los elementos de pila1.
`push(dato)`	Introduce un dato en la pila.
`top()`	Accede al último elemento de la pila.
`pop()`	Elimina el último elemento de la pila.
`size()`	Esta función devuelve un numero entero no negativo que representa la longitud (numero de elementos) de la pila.
`empty()`	Devuelve **true** (verdadero) si la pila está vacía, caso contrario devuelve **false** (falso).

6.3 QUEUES (COLAS) Y PRIORITY_QUEUES(COLAS DE PRIORIDAD)

6.3.1 CONSTRUCTORES Y FUNCIONES DE LA COLA

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<queue>//Libreria para el manejo de colas
using namespace std;
//Metodo para imprimir los datos que contiene una cola
void mostrar_cola(queue<int> X){
    int n=X.size();//Obtiene el tamaño de la cola
    int dato;
    for(int i=0;i<n;i++){
        dato=X.front();//Obtiene el primer elemento de la cola
        cout<<dato<<" ";//Imprime el dato
        X.pop() //Elimina el primer elemento de la cola
        X.push(dato);//Introduce de nuevo al final de la cola
    }
    cout<<endl;
}
int main(){
    //Crea una cola vacia
    queue<int>cola1;
    //Introducimos los datos
    cola1.push(14);
    cola1.push(27);
    cola1.push(9);
    cola1.push(5);
    //creamos la cola2 con los elementos de la cola1
    queue<int>cola2(cola1);
     //Mostramos los datos
    cout<<"Cola1: ";
    mostrar_cola(cola1);//Imprime: 14 27 9 5
    cout<<"Cola2: ";
    while(!cola2.empty()){//Mientras la cola2 no este vacia
        cout<<cola2.front()<<" ";//Obtiene el primer elemento de la cola y lo imprime
        cola2.pop();//Eliminando el primer elemento de la cola
    }//Imprime: 14 27 9 5
    cout<<endl;

    //Tamaño de la cola
    cout<<"Longitud de la cola1: "<<cola1.size()<<endl;//4
    cout<<"Longitud de la cola2: "<<cola2.size()<<endl;//0
    return 0;
}
```

`queue<tipo_dato>cola`	Crea una cola vacía.
`queue<tipo_dato>cola2(cola1)`	Crea una cola2 que contiene los elementos de cola1.
`push(dato)`	Introduce un dato en la cola.
`front()`	Accede al primer elemento de la cola.
`pop()`	Elimina el primer elemento de la cola.
`size()`	Esta función devuelve un numero entero que representa la longitud (numero de elementos) de la cola.
`empty()`	Devuelve **true** (verdadero) si la cola está vacía, caso contrario devuelve **false** (falso).

6.3.2 CONSTRUCTORES Y FUNCIONES DE LA COLA DE PRIORIDADES

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<queue>//Libreria para el manejo de colas
```

```
#include<vector>//Libreria para el manejo de vectores
using namespace std;
void mostrar_colap(priority_queue<int> A){
    while(!A.empty()){//Mientras que la cola no este vacia
        cout<<A.top()<<" ";//Obtiene el primer elemento de la cola y lo imprime
        A.pop();//Elimina el primer elemento de la cola
    }
    cout<<endl;
}
int main(){
    //Crea una cola de prioridades vacia
    priority_queue<int>colap1;
    //Introducimos los datos en colap1
    colap1.push(14);
    colap1.push(27);
    colap1.push(9);
    colap1.push(5);
    //crea la colap2 con los elementos de la colap1
    priority_queue<int>colap2(colap1);
    //Creamos la colap3 con los elemento de un array[]
    int x[]={25,11,7,2,16};
    priority_queue<int>colap3(x,x+5);
    //Creamos la colap4 con los elemento de un vector
    vector<int>v(x,x+5);
    priority_queue<int>colap4(v.begin(),v.end());
    //Tamaño de las colas
    cout<<"Longitud de la colap1: "<<colap1.size()<<endl;//Imprime: 4
    cout<<"Longitud de la colap3: "<<colap3.size()<<endl;//Imprime: 5
    //Mostrando los datos
    cout<<"colap1: ";
    mostrar_colap(colap1);//Imprime: 27 14 9 5
    cout<<"colap2: ";
    mostrar_colap(colap2);//Imprime: 27 14 9 5
    cout<<"colap3: ";
    mostrar_colap(colap3);//Imprime: 25 16 11 2
    cout<<"colap4: ";
    mostrar_colap(colap4);//Imprime: 25 16 11 2
    return 0;
}
```

`priority_queue<tipo_dato>colap`	Crea una cola de prioridades vacía.
`priority_<tipo_dato>CP2(CP1)`	Crea una cola de prioridades CP2 que contiene los elementos de la cola de prioridades CP1.
`x[m]` `priority_queue<int>CP(x+i,x+j)`	Donde i, j son enteros no negativos. Crea una cola de prioridades CP con los elementos de x desde la posición i hasta la posición j.
`priority_queue<int>CP(a, b)`	Donde a y b son iteradores. Crea una cola de prioridades CP con los elementos de otro contenedor STL desde la posición a hasta la posición n.
`push(dato)`	Introduce un dato en la cola de prioridades ordenandolos descendentemente.
`top()`	Accede al primer elemento de la cola, es decir que accede al mayor de los elementos.
`pop()`	Elimina el primer elemento de la cola, es decir que elimina al mayor de los elementos.
`size()`	Esta función devuelve un numero entero que representa la longitud (numero de elementos) de la cola.
`empty()`	Devuelve **true** (verdadero) si la cola está vacía, caso contrario devuelve **false** (falso).

6.3.3 COLA DE PRIORIDADES CON CLASES U OBJETOS

```
#include<iostream>
#include<queue>
using namespace std;
struct estudiante{
    string paterno;
    string materno;
    string nombre;
    string ci;
    int id;
    estudiante(){}
    estudiante(string,string,string,string,int);
    void mostrar();
};
estudiante::estudiante(string pat,string mat,string nom,string c,int i){
    paterno=pat;
    materno=mat;
    nombre=nom;
    ci=c;
    id=i;
}
void estudiante ::mostrar(){
    cout<<id<<" "<<paterno<<" "<<materno<<" "<<nombre<<" "<<ci<<endl;
}
bool operator <(estudiante est1,estudiante est2){
    return est1.id>est2.id;
}
bool operator >(estudiante est1,estudiante est2){
    return est1.id<est2.id;
}
int main(){
  priority_queue<estudiante>cpe1;
  cpe1.push(estudiante("Espejo","Cuba","Jose Gonzalo","4863359 L.P.",21));
  cpe1.push(estudiante("De la Quinata","Illanes","Mauricio Mijail","6483259 L.P.",23));
  cpe1.push(estudiante("Choquehuanca","Villca","ROger","4565234 L.P.",22));
  cpe1.push(estudiante("Crispin","Machicado","Omar Gonzalo","4737778 L.P.",27));
  cpe1.push(estudiante("Espinoza","Argollo","Branimir Fernando","6783880 L.P.",25));
  priority_queue<estudiante,vector<estudiante>,greater<vector<estudiante>::value_type> >cpe2;
  //priority_queue<estudiante,vector<estudiante>,less<vector<estudiante>::value_type> >cpe2;
  cpe2.push(estudiante("Espejo","Cuba","Jose Gonzalo","4863359 L.P.",21));
  cpe2.push(estudiante("De la Quinata","Illanes","Mauricio Mijail","6483259 L.P.",23));
  cpe2.push(estudiante("Choquehuanca","Villca","ROger","4565234 L.P.",22));
  cpe2.push(estudiante("Crispin","Machicado","Omar Gonzalo","4737778 L.P.",27));
  cpe2.push(estudiante("Espinoza","Argollo","Branimir Fernando","6783880 L.P.",25));
  estudiante e;
  cout<<"Mostrando el contenido de la cola de prioridades cpe1"<<endl;
  while(!cpe1.empty()){
       e=cpe1.top();
       e.mostrar();
       cpe1.pop();
  }
  cout<<"\nMostrando el contenido de la cola de prioridades cpe2"<<endl;
  while(!cpe2.empty()){
       e=cpe2.top();
       e.mostrar();
       cpe2.pop();
  }
  return 0;
}
```

```
//El programa muestra:
Mostrando el contenido de la cola de prioridades cpe1
21 Espejo Cuba Jose Gonzalo 4863359 L.P.
22 Choquehuanca Villca ROger 4565234 L.P.
23 De la Quinata Illanes Mauricio Mijail 6483259 L.P.
25 Espinoza Argollo Branimir Fernando 6783880 L.P.
27 Crispin Machicado Omar Gonzalo 4737778 L.P.

Mostrando el contenido de la cola de prioridades cpe2
```

```
27 Crispin Machicado Omar Gonzalo 4737778 L.P.
25 Espinoza Argollo Branimir Fernando 6783880 L.P.
23 De la Quinata Illanes Mauricio Mijail 6483259 L.P.
22 Choquehuanca Villca ROger 4565234 L.P.
21 Espejo Cuba Jose Gonzalo 4863359 L.P.
```

6.4 DEQUES (BICOLAS)

6.4.1 CONSTRUCTORES Y DESTRUCTORES

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<deque>//Libreria para el manejo de bicolas
using namespace std;
void mostrar_bicola(deque<int> &bc){
    for(int i=0;i<bc.size();i++)
        cout<<bc[i]<<" ";
    cout<<endl;
}
int main(){
    //crea una bicola vacia
    deque<int>bicola1;
    //crea una bicola con cuatro elementos nulos
    deque<int>bicola2(4);
    //crea una bicola con cuatro copias del numero 27
    deque<int>bicola3(4,27);
    //crea una bicola con los elementos de la tercera bicola
    deque<int>bicola4(bicola3.begin(),bicola3.end());
    //crea una bicola con los dos elementos de la tercera bicola
    deque<int>bicola5(bicola3.begin(),bicola3.begin()+2);
    //Creamos un vector con cinco elementos
    int vec[]={25,11,16,7,2};
    deque<int>bicola6(vec,vec+5);
    cout<<"Mostrando los elementos de la bicola2"<<endl;
    mostrar_bicola(bicola2);//Nos muestra: 0 0 0 0
    cout<<"Mostrando los elementos de la bicola3"<<endl;
    mostrar_bicola(bicola3);//Nos muestra: 27 27 27 27
    cout<<"Mostrando los elementos de la bicola4"<<endl;
    mostrar_bicola(bicola4);//Nos muestra: 27 27 27 27
    cout<<"Mostrando los elementos de la bicola5"<<endl;
    mostrar_bicola(bicola5);//Nos muestra: 27 27
    cout<<"Mostrando los elementos de la bicola6"<<endl;
    mostrar_bicola(bicola6);//Nos muestra: 25 11 16 7 2
    return 0;
}
```

`deque<tipo_dato>bicola`	Crea una bicola vacía.
`deque<tipo_dato>bicola(k)`	Donde k es un entero positivo. Crea una bicola con k elementos nulos.
`deque<tipo_dato>bicola(k,dato)`	Donde k es un entero positivo. Crea una bicola con k copias del dato.
`deque<int>A(itB1, itB2)`	Donde itB1 y itB2 son iteradores de otro contenedor B, crea una bicola A con los elementos de B ene l rango [itB1, itB2].
`bicola.~deque()`	Destruye la bicola.

6.4.2 COMPARACION DE BICOLAS

```
#include<iostream>
#include<deque>
using namespace std;
int main(){
```

```
    int x[]={25,7,11,2,19,86,14};
    int y[]={26,7,11,2,19,86,14};
    deque<int>bicola1(x,x+7);
    deque<int>bicola2(x,x+7);
    deque<int>bicola3(y,y+7);

    cout<<"OPERADOR =="<<endl;
    if(bicola1==bicola2)
        cout<<"Las bicolas 1 y 2 son iguales"<<endl;
    else
        cout<<"Las bicolas 1 y 2 son diferentes"<<endl;
    //Muestra: Los bicolas 1 y 2 son iguales

    cout<<"OPERADOR !="<<endl;
    if(bicola1!=bicola3)
        cout<<"Las bicolas 1 y 3 son diferentes"<<endl;
    else
        cout<<"Las bicolas 1 y 3 son iguales"<<endl;
    //Muestra: Las bicolas 1 y 3 son diferentes

    cout<<"OPERADOR >"<<endl;
    if(bicola1>bicola3)
        cout<<"La bicola 1 es mayor a la bicola 3"<<endl;
    else
        cout<<"La bicola 1 NO es mayor a la bicola 3"<<endl;
    //Muestra: La bicola 1 NO es mayor a la bicola 3

    cout<<"OPERADOR <"<<endl;
    if(bicola1<bicola3)
        cout<<"La bicola 1 es menor a la bicola 3"<<endl;
    else
        cout<<"La bicola 1 NO es menor a la bicola 3"<<endl;
    //Muestra: La bicola 1 es menor a la bicola 3
    return 0;
}
```

==	Esta comparación devuelve **true** (verdadrero) si las bicolas son iguales, es decir que los elementos de ambas bicolas son iguales. En caso de que las bicolas sean diferentes la comparación devolverá **false** (falso).
!=	Esta comparación devuelve **true** (verdadrero) si las bicolas son diferentes, es decir que los elementos de ambas bicolas son diferentes. En caso de que las bicolas sean iguales la comparación devolverá **false** (falso).
>	Esta comparación devuelve **true** (verdadrero) si la bicola 1 es mayor lexicográficamente a la bicola 2. Caso contrario devuelve **false** (falso).
<	Esta comparación devuelve **true** (verdadrero) si la bicola 1 es menor lexicográficamente a la bicola 2. Caso contrario devuelve **false** (falso).
>=	Esta comparación devuelve **true** (verdadrero) si la bicola 1 es mayor o igual lexicográficamente a la bicola 2. Caso contrario devuelve **false** (falso).
<=	Esta comparación devuelve **true** (verdadrero) si la bicola 1 es menor lexicográficamente a la bicola 2. Caso contrario devuelve **false** (falso).

6.4.3 INSERCCION DE ELEMENTOS EN LA BICOLA

```
#include<iostream>
#include<deque>
using namespace std;
void mostrar_bicola(deque<int> bc){
    for(int i=0;i<bc.size();i++)
        cout<<bc[i]<<" ";
    cout<<endl;
```

```
}
int main(){
    deque<int> A,B;
    cout<<"\nFUNCION PUSH_BACK"<<endl;
    cout<<"Introduce por detras los numeros: 25, 56, 72 a la bicola A"<<endl;
    A.push_back(25);
    A.push_back(56);
    A.push_back(72);
    cout<<"A: ";
    mostrar_bicola(A);//Muestra: 25 56 72

    cout<<"\nFUNCION PUSH_FRONT"<<endl;
    cout<<"Introduce por adelante los numeros: 11, 2, 8 al la bicola A"<<endl;
    A.push_front(11);
    A.push_front(2);
    A.push_front(8);
    cout<<"A: ";
    mostrar_bicola(A);//Muestra: 8 2 11 25 56 72

    cout<<"\nFUNCION INSERT"<<endl;
    cout<<"Introduce el numero 64 en la 3ra posicion de la bicola A"<<endl;
    A.insert(A.begin()+2,64);
    cout<<"A: ";
    mostrar_bicola(A);//Muestra: 8 2 64 11 25 56 72
    cout<<"Introduce los tres primeros numeros de A en la 1ra posicion de la B"<<endl;
    B.insert(B.begin(),A.begin(),A.begin()+3);
    cout<<"B: ";
    mostrar_bicola(B);//Muestra: 8 2 64
    return 0;
}
```

`push_back(dato)`	Introduce por detrás un dato a la bicola.
`push_front(dato)`	Introduce por adelante un dato a la bicola.
`bicola.insert(it,dato)`	Donde it es un iterador, introduce el dato en la posición it.
`A.insert(itA,itB1,itB2)`	Donde itA es un iterador de A, itB1 e itB2 son iteradores del contenedor B, toma los datos de B en el rango [itB1, itB2] y los introduce en la posición de itA en A.

6.4.4 ASIGNACION E INTERCAMBIO DEL BICOLAS

```
#include<iostream>
#include<deque>
using namespace std;
void mostrar_bicola(deque<int> bc){
    for(int i=0;i<bc.size();i++)
        cout<<bc[i]<<" ";
    cout<<endl;
}
int main(){
    deque<int> bicola1,bicola2,bicola3,bicola4,bicola5;
    cout<<"FUNCION ASSIGN"<<endl;
    cout<<"Asignamos 4 numeros de valor 25 a la bicola1"<<endl;
    bicola1.assign(4,25);
    cout<<"bicola1: ";
    mostrar_bicola(bicola1);//Muestra: 25 25 25 25
    cout<<"Asignamos los elementos de la bicola1 a la bicola2"<<endl;
    bicola2.assign(bicola1.begin(),bicola1.end());
    cout<<"bicola2: ";
    mostrar_bicola(bicola2);//Muestra: 25 25 25 25
    int vec[]={25,11,16,7,2};
    cout<<"Asignamos los elementos de un vector a la bicola3"<<endl;
```

```
    bicola3.assign(vec,vec+5);
    cout<<"bicola3: ";
    mostrar_bicola(bicola3);//Muestra: 25 11 16 7 2
    cout<<"Asignamos los 3 elementos centrales de la bicola3 a la bicola4"<<endl;
    bicola4.assign(bicola3.begin()+1,bicola3.end()-1);
    cout<<"bicola4: ";
    mostrar_bicola(bicola4);//Muestra: 11 16 7

    cout<<"\nOPERADOR ="<<endl;
    cout<<"La bicola5 que esta vacia toma los valores de la bicola4"<<endl;
    bicola5=bicola4;
    cout<<"bicola5: ";
    mostrar_bicola(bicola5);//Nos muestra: 11 6 7

    cout<<"\nFUNCION SWAP"<<endl;
    cout<<"bicola2: ";
    mostrar_bicola(bicola2);//Muestra: 25 25 25 25
    cout<<"bicola5: ";
    mostrar_bicola(bicola5);//Nos muestra: 11 6 7
    cout<<"Intercambiamos las bicolas 2 y 5"<<endl;
    bicola2.swap(bicola5);
    cout<<"bicola2: ";
    mostrar_bicola(bicola2);//Muestra: 11 6 7
    cout<<"bicola5: ";
    mostrar_bicola(bicola5);//Nos muestra: 25 25 25 25
    return 0;
}
```

`bicola.assign(k,dato)`	Donde k es un entero positivo, asigna k copias del dato a la bicola.
`A.assign(itB1,itB2)`	Donde itB1 e itB2 son iteradores de B, toma los elementos de B en el rango [itB1,itB2] y los asigna a la bicola A.
`bicola1=bicola2`	El operador = asigna los elementos de bicola2 a la bicola1.
`bicola1.swap(bicola2)`	Intercambia los elementos de la bicola1 con los elementos de la bicola2.

6.4.5 ELIMINACION DE DATOS DE LA BICOLA

```
#include<iostream>
#include<deque>
using namespace std;
void mostrar_bicola(deque<int> bc){
    for(int i=0;i<bc.size();i++)
        cout<<bc[i]<<" ";
    cout<<endl;
}
int main(){
    deque<int> bicola;
    for(int i=1;i<=15;i++)
        bicola.push_back(i);
    mostrar_bicola(bicola);//Muestra: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

    cout<<"\nFUNCION POP_BACK"<<endl;
    cout<<"Eliminando los tres ultimos elementos de la bicola"<<endl;
    bicola.pop_back();
    bicola.pop_back();
    bicola.pop_back();
    mostrar_bicola(bicola);//Muestra: 1 2 3 4 5 6 7 8 9 10 11 12

    cout<<"\nFUNCION POP_FRONT"<<endl;
    cout<<"Eliminando los tres primeros elementos de la bicola"<<endl;
    bicola.pop_front();
```

```
    bicola.pop_front();
    bicola.pop_front();
    mostrar_bicola(bicola);//Muestra: 4 5 6 7 8 9 10 11 12

    cout<<"\nFUNCION ERASE"<<endl;
    cout<<"Eliminando el tercer elemento de la bicola"<<endl;
    bicola.erase(bicola.begin()+2);
    mostrar_bicola(bicola);//Muestra: 4 5 7 8 9 10 11 12
    cout<<"Eliminando los elementos: 2do, 3ro y 4to la bicola"<<endl;
    bicola.erase(bicola.begin()+1,bicola.begin()+4);
    mostrar_bicola(bicola);//Muestra: 4 9 10 11 12

    cout<<"\nFUNCION CLEAR"<<endl;
    cout<<"Borrando todos los elementos de la bicola"<<endl;
    bicola.clear();
    mostrar_bicola(bicola);//No muestra nada
    return 0;
}
```

`pop_back()`	Elimina el último dato de la bicola.
`pop_front()`	Elimina el primer dato de la bicola.
`bicola.erase(it)`	Donde it es un iterador, elimina el dato que se encuentra en la posición de it.
`bicola.erase(it1, it2)`	Donde it1 y it2 son iteradores, elimina los datos de la bicola a partir de la posición de it1 hasta la posición de it2.
`clear()`	Limpia la bicola, es decir elimina todos los elementos de la bicola.

6.4.6 LONGITUD DE LA BICOLA

```
#include<iostream>
#include<deque>
using namespace std;
void mostrar_bicola(deque<int> bc){
    for(int i=0;i<bc.size();i++)
        cout<<bc[i]<<" ";
    cout<<endl;
}
int main(){
    deque<int> bicola;
    bicola.push_back(25);
    bicola.push_back(56);
    bicola.push_back(72);
    bicola.push_back(64);
    bicola.push_back(100);
    cout<<"bicola: ";
    mostrar_bicola(bicola);//Imprime: 25 56 72 64 100

    cout<<"\nFUNCION SIZE Y MAX_SIZE"<<endl;
    cout<<"La longitud de la bicola es: "<<bicola.size()<<endl;//Imprime: 5
    cout<<"La longitud maxima de la bicola es: "<<bicola.max_size()<<endl;
    //Imprime:1073741823

    cout<<"\nFUNCION RESIZE"<<endl;
    cout<<"Modificamos la longitud de 5 a 3"<<endl;
    bicola.resize(3);
    cout<<"bicola: ";
    mostrar_bicola(bicola);//Imprime: 25 56 72
    cout<<"Modificamos la longitud de 3 a 7"<<endl;
    bicola.resize(7);
    cout<<"bicola: ";
```

```
    mostrar_bicola(bicola);//Imprime: 25 56 72 0 0 0 0
    cout<<"Modificamos la longitud de 7 a 10 con copias del numero 32"<<endl;
    bicola.resize(10,32);
    cout<<"bicola: ";
    mostrar_bicola(bicola);//Nos muestra: 25 56 72 0 0 0 0 32 32 32

    cout<<"\nFUNCION EMPTY"<<endl;
    cout<<"Vaciaremos los elementos de la bicola"<<endl;
    cout<<"bicola: ";
    mostrar_bicola(bicola);
    while(!bicola.empty())//Mientras la bicola no este vacia
        bicola.pop_back();//Eliminando el ultimo dato hasta que la bicola quede vacia
    cout<<"bicola: ";
    mostrar_bicola(bicola);//No imprime nada
    return 0;
}
```

`size()`	Devuelve un entero que representa el tamaño de la bicola.
`max_size()`	Devuelve un entero que representa el tamaño máximo de la bicola: 1073741823.
`resize(m)`	Donde m es un entero positivo, modifica el tamaño de la bicola a m. Si m es menor al tamaño actual de la bicola entonces se conservan los primeros m valores. Si m es mayor al tamaño actual de la bicola entonces se conservan los datos de la bicola añadiéndole elementos vacíos, como ejemplo: si la bicola es de números entonces llenara los datos restantes con 0. Si la bicola es de cadenas entonces llenara los datos restantes con cadenas vacías.
`resize(m,elem)`	Donde m es un entero positivo, modifica el tamaño de la bicola a m. Si m es menor al tamaño actual de la bicola entonces se conservan los primeros m valores. Si m es mayor al tamaño actual de la bicola entonces se conservan los datos de la bicola añadiéndole elementos elem.
`empty()`	Devuelve true (verdadero) si la bicola está vacía, caso contrario devuelve false (falso).

6.4.7 ACCESO A LOS ELEMENTOS DE LA BICOLA

```
#include<iostream>
#include<deque>
using namespace std;
int main(){
    deque<int> bicola;
    bicola.push_back(25);
    bicola.push_back(56);
    bicola.push_back(72);
    bicola.push_back(64);
    bicola.push_back(100);
    cout<<"OPERADOR []"<<endl;
    for(int i=0;i<bicola.size();i++)
        cout<<bicola[i]<<" ";//Imprime: 25 56 72 64 100
    cout<<endl;

    cout<<"\nOPERADOR AT"<<endl;
    for(int i=0;i<bicola.size();i++)
        cout<<bicola.at(i)<<" ";//Imprime: 25 56 72 64 100
    cout<<endl;

    cout<<"\nOPERADOR FRONT"<<endl;
    cout<<"Primer dato de la bicola: "<<bicola.front()<<endl;//Imprime: 25

    cout<<"\nOPERADOR BACK"<<endl;
    cout<<"Ultimo dato de la bicola: "<<bicola.back()<<endl;//Imprime: 100
    return 0;
}
```

`[i]`	Donde i es un entero no negativo. Accede al i-esimo elemento del vector.
`at(i)`	Donde i es un entero no negativo. Accede al i-esimo elemento del vector.
`front()`	Accede al primer elemento del vector
`back()`	Accede al último elemento del vector

6.4.8 ITERADORES

```
#include<iostream>
#include<deque>
using namespace std;
void mostrar_bicola(deque<int> bc){
    for(int i=0;i<bc.size();i++)
        cout<<bc[i]<<" ";
    cout<<endl;
}
int main(){
    deque<int> bicola;
    bicola.push_back(25);
    bicola.push_back(56);
    bicola.push_back(72);
    bicola.push_back(64);
    bicola.push_back(100);
    cout<<"bicola: ";
    mostrar_bicola(bicola);//Nos muestra: 25 56 72 64 100

    cout<<"\nITERADORES"<<endl;
    deque<int>::iterator it;
    for(it=bicola.begin();it!=bicola.end();it++)
        cout<<*it<<" ";
    cout<<endl;

    cout<<"\nITERADORES DE REVERSA"<<endl;
    deque<int>::reverse_iterator rit;
    for(rit=bicola.rbegin();rit!=bicola.rend();rit++)
        cout<<*rit<<" ";
    cout<<endl;

    cout<<"\nIntroduciendo datos mediante la funcion: INSERT"<<endl;
    it=bicola.begin();//iniciamos el iterator en la posicion 2
    it=it+2;//Avanzamos dos posiciones
    cout<<"Introducimos el numero 14 en la posicion 2"<<endl;
    bicola.insert(it,14);
    cout<<"bicola: ";
    mostrar_bicola(bicola);//Nos muestra: 25 56 14 72 64 100

    cout<<"\nEliminando datos mediante la funcion: ERASE"<<endl;
    it=bicola.begin();//iniciamos el iterator en la posicion 2
    it=it+4;//Avanzamos cuatro posiciones
    cout<<"Eliminamos el numero de la posicion 4"<<endl;
    bicola.erase(it);
    cout<<"bicola: ";
    mostrar_bicola(bicola);//Nos muestra: 25 56 14 72 100
    //Creamos 2 iteradores
    deque<int>::iterator ia,ib;
    ia=bicola.begin()+1;//Toma la posicion 1 de la bicola
    ib=bicola.begin()+3;//Toma la posicion 3 de la bicola
    cout<<"Eliminamos los numeros desde la posicion 1 hasta la 3"<<endl;
    bicola.erase(ia,ib);
    cout<<"bicola: ";
```

```
    mostrar_bicola(bicola);//Nos muestra: 25 72 100
    return 0;
}
```

deque<tipo_dato>:: iterator it	Crea un iterator de deque denominado it.
deque<tipo_dato>:: reverse_iterator rit	Crea un iterator reverso de deque denominado rit.
begin()	Iterador que apunta al principio de la bicola.
end()	Iterador que apunta al final de la bicola.

6.5 LISTS (LISTAS)

6.5.1 CONSTRUCTORES DE LISTA

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<list>//Libreria para el manejo de listas
using namespace std;
void mostrar_lista(list<int> &L){
    list<int>:: iterator it;
    for(it=L.begin();it!=L.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    //crea una lista vacia
    list<int>lista1;
    //crea una lista con cinco elementos vacios
    list<int>lista2(5);
    //crea una lista con cinco copias del numero 34
    list<int>lista3(5,34);
    //crea una lista4 con los elementos de la lista3
    list<int>lista4(lista3.begin(),lista3.end());
    //crea una lista5 con los elementos de la lista4
    list<int>lista5(lista4);
    //crea una lista con los elementos de un array
    int x[]={12,46,0,10,3,1,23};
    list<int>lista6(x,x+sizeof(x)/sizeof(int));

    cout<<"Mostrando el contenido de las listas"<<endl;
    cout<<"lista 2: ";
    mostrar_lista(lista2);//Muestra: 0 0 0 0 0
    cout<<"lista 3: ";
    mostrar_lista(lista3);//Muestra: 34 34 34 34 34
    cout<<"lista 4: ";
    mostrar_lista(lista4);//Muestra: 34 34 34 34 34
    cout<<"lista 5: ";
    mostrar_lista(lista5);//Muestra: 34 34 34 34 34
    cout<<"lista 6: ";
    mostrar_lista(lista6);//Muestra: 12 46 0 10 3 1 23
    return 0;
}
```

list<tipo_dato>lista	Crea una lista vacía.
list<tipo_dato>lista (m)	Donde m es un entero positivo. Crea una lista con m elementos nulos.
list<tipo_dato>lista (m,dato)	Donde m es un entero positivo. Crea una lista con m copias del dato.
list<tipo_dato>A(itB1, itB2)	Crea la lista A con los elementos de la lista B en el rango [itB1, itB2].
list<tipo_dato>listaA(listaB)	Crea una lista A con todos los elementos de la lista B
x[m]	Donde i, j son enteros positivos. Crea una lista con los elementos de x[]

`list<int>lista(x+i,x+j)`	desde la posición i hasta la posición j.
`lista.~list()`	Destruye la lista.

6.5.2 COMPARACION DE LISTAS

```
#include<iostream>
#include<list>
using namespace std;
int main(){
    int x[]={25,7,11,2,19,86,14};
    int y[]={26,7,11,2,19,86,14};
    list<int>lista1(x,x+7);
    list<int>lista2(x,x+7);
    list<int>lista3(y,y+7);

    cout<<"OPERADOR =="<<endl;
    if(lista1==lista2)
        cout<<"Las listas 1 y 2 son iguales"<<endl;
    else
        cout<<"Las listas 1 y 2 son diferentes"<<endl;
    //Muestra: Los listas 1 y 2 son iguales
    cout<<"OPERADOR !="<<endl;
          if(lista1!=lista3)
        cout<<"Las listas 1 y 3 son diferentes"<<endl;
    else
        cout<<"Las listas 1 y 3 son iguales"<<endl;
    //Muestra: Las listas 1 y 3 son diferentes

    cout<<"OPERADOR >"<<endl;
          if(lista1>lista3)
        cout<<"La lista 1 es mayor a la lista 3"<<endl;
    else
        cout<<"La lista 1 NO es mayor a la lista 3"<<endl;
    //Muestra: La lista 1 NO es mayor a la lista 3

    cout<<"OPERADOR <"<<endl;
          if(lista1<lista3)
        cout<<"La lista 1 es menor a la lista 3"<<endl;
    else
        cout<<"La lista 1 NO es menor a la lista 3"<<endl;
    //Muestra: La lista 1 es menor a la lista 3
    return 0;
}
```

==	Esta comparación devuelve **true** (verdadrero) si las listas son iguales, es decir que los elementos de ambas listas son iguales. En caso de que las listas sean diferentes la comparación devolverá **false** (falso).
!=	Esta comparación devuelve **true** (verdadrero) si las listas son diferentes, es decir que los elementos de ambas listas son diferentes. En caso de que las listas sean iguales la comparación devolverá **false** (falso).
>	Esta comparación devuelve **true** (verdadrero) si la lista 1 es mayor lexicográficamente a la lista 2. Caso contrario devuelve **false** (falso).
<	Esta comparación devuelve **true** (verdadrero) si la lista 1 es menor lexicográficamente a la lista 2. Caso contrario devuelve **false** (falso).
>=	Esta comparación devuelve **true** (verdadrero) si la listas 1 es mayor o igual lexicográficamente a la lista 2. Caso contrario devuelve **false** (falso).
<=	Esta comparación devuelve **true** (verdadrero) si la lista 1 es menor lexicográficamente a la lista 2. Caso contrario devuelve **false** (falso).

6.5.3 ASIGNACION E INTERCAMBIO DE LISTAS

```
#include<iostream>
#include<list>
using namespace std;
void mostrar_lista(list<int> &L){
    list<int>:: iterator it;
    for(it=L.begin();it!=L.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    list<int>lista1,lista2,lista3,lista4;
    int x[]={123,45,25,67,55};

    cout<<"FUNCION ASSIGN"<<endl;
    cout<<"Asignamos los elementos de un vector a la lista1"<<endl;
    lista1.assign(x,x+5);
    cout<<"Lista1: ";
    mostrar_lista(lista1);//Imprime: 123 45 25 67 55
    cout<<"Asignamos los elementos dela lista1 a la lista2"<<endl;
    lista2.assign(lista1.begin(),lista1.end());
    cout<<"Lista2: ";
    mostrar_lista(lista2);//Imprime: 123 45 25 67 55
    cout<<"Asignamos 7 copias del numero 100 a la lista3"<<endl;
    lista3.assign(7,100);
    cout<<"Lista3: ";
    mostrar_lista(lista3);//Imprime: 100 100 100 100 100 100 100

    cout<<"\nOPERADOR ="<<endl;
    cout<<"Asignamos los elementos dela lista1 a la lista4"<<endl;
    lista4=lista1;
    cout<<"Lista4: ";
    mostrar_lista(lista4);//Imprime: 123 45 25 67 55

    cout<<"\nFUNCION SWAP"<<endl;
    cout<<"Lista1: ";
    mostrar_lista(lista1);//Imprime: 123 45 25 67 55
    cout<<"Lista3: ";
    mostrar_lista(lista3);//Imprime: 100 100 100 100 100 100 100
    cout<<"Intercambiamos los valores de la lista1 y la lista3"<<endl;
    lista1.swap(lista3);
    cout<<"Lista1: ";
    mostrar_lista(lista1);//Imprime: 100 100 100 100 100 100 100
    cout<<"Lista3: ";
    mostrar_lista(lista3);//Imprime: 123 45 25 67 55
    return 0;
}
```

`listaA.assign(listaB)`	Asigna los elementos de la lista B en la lista A.
`listaA.assign(listaB.begin(), lista.end())`	Asigna los elementos de la lista B en la lista A.
`listaA=listaB`	Asigna los elementos de la lista B en la lista A.
`lista.assign(k,dato)`	Donde k es un número entero positivo, assigna a la lista k copias del dato.
`listaA.swap(listaB)`	Intercambia los elementos de la lista con los elementos de la listaB

6.5.4 INSERCCION DE DATOS EN LAS LISTAS

```
#include<iostream>
#include<list>
```

```
using namespace std;
void mostrar_lista(list<int> &L){
    list<int>:: iterator it;
    for(it=L.begin();it!=L.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    list<int>lista1,lista2;
    cout<<"\nFUNCION PUSH_FRONT"<<endl;
    cout<<"Introduciendo los datos: 38 20 13 4 11 por delante en la lista 1"<<endl;
    lista1.push_front(38);
    lista1.push_front(20);
    lista1.push_front(13);
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: 13 20 38

    cout<<"\nFUNCION PUSH_BACK"<<endl;
    cout<<"Introduciendo los datos: 42 22 11 31 14 por detras en la lista 1"<<endl;
    lista1.push_back(42);
    lista1.push_back(22);
    lista1.push_back(11);
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: 13 20 38 42 22 11

    cout<<"\nFUNCION INSERT"<<endl;
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: 13 20 38 42 22 11
    cout<<"Insertamos al principio (Posicion 0) de la lista 1 el numero 72"<<endl;
    lista1.insert(lista1.begin(),72);
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: 72 13 20 38 42 22 11
    cout<<"Insertamos al final de la lista 1 el numero 24"<<endl;
    lista1.insert(lista1.end(),24);
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: 72 13 20 38 42 22 11 24
    list<int>:: iterator it; //Creamos un iterator
    it=lista1.begin();//El iterator toma el principio de la lista
    advance(it,3); //Avanzamos tres posiciones
    cout<<"Elemento de la posicion 3 de la lista: "<<*it<<endl;//Muestra: 38
    cout<<"Insertamos el numero 56 en la posicion 3"<<endl;
    lista1.insert(it,56);
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: 72 13 20 56 38 42 22 11 24
    //El puntero aun apunta al numero 24, después señalamos la posición 3
    it=lista1.begin();
    advance(it,3);
    cout<<"Insertamos cuatro copias del numero 9 en la posicion 3"<<endl;
    lista1.insert(it,4,9);
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: 72 13 20 9 9 9 9 56 38 42 22 11 24
    cout<<"Insertamos el contenido del vector en la posicion 6"<<endl;
    int x[]={10,16,23,3,7};
    it--;//El puntero vuelve una posicion atras
    lista1.insert(it,x,x+5);
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: 72 13 20 9 9 9 10 16 23 3 7 9 56 38 42 22 11 24
    return 0;
}
```

`push_back(dato)`	Introduce un dato en la posición final de la lista.
`push_front(dato)`	Introduce un dato en la posición inicial de la lista.

`lista.insert(it,dato)`	Donde it es iterador, introduce el dato en la posición del iterador it.
`lista.insert(it,k,dato)`	Donde it es un iterador, k es un entero positivo, introduce k copias del dato en la posición del iterador it.
`listaA.insert(it,itb1, itb2)`	Donde it es un iterador de la lista, itb1, itb2 son iteradores de otro contenedor STL, introduce los elementos del contenedor B en la lista A en la posición del iterador it.

6.5.5 ELIMINACION DE DATOS DE LAS LISTAS

```
#include<iostream>
#include<list>
using namespace std;
void mostrar_lista(list<int> L){
    list<int>:: iterator it;
    for(it=L.begin();it!=L.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    int x[]={72, 10, 16, 10, 16, 23, 3, 56, 24};
    list<int>lista(x,x+9);
    list<int>::iterator a,b;

    cout<<"\nFUNCION ERASE"<<endl;
    cout<<"Lista: ";
    mostrar_lista(lista);//Muestra: 72 10 16 10 16 23 3 56 24
    cout<<"Eliminamos el primer elemento (Posicion 0) de la lista"<<endl;
    lista.erase(lista.begin());
    cout<<"Lista: ";
    mostrar_lista(lista);//Muestra: 10 16 10 16 23 3 56 24
    //el iterador apunta al principio de la lista
    a=lista.begin();
    advance(a,5);
    cout<<"Eliminamos el cuarto elemento (Posicion 5) de la lista"<<endl;
    lista.erase(a);
    cout<<"Lista: ";
    mostrar_lista(lista);//Muestra: 10 16 10 16 23 56 24
    //Los iteradores los inicializamos en el principio(Posicion 0)
    cout<<"Eliminamos el 3er, 4to y 5to elemento"<<endl;
    a=b=lista.begin();
    //El iterador a avanza dos posiciones
    advance(a,2);
    //El iterador b avanza cinco posiciones
    advance(b,5);
    lista.erase(a,b);
    cout<<"Lista: ";
    mostrar_lista(lista);//Muestra: 10 16 56 24

    cout<<"\nFUNCION CLEAR"<<endl;
    cout<<"Lista: ";
    mostrar_lista(lista);//Muestra: 10 16 56 24
    cout<<"Eliminamos todos los elementos de la lista"<<endl;
    lista.clear();
    cout<<"Lista: ";
    mostrar_lista(lista);//No muestra nada
    return 0;
}
```

`erase(it)`	Donde it es un iterador de lista, elimina el dato que se encuentra en la posición del iterador.
`erase(it1,it2)`	Donde it1 e it2 son iteradores de lista, elimina los datos que se encuentra a partir de la posición

	de it1 hasta la posición de it2.
`clear()`	Elimina todos los datos de la lista

6.5.6 TAMAÑO DE LA LISTA

```
#include<iostream>
#include<list>
using namespace std;
void mostrar_lista(list<int> L){
    list<int>:: iterator it;
    for(it=L.begin();it!=L.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    list<int> lista;
    cout<<"\nFUNCION SIZE Y MAX_SIZE"<<endl;
    lista.push_back(25);
    lista.push_back(56);
    lista.push_back(72);
    lista.push_back(64);
    lista.push_back(100);
    cout<<"lista: ";
    mostrar_lista(lista);//Muestra: 25 56 72 64 100
    cout<<"La longitud de la lista es: "<<lista.size()<<endl;//5
    cout<<"La longitud maxima de la lista es: "<<lista.max_size()<<endl;//357913941

    cout<<"\nFUNCION RESIZE"<<endl;
    cout<<"Modificamos la longitud de 5 a 3"<<endl;
    lista.resize(3);
    cout<<"lista: ";
    mostrar_lista(lista);//Nos muestra: 25 56 72
    cout<<"Modificamos la longitud de 3 a 7"<<endl;
    lista.resize(7);
    cout<<"lista: ";
    mostrar_lista(lista);//Nos muestra: 25 56 72 0 0 0 0
    cout<<"Modificamos la longitud de 7 a 10 y ademas con copias del numero 32"<<endl;
    lista.resize(10,32);
    cout<<"lista: ";
    mostrar_lista(lista);//Nos muestra: 25 56 72 0 0 0 0

    cout<<"\nFUNCION EMPTY"<<endl;
    cout<<"Vaciaremos los elementos de la lista"<<endl;
    cout<<"lista: ";
    mostrar_lista(lista);//Nos muestra: 25 56 72 0 0 0 0
    while(!lista.empty())//Mientras la lista no este vacia
        lista.pop_back();//Eliminando todos los datos de la lista
    return 0;
}
```

`size()`	Devuelve un entero no negativo que representa el tamaño de la lista.
`max_size()`	Devuelve un entero que representa el tamaño máximo de la lista: 357913941.
`resize(m)`	Donde m es un entero positivo, modifica el tamaño de la lista a m. Si m es menor al tamaño actual de la lista entonces se conservan los primeros m valores. Si m es mayor al tamaño actual de la lista entonces se conservan los datos de la lista añadiéndole elementos vacíos, como ejemplo: si la lista es de números entonces llenara los datos restantes con 0. Si la lista es de cadenas entonces llenara los datos restantes con cadenas vacías.
`resize(m,elem)`	Donde m es un entero positivo, modifica el tamaño de la lista a m. Si m es menor al tamaño actual de la lista entonces se conservan los primeros m valores.

	Si m es mayor al tamaño actual de la lista entonces se conservan los datos de la lista añadiéndole elementos elem.
`empty()`	Devuelve true (verdadero) si la lista está vacío caso contrario devuelve false (falso).

6.5.7 ACCESO A LOS ELEMENTOS DE LA LISTA

```
#include<iostream>
#include<list>
using namespace std;
int main(){
    list<int> lista;
    lista.push_back(25);
    lista.push_back(11);
    lista.push_back(7);
    lista.push_back(2);
    lista.push_back(16);

    cout<<"FUNCIUON FRONT Y BACK"<<endl;
    cout<<"Primer elemento"<<endl;
    cout<<lista.front()<<endl;//Muestra 25
    cout<<"Ultimo elemento"<<endl;
    cout<<lista.back()<<endl;//Muestra 16
    return 0;
}
```

`front()`	Accede al primer elemento de la lista
`back()`	Accede al último elemento de la lista

6.5.8 ITERADORES

```
#include<iostream>
#include<list>
using namespace std;
int main(){
    list<int> lista;
    lista.push_back(25);
    lista.push_back(11);
    lista.push_back(7);
    lista.push_back(2);
    lista.push_back(16);

    cout<<"ITERADORES"<<endl;
    list<int>::iterator it;
    for(it=lista.begin();it!=lista.end();it++)
        cout<<*it<<" ";//Imprime: 25 11 7 2 16
    cout<<endl;

    cout<<"\nITERADORES DE REVERSA"<<endl;
    list<int>::reverse_iterator rit;
    for(rit=lista.rbegin();rit!=lista.rend();rit++)
        cout<<*rit<<" ";//Imprime: 16 2 7 11 25
    cout<<endl;

    cout<<"\nOPERACION DE INSERT CON ITERADORES"<<endl;
    list<int>otralista;
    list<int>::iterator a,b;
    a=lista.begin();
    b=lista.end();
    advance(a,2);
```

```
    advance(b,-1);
    otralista.insert(otralista.begin(),a,b); //Introduce 7 2
    for(it=otralista.begin();it!=otralista.end();it++)
        cout<<*it<<" ";//Imprime: 7 2
    cout<<endl;
    return 0;
}
```

`list<tipo_dato>:: iterator it`	Creamos un iterador de lista denominado it.
`list<tipo_dato>:: reverse_iterator rit`	Creamos un iterador de reversa de lista denominado rit.
`begin()`	Iterador que apunta al principio de la lista.
`end()`	Iterador que apunta al final de la lista.
`rbegin()`	Iterador reverso que apunta al principio de la lista invertida.
`rend()`	Iterador reverso que apunta al final de la lista invertida.

6.5.9 OPERACIONES DE LISTA

```
#include<iostream>
#include<list>
using namespace std;
void mostrar_lista(list<string> &L){
    list<string>::iterator it;
    for(it=L.begin();it!=L.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
void mostrar_lista(list<int> &L){
    list<int>::iterator it;
    for(it=L.begin();it!=L.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
bool comparar(string cad1,string cad2){
    int i=0;
    while(i<cad1.length() && i<cad2.length()){
        if(tolower(cad1[i])<tolower(cad2[i]))
            return true;
        else
            if(tolower(cad1[i])>tolower(cad2[i]))
                return false;
        i++;
    }
    return (cad1.length()<cad2.length());
}
bool menor(int num1,int num2){return (num1<num2);}
bool esimpar(int num){return(num&1);}
int main(){
    //creamos las listas y las cargamos con datos
    list<string>lista1;
    list<int>lista2,lista3;

    lista1.push_back("Jose");
    lista1.push_back("Roger");
    lista1.push_back("mAuriCio");
    lista1.push_back("jose");
    lista1.push_back("rogeR");
    lista1.push_back("Mauricio");
```

```
    lista2.push_back(10);
    lista2.push_back(4);
    lista2.push_back(22);
    lista2.push_back(4);
    lista2.push_back(20);
    lista2.push_back(10);
    lista2.push_back(25);
    lista2.push_back(11);
    lista2.push_back(22);
    lista2.push_back(27);

    lista3.push_back(10);
    lista3.push_back(30);
    lista3.push_back(21);
    lista3.push_back(1);
    lista3.push_back(12);
    lista3.push_back(2);
    lista3.push_back(3);

    list<int>lista4(lista2);
    list<int>lista5(lista3);

    cout<<"OPERACION SORT"<<endl;
    cout<<"Lista 1 sin ordenar: ";
    mostrar_lista(lista1);//Muestra: Jose Roger mAuriCio jose rogeR Mauricio
    cout<<"Lista 1 ordenada : ";
    lista1.sort();
    mostrar_lista(lista1);//Muestra: Jose Mauricio Roger jose mAuriCio rogeR
    cout<<"Lista 1 ordenada con la funcion comparar: ";
    lista1.sort(comparar);
    mostrar_lista(lista1);//Muestra: Jose jose Mauricio mAuriCio Roger rogeR

    cout<<"\nOPERACION REVERSE"<<endl;
    cout<<"Lista 1 actual: ";
    mostrar_lista(lista1);//Muestra: Jose jose Mauricio mAuriCio Roger rogeR
    cout<<"Lista 1 volteada al reves: ";
    lista1.reverse();
    mostrar_lista(lista1);//Muestra: rogeR Roger mAuriCio Mauricio jose Jose

    cout<<"\nOPERACION UNIQUE"<<endl;
    cout<<"Lista 2: ";
    mostrar_lista(lista2);//Muestra: 10 4 22 4 20 10 25 11 22 27
    //Es necesario ordenar la lista
    lista2.sort();
    cout<<"Eliminando los elementos repetidos"<<endl;
    lista2.unique();
    cout<<"Lista 2: ";
    mostrar_lista(lista2);//Muestra: 4 10 11 20 22 25 27

    cout<<"\nOPERACION REMOVE"<<endl;
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: rogeR Roger mAuriCio Mauricio jose Jose
    list<string>:: iterator it;
    cout<<"Eliminamos el cuarto elemento mediante el iterador"<<endl;
    it=lista1.begin();
    advance(it,3);
    lista1.remove(*it);
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: ogeR Roger mAuriCio jose Jose
    cout<<"Eliminamos el elemento 'jose' directamente"<<endl;
    lista1.remove("jose");
    cout<<"Lista 1: ";
    mostrar_lista(lista1);//Muestra: rogeR Roger mAuriCio Jose
```

```
    cout<<"\nOPERACION REMOVE IF"<<endl;
    cout<<"Lista 2: ";
    mostrar_lista(lista2);//Muestra: 4 10 11 20 22 25 27
    cout<<"Eliminamos los elementos impares"<<endl;
    lista2.remove_if(esimpar);
    cout<<"Lista 2: ";
    mostrar_lista(lista2);//Muestra: 4 10 20 22

    cout<<"\nOPERACION MERGE"<<endl;
    cout<<"Lista 2: ";
    mostrar_lista(lista2);//Muestra: 4 10 20 22
    cout<<"Lista 3: ";
    mostrar_lista(lista3);//Muestra: 10 30 21 1 12 2 3
    //ordenamos la lista 3, es necesario que ambas listas esten ordenadas
    lista3.sort();
    lista3.merge(lista2,menor);
    cout<<"Los elementos de la lista 2 van a la lista 3 de manera ordenada"<<endl;
    cout<<"Lista 3: ";
    mostrar_lista(lista3);//Muestra: 1 2 3 4 10 10 12 20 21 22 30

    cout<<"\nOPERACION SPLICE"<<endl;
    cout<<"Lista 4: ";
    mostrar_lista(lista4);//Muestra: 10 4 22 4 20 10 25 11 22 27
    cout<<"Lista 5: ";
    mostrar_lista(lista5);//Muestra: 10 30 21 1 12 2 3
    //creamos un iterador
    list<int>:: iterator ab;
    ab=lista4.begin();
    advance(ab,5);
    cout<<"Llevamos los elementos de la lista 5 a la lista 4 en la posicion 5"<<endl;
    lista4.splice(ab,lista5);
    cout<<"Lista 4: ";
    mostrar_lista(lista4);//Muestra: 10 4 22 4 20 10 30 21 1 12 2 3 10 25 11 22 27
    return 0;
}
```

`sort()`	Ordena los elementos de la lista
`reverse()`	Invierte los elementos de la lista
`unique()`	Elimina los elementos repetidos de la lista siempre y cuando la lista haya sido ordenada.
`remove(dato)`	Elimina el dato de la lista, si el dato no buscado no se encuentra en la lista entonces no pasara nada.
`remove_if(condicion)`	Dada una condición, se eliminara los datos que cumplan con esa condición.
`listaA.merge(listaB,condicion)`	Los elementos de la lista B se van a la lista A, de manera ordenada, (es necesario que las listas A y B estén ordenadas). La lista B queda vacía.
`listaA.merge(itA,listaB)`	Los elementos de la lista B se van a la lista A en la posición del iterador itA de la lista A. La lista B queda vacía.

6.6 SETS (CONJUNTOS)

6.6.1 CONSTRUCTORES DE CONJUNTOS

```
#include<iostream>//Libreria para la entrada y salidad de datos
#include<set>//Libreria para el manejo de conjuntos
using namespace std;
void mostrar_conjunto(set<int> c){
    for(set<int>::iterator it=c.begin();it!=c.end();it++)
        cout<<*it<<" ";
```

```
    cout<<endl;
}
int main(){
    //crea un conjunto 1 vacio
    set<int>conj1;
    //creamos un conjunto 2 con los elementos de un vector
    int x[]={27,4,20,13,16,4,9};
    set<int>conj2(x,x+7);
    //creamos un conjunto 3 con los elementos del conjunto 2
    set<int>conj3(conj2);
    //Creamos dos iteradores
    set<int>::iterator a,b;
    //creamos un conjunto 4 con elementos del conjunto 2
    //a partir de la posicion 2 hasta la 4
    a=b=conj2.begin();
    advance(a,2);
    advance(b,4);
    set<int>conj4(a,b);
    cout<<"Mostrando los elementos"<<endl;
    cout<<"Conjunto 1: ";
    mostrar_conjunto(conj1);//No Muestra nada
    cout<<"Conjunto 2: ";
    mostrar_conjunto(conj2);//Muestra: 4 9 13 16 20 27
    cout<<"Conjunto 3: ";
    mostrar_conjunto(conj3);//Muestra: 4 9 13 16 20 27
    cout<<"Conjunto 4: ";
    mostrar_conjunto(conj4);//Muestra: 13 16
    return 0;
}
```

`set<tipo_dato>conjunto`	Crea un conjunto vacio.
`set<tipo_dato>A(B)`	Crea un conjunto A con los elementos del conjunto B.
`set<tipo_dato>A(it1, it2)`	Donde it1 e it2 son iteradores de un contenedor STL. Crea un conjunto A con los elementos de ese contenedor.
`set<tipo_dato>A(itB1,itB2)`	Crea el conjunto A con los elementos de B a partir de la posición del iterador itB1 hasta la posición del iterador itB2.
`x[m]` `set<int>conjunto(x+i,x+j)`	Donde i, j son enteros positivos. Crea un conjunto con los elementos de x[] desde la posición i hasta la posición j.
`Conjunto.~set();`	Destruimos el conjunto.

6.6.2 COMPARACION DE CONJUNTOS

```
#include<iostream>
#include<set>
using namespace std;
int main(){
    int x[]={25,7,11,2,19,86,14};
    int y[]={26,7,11,2,19,86,14};
    set<int>conjunto1(x,x+7);
    set<int>conjunto2(x,x+7);
    set<int>conjunto3(y,y+7);

    cout<<"OPERADOR =="<<endl;
    if(conjunto1==conjunto2)
        cout<<"Los conjuntos 1 y 2 son iguales"<<endl;
    else
        cout<<"Los conjuntos 1 y 2 son diferentes"<<endl;
    //Muestra: Los conjuntos 1 y 2 son iguales
```

```
    cout<<"OPERADOR !="<<endl;
          if(conjunto1!=conjunto3)
        cout<<"Los conjuntos 1 y 3 son diferentes"<<endl;
    else
        cout<<"Los conjuntos 1 y 3 son iguales"<<endl;
    //Muestra: Los conjuntos 1 y 3 son diferentes

    cout<<"OPERADOR >"<<endl;
          if(conjunto1>conjunto3)
        cout<<"El conjunto 1 es mayor al conjunto 3"<<endl;
    else
        cout<<"El conjunto 1 NO es mayor al conjunto 3"<<endl;
    //Muestra: El conjunto 1 NO es mayor al conjunto 3
    cout<<"OPERADOR <"<<endl;
          if(conjunto1<conjunto3)
        cout<<"El conjunto 1 es menor al conjunto 3"<<endl;
    else
        cout<<"El conjunto 1 NO es menor al conjunto 3"<<endl;
    //Muestra: El conjunto 1 es menor al conjunto 3
    return 0;
}
```

==	Esta comparación devuelve **true** (verdadrero) si los conjuntos son iguales, es decir que los elementos de ambos conjuntos son iguales. En caso de que sean diferentes la comparación devolverá **false** (falso).
!=	Esta comparación devuelve **true** (verdadrero) si los conjuntos son diferentes, es decir que los elementos de ambos conjuntos son diferentes. En caso de que los conjuntos sean iguales la comparación devolverá **false** (falso).
>	Esta comparación devuelve **true** (verdadrero) si el conjunto 1 es mayor lexicográficamente al conjunto 2. Caso contrario devuelve **false** (falso).
<	Esta comparación devuelve **true** (verdadrero) si el conjunto 1 es menor lexicográficamente al conjunto 2. Caso contrario devuelve **false** (falso).
>=	Esta comparación devuelve **true** (verdadrero) si el conjunto 1 es mayor o igual lexicográficamente al conjunto 2. Caso contrario devuelve **false** (falso).
<=	Esta comparación devuelve **true** (verdadrero) si el conjunto 1 es menor lexicográficamente al conjunto 2. Caso contrario devuelve **false** (falso).

6.6.3 ASIGNACION E INTERCAMBIO DE CONJUNTOS

```
#include<iostream>
#include<set>
using namespace std;
void mostrar_conjunto(set<int> c){
    for(set<int>::iterator it=c.begin();it!=c.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    int x[]={25,7,11,2,19,86,14};
    int y[]={11,25,19,86,9};
    set<int>conj1(x,x+7),conj2(y,y+5),conj3;

    cout<<"OPERADOR ="<<endl;
    //asignamos los elementos del conjunto 1 al conjunto 3
    conj3=conj1;
    cout<<"Conjunto 3: ";
    mostrar_conjunto(conj3);//Imprime: 2 7 11 14 19 25 86

    cout<<"\nFUNCION SWAP"<<endl;
    cout<<"Conjunto 1: ";
```

```
    mostrar_conjunto(conj1);//Imprime: 2 7 11 14 19 25 86
    cout<<"Conjunto 2: ";
    mostrar_conjunto(conj2);//Imprime: 9 11 19 25 86
    cout<<"Intercambiando el conjunto 1 con el conjunto 2"<<endl;
    conj1.swap(conj2);
    cout<<"Conjunto 1: ";
    mostrar_conjunto(conj1);//Imprime: 9 11 19 25 86
    cout<<"Conjunto 2: ";
    mostrar_conjunto(conj2);//Imprime: 2 7 11 14 19 25 86
    return 0;
}
```

A=B	El operador = Asigna los valores del conjunto B al conjunto A.
A.swap(B)	Intercambia los elementos de los conjuntos A y B.

6.6.4 INSERCCION Y ELIMINACION DE DATOS

```
#include<iostream>
#include<set>
using namespace std;
void mostrar_conjunto(set<int> c){
    for(set<int>::iterator it=c.begin();it!=c.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    set<int>conjunto;
    int x[]={11,25,19,86,9};

    cout<<"FUNCION INSERT"<<endl;
    cout<<"Introducimos los datos: 9 45 27 4 29 13 36 18 9 27 al conjunto"<<endl;
    conjunto.insert(9);
    conjunto.insert(45);
    conjunto.insert(27);
    conjunto.insert(4);
    conjunto.insert(29);
    conjunto.insert(13);
    conjunto.insert(36);
    conjunto.insert(18);
    conjunto.insert(9);
    conjunto.insert(27);
    mostrar_conjunto(conjunto);//Muestra: 4 9 13 18 27 29 36 45
    cout<<"Introducimos los datos de un vector al conjunto 2"<<endl;
    conjunto.insert(x,x+5);
    mostrar_conjunto(conjunto);//Muestra: 4 9 11 13 18 19 25 27 29 36 45 86

    cout<<"\nFUNCION ERASE"<<endl;
    set<int>::iterator a,b; //creamos dos iteradores
    a=b=conjunto.begin();
    cout<<"Eliminamos el segundo dato del conjunto"<<endl;
    advance(a,1);
    conjunto.erase(a);
    mostrar_conjunto(conjunto);//Muestra: 4 11 13 18 19 25 27 29 36 45 86
    cout<<"Eliminamos el elemento '36'  del conjunto"<<endl;
    conjunto.erase(36);
    mostrar_conjunto(conjunto);//Muestra: 4 11 13 18 19 25 27 29 45 86
    cout<<"Eliminamos el elemento '100' (no existe) del conjunto"<<endl;
    conjunto.erase(100);
    mostrar_conjunto(conjunto);//Muestra: 4 11 13 18 19 25 27 29 45 86
    cout<<"Eliminamos datos a partir de la posicion 1 hasta la posicion 4"<<endl;
    a=conjunto.begin();
```

```
    advance(a,1);
    advance(b,4);
    conjunto.erase(a,b);
    mostrar_conjunto(conjunto);//Muestra: 4 19 25 27 29 45 86

    cout<<"\nFUNCION CLEAR"<<endl;
    cout<<"Vaciamos el contendido del conjunto"<<endl;
    conjunto.clear();
    mostrar_conjunto(conjunto);//No muestra nada:
    return 0;
}
```

`insert(dato)`	Introduce un dato en el conjunto, el dato se almacena si es que antes no se haya almacenado. Es decir que el conjunto no admite conjuntos repetidos.
`insert(itA1, itA1)`	Donte it1 e it2 son iteradores. Introduce los datos de otro contenedor STL mediate iteradores.
`erase(dato)`	Encuentra un dato y lo elimina.
`erase(it)`	Dado un iterador de conjunto elimina el dato que se encuentra en la posición del iterador.
`erase(it1,it2)`	Donte it1 e it2 son iteradores. Elimina datos del conjunto a partir de la posición del iterador it1, hasta la posición del iterador it2.
`clear()`	Limpia el contenido del conjunto.

6.6.5 TAMAÑO DEL CONJUNTO

```
#include<iostream>
#include<set>
using namespace std;
void mostrar_conjunto(set<int> c){
    for(set<int>::iterator it=c.begin();it!=c.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    int x[]={11,25,19,86,9};
    set<int>conj;
    conj.insert(x,x+5);
    cout<<"Conjunto: ";
    mostrar_conjunto(conj);//Muestra: 9 11 19 25 86
    cout<<"\nFUNCION SIZE"<<endl;
    cout<<"Longitud del conjunto: "<<conj.size()<<endl;//Muestra: 5

    cout<<"\nFUNCION MAXSIZE"<<endl;
    cout<<"Maxima longitud del conjunto: "<<conj.max_size()<<endl;//Muestra: 214748364

    cout<<"\nFUNCION EMPTY"<<endl;
    cout<<"Vaciando el conjunto"<<endl;
    while(!conj.empty())//Mientras el conjunto conj no este vacio
        conj.erase(conj.begin());
    cout<<"Conjunto: ";
    mostrar_conjunto(conj);//Muestra:
    return 0;
}
```

`size()`	Devuelve un entero que representa el tamaño del conjunto.
`max_size()`	Devuelve un entero que representa el tamaño máximo del conjunto: 214748364.
`empty()`	Devuelve true (verdadero) si el conjunto está vacío caso contrario devuelve false (falso).

6.6.6 ITERADORES DE CONJUNTO

```
#include<iostream>
#include<set>
using namespace std;
int main(){
    int x[]={11,25,19,86,9};
    set<int>conj;
    conj.insert(x,x+5);

    cout<<"ITERADOR"<<endl;
    //Creamos un iterador
    set<int>::iterator it;
    for(it=conj.begin();it!=conj.end();it++)
        cout<<*it<<" ";//Imprime: 9 11 19 25 86
    cout<<endl;

    cout<<"\nITERADOR REVERSO"<<endl;
    //Creamos un iterador
    set<int>::reverse_iterator rit;
    for(rit=conj.rbegin();rit!=conj.rend();rit++)
        cout<<*rit<<" ";//Imprime: 86 25 19 11 9
    cout<<endl;
    return 0;
}
```

`set<tipo_dato>:: iterator it`	Creamos un iterador de conjunto denominado it.
`set<tipo_dato>:: reverse_iterator rit`	Creamos un iterador de reversa de conjunto denominado rit.
`begin()`	Iterador que apunta al principio del conjunto.
`end()`	Iterador que apunta al final del conjunto.
`rbegin()`	Iterador reverso que apunta al principio del conjunto invertido.
`rend()`	Iterador reverso que apunta al final del conjunto invertido.

6.6.7 OPERACIONES DE CONJUNTOS

```
#include<iostream>
#include<set>
using namespace std;
void mostrar_conjunto(set<int> c){
    for(set<int>::iterator it=c.begin();it!=c.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    int x[]={11,25,19,86,9,18,36};
    set<int>conj;
    conj.insert(x,x+7);
    set<int>::iterator it;
    cout<<"Conjunto: ";
    mostrar_conjunto(conj);//Muestra: 9 11 18 19 25 36 86

    cout<<"\nOPERACION FIND"<<endl;
    cout<<"Buscando el elemento '18' para eliminarlo"<<endl;
    conj.erase(conj.find(18));
    cout<<"Conjunto: ";
    mostrar_conjunto(conj);//Muestra: 9 11 19 25 36 86
    cout<<"Buscando el elemento '36' para eliminarlo"<<endl;
    it=conj.find(36);
    conj.erase(it);
```

```
    cout<<"Conjunto: ";
    mostrar_conjunto(conj);//Muestra: 9 11 19 25 86

    cout<<"\nOPERACION COUNT"<<endl;
    cout<<"Contamos cuantas veces aparece el elemento '13'"<<endl;
    cout<<conj.count(13)<<endl;//Muestra: 0
    cout<<"Contamos cuantas veces aparece el elemento '18'"<<endl;
    cout<<conj.count(19)<<endl;//Muestra: 1
    return 0;
}
```

`find(dato)`	Busca el dato en el conjunto si es que el dato buscado se encuentra en el contenedor.
`count(dato)`	Retorna un número que representa cuantas veces aparece un dato. En el caso del set que contiene elementos no repetidos entonces el número será 1 o en el caso que el dato no se encuentre en el contenedor retornara 0.

6.6.8 MULTICONJUNTO

```
#include<iostream>
#include<set>
using namespace std;
void mostrar_conjunto(multiset<int> c){
    for(set<int>::iterator it=c.begin();it!=c.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
int main(){
    //Creamos un multiconjunto
    multiset<int> mc;
    mc.insert(9);
    mc.insert(45);
    mc.insert(27);
    mc.insert(4);
    mc.insert(29);
    mc.insert(13);
    mc.insert(36);
    mc.insert(18);
    mc.insert(9);
    mc.insert(27);
    mostrar_conjunto(mc);//Muestra: 4 9 9 13 18 27 27 29 36 45
    cout<<"Longitud del multiconjunto: "<<mc.size()<<endl;//Muestra: 10

    cout<<"\nContamos cuantas veces se presenta elemento '27'"<<endl;
    cout<<mc.count(27)<<endl;//Muestra: 2

    cout<<"\nBuscamos el elemento '18' para eliminarlo"<<endl;
    multiset<int>::iterator it; //creamos un iterador de multiconjunto
    it=mc.begin();
    it=mc.find(18);
    mc.erase(it);
    mostrar_conjunto(mc);//Muestra: 4 9 9 13 27 27 29 36 45
    return 0;
}
```

El multiconjunto tiene las mismas funciones que el conjunto normal además ambos ordenan sus elementos, la diferencia es que un **conjunto (set)** contiene una sola copia de cada elemento y el **multiconjunto (multiset)** puede contener mas de una copia de cada elemento.

6.7 MAPS (MAPAS)

6.7.1 CONSTRUCTORES DE MAPAS

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<map>//Libreria para el manejo de mapas
using namespace std;
void mostrar_mapa(map<char,string>A){
    map<char,string>::iterator it;
    for(it=A.begin();it!=A.end();it++)
        cout<<it->first<<" "<<it->second<<endl;
}
int main(){
    //crea el mapa 1 vacio
    map<char,string>mapa1;
    //ingresamos datos
    mapa1['q']="Jose";
    mapa1['w']="Roberto";
    mapa1['b']="Mijael";
    mapa1['e']="Juan";
    mapa1['w']="Fermin";
    mapa1['x']="Oscar";
    mapa1['e']="Roger";
    mapa1['a']="Rosario";
    cout<<"Contenido del mapa 1"<<endl;
    mostrar_mapa(mapa1);
    //crea el mapa 2 con el contenido del mapa 1
    map<char,string>mapa2(mapa1.begin(),mapa1.end());
    cout<<"\nContenido del mapa 2"<<endl;
    mostrar_mapa(mapa2);
    //crea el mapa 3 con el contenido del mapa 1
    map<char,string>mapa3(mapa1);
    cout<<"\nContenido del mapa 3"<<endl;
    mostrar_mapa(mapa3);
    return 0;
}
```

Contenido del mapa 1	Contenido del mapa 2	Contenido del mapa 3
a Rosario b Mijael e Roger q Jose w Fermin x Oscar	a Rosario b Mijael e Roger q Jose w Fermin x Oscar	a Rosario b Mijael e Roger q Jose w Fermin x Oscar

`map<tipo_dato,tipo_dato>mapa`	Crea un mapa vacio.
`map<tipo_dato,tipo_dato>A(itB1,itB2)`	Crea un mapa A con los elementos del mapa B a partir de la posición del iterador itB1 hasta la posición del iterador itB2.
`map<tipo_dato,tipo_dato>A(B)`	Crea un mapa A con los elementos del mapa B.
`mapa.~map();`	Destruimos el mapa.

6.7.2 COMPARACION DE MAPAS

```
#include<iostream>
#include<map>
using namespace std;
int main(){
    map<char,string>mapa1,mapa2,mapa3;
    mapa1['a']="Fermin";
    mapa1['b']="Jose";
    mapa1['c']="Maudel";
```

```
    mapa2['a']="Fermin";
    mapa2['b']="JosE";
    mapa2['c']="Maudel";
    mapa3['a']="Roger";
    mapa3['d']="Reynaldo";

    cout<<"OPERADOR =="<<endl;
    if(mapa1==mapa2)
        cout<<"Los mapas 1 y 2 son iguales"<<endl;
    else
        cout<<"Los mapas 1 y 2 son diferentes"<<endl;
    //Muestra: Los mapas 1 y 2 son diferentes

    cout<<"OPERADOR !="<<endl;
          if(mapa1!=mapa3)
        cout<<"Los mapas 1 y 3 son diferentes"<<endl;
    else
        cout<<"Los mapas 1 y 3 son iguales"<<endl;
    //Muestra: Los mapas 1 y 3 son diferentes

    cout<<"OPERADOR >"<<endl;
          if(mapa1>mapa3)
        cout<<"El mapa 1 es mayor al mapa 3"<<endl;
    else
        cout<<"El mapa 1 NO es mayor al mapa 3"<<endl;
    //Muestra: El mapa 1 NO es mayor al mapa 3

    cout<<"OPERADOR <"<<endl;
          if(mapa1<mapa3)
        cout<<"El mapa 1 es menor al mapa 3"<<endl;
    else
        cout<<"El mapa 1 NO es menor al mapa 3"<<endl;
    //Muestra: El mapa 1 es menor al mapa 3
          return 0;
}
```

==	Esta comparación devuelve **true** (verdadrero) si los mapas son iguales, es decir que los elementos de ambos mapas son iguales. En caso de que sean diferentes la comparación devolverá **false** (falso).
!=	Esta comparación devuelve **true** (verdadrero) si los mapas son diferentes, es decir que los elementos de ambos mapas son diferentes. En caso de que los mapas sean iguales la comparación devolverá **false** (falso).
>	Esta comparación devuelve **true** (verdadrero) si el mapa 1 es mayor lexicográficamente al mapa 2. Caso contrario devuelve **false** (falso).
<	Esta comparación devuelve **true** (verdadrero) si el mapa 1 es menor lexicográficamente al mapa 2. Caso contrario devuelve **false** (falso).
>=	Esta comparación devuelve **true** (verdadrero) si el mapa 1 es mayor o igual lexicográficamente al mapa 2. Caso contrario devuelve **false** (falso).
<=	Esta comparación devuelve **true** (verdadrero) si el mapa 1 es menor lexicográficamente al mapa 2. Caso contrario devuelve **false** (falso).

6.7.3 ASIGNACION E INTERCAMBIO DE MAPAS

```
#include<iostream>
#include<map>
using namespace std;
#include<iostream>
#include<map>
using namespace std;
void mostrar_mapa(map<char,string>A){
```

```
    map<char,string>::iterator it;
    for(it=A.begin();it!=A.end();it++)
        cout<<it->first<<" "<<it->second<<endl;
}
int main(){
    map<char,string>mapa1,mapa2,mapa3;
    mapa1['a']="Fermin";
    mapa1['b']="Jose";
    mapa1['c']="Maudel";
    mapa2['x']="Maudel";
    mapa2['q']="Roger";
    mapa2['d']="Reynaldo";

    cout<<"OPERADOR ="<<endl;
    //asignamos los elementos del mapa 1 al mapa 3
    mapa3=mapa1;
    cout<<"Mapa 3: "<<endl;
    mostrar_mapa(mapa3);

    cout<<"\nFUNCION SWAP"<<endl;
    cout<<"Intercambiando el mapa 1 con el mapa 2"<<endl;
    mapa1.swap(mapa2);
    cout<<"Mapa 1: "<<endl;
    mostrar_mapa(mapa1);
    cout<<"Mapa 2: "<<endl;
    mostrar_mapa(mapa2);
    return 0;
}
```

Mapa 3:	Mapa 1:	Mapa 2:
a Fermin	d Reynaldo	a Fermin
b Jose	q Roger	b Jose
c Maudel	x Maudel	c Maudel

mapaA=mapaB	El operador = Asigna los valores del mapa B al mapa A.
mapaA.swap(mapaB)	Intercambia el contenido del mapa A con el contenido del mapaB.

6.7.4 INTRODUCCION Y ELIMINACION DE DATOS

```
#include<iostream>
#include<map>
using namespace std;
void mostrar_mapa(map<char,int>A){
    map<char,int>::iterator it;
    for(it=A.begin();it!=A.end();it++){
        cout<<it->first<<" "<<it->second<<endl;
    }
}
int main(){
    map<char,int>mapa1,mapa2,mapa3;
    cout<<"FUNCION INSERT"<<endl;
    //Primera forma de introduccion de datos
    mapa1.insert(pair<char,int>('a',1));
    mapa1.insert(pair<char,int>('b',12));
    mapa1.insert(pair<char,int>('b',17));
    mapa1.insert(pair<char,int>('h',24));
    //Segunda forma de introduccion de datos con un iterador par
    pair<map<char,int>::iterator,bool>et;
    et=mapa1.insert(pair<char,int>('t',48));
    //tercera forma de introduccion de datos con un iterador de mapa
    map<char,int>::iterator it=mapa1.begin();
```

```
    mapa1.insert(it,pair<char,int>('q',24));
    mapa1.insert(it,pair<char,int>('c',45));
    cout<<"Mapa 1"<<endl;
    mostrar_mapa(mapa1);
    //Introduccion de datos del mapa1 al mapa2
    mapa2.insert(mapa1.begin(),mapa1.end());
    cout<<"Mapa 2"<<endl;
    mostrar_mapa(mapa2);
    //Introduccion de datos del mapa1 al mapa3 hasta el elemento c
    mapa3.insert(mapa1.begin(),mapa1.find('c'));
    cout<<"Mapa 3"<<endl;
    mostrar_mapa(mapa3);

    cout<<"\nFUNCION ERASE"<<endl;
    cout<<"Mapa 1"<<endl;
    mostrar_mapa(mapa1);
    cout<<"Borrando el elemento 'b'"<<endl;
    mapa1.erase('b');
    cout<<"Borrando el elemento 'q'"<<endl;
    //utilizamos el iterador que habiamos creado anteriormente
    it=mapa1.find('q');
    cout<<"Borrando a partir del elemento 'h'"<<endl;
    it=mapa1.find('h');
    mapa1.erase(it,mapa1.end());
    cout<<"Mapa 1"<<endl;
    mostrar_mapa(mapa1);

    cout<<"\nFUNCION CLEAR"<<endl;
    cout<<"Eliminamos los elementos del mapa 1"<<endl;
    mapa1.clear();
    cout<<"Mapa 1"<<endl;
    mostrar_mapa(mapa1);//Esta vacio
    return 0;
}
```

`insert(pair<tipo_dato1,tipo_dato2>(dato1, dato2))`	Introduce un par de datos, el dato1 sera la llave la cual será la única, es decir que el contenedor no almacenara datos repetidos en el primer parámetro.
`mapaA.insert(itB1,itB2)`	Introduce los datos del mapa A con los elementos del mapa B a partir de la posición del iterador itB1 hasta la posición del iterador itB2.
`mapaA.insert(itB1,mapaB.find(dato))` `mapaA.insert(mapaB.find(dato),itB2)`	Indtroduce los datos del mapa A con los elementos del mapa B a partir de la posición del iterador itB1 hasta el dato buscado o viceversa.
`erase(dato)` `erase(find(dato))`	Elimina el dato buscado si es que este se encuentra en el mapa.
`mapaA.erase(itA1,itA2);`	Elimina los datos del mapa A partir de la posición del iterador itA1 hasta la posición del iterador itB2.
`mapaA.insert(itA1,mapaA.find(dato))` `mapaA.erase(mapaA.find(dato),itA2);`	Elimina los datos del mapa A partir de la posición del iterador itA1 hasta la posición del dato buscado, o viceversa.
`clear()`	Elimina todos los datos del mapa.

6.7.5 TAMAÑO DEL MAPA Y ACCESO A LOS ELEMENTOS DEL MAPA

```
#include<iostream>
```

```
#include<map>
using namespace std;
int main(){
    map<int,string>mapa;
    mapa.insert(pair<int,string>(486,"Rosario"));
    mapa.insert(pair<int,string>(125,"Oscar"));
    mapa.insert(pair<int,string>(236,"Fermin"));
    mapa.insert(pair<int,string>(218,"Roger"));

    cout<<"OPERADOR []"<<endl;
    cout<<mapa[486]<<endl;//Muestra: Rosario
    map<int,string>::iterator it;
    it=mapa.begin();//Accediendo al primer elemento
    cout<<mapa[it->first]<<endl;//Muestra: Oscar
    it++;//Accediendo al segundo elemento
    cout<<mapa[it->first]<<endl;//Muestra: Roger

    cout<<"\nFUNCION SIZE()"<<endl;
    cout<<"La longitud del mapa es: "<<mapa.size()<<endl;//Muestra: 4

    cout<<"\nFUNCION MAX_SIZE"<<endl;
    cout<<"La maxima longitud del mapa: "<<mapa.max_size()<<endl;//Muestra: 178956970

    cout<<"\nFUNCION EMPTY"<<endl;
    cout<<"Vaciando el mapa"<<endl;
    while(!mapa.empty())//Mientras el mapa no este vacio
        mapa.erase(mapa.begin());
    return 0;
}
```

`size()`	Devuelve un entero que representa el tamaño del mapa.
`max_size()`	Devuelve un entero que representa el máximo tamaño del mapa.
`empty()`	Devuelve true (verdadero) si el mapa está vacío caso contrario devuelve false (falso).
`[llave]`	Accede a los elementos del mapa mediante las llaves que son únicas en un mapa.

6.7.6 ITERADORES

```
#include<iostream>
#include<map>
using namespace std;
int main(){
    map<int,string>mapa;
    mapa.insert(pair<int,string>(486,"Rosario"));
    mapa.insert(pair<int,string>(125,"Oscar"));
    mapa.insert(pair<int,string>(125,"Rolando"));
    mapa.insert(pair<int,string>(236,"Fermin"));
    mapa.insert(pair<int,string>(218,"Roger"));
    map<int,string>::iterator it;
    map<int,string>::reverse_iterator rit;

    cout<<"ITERADORES"<<endl;
    for(it=mapa.begin();it!=mapa.end();it++)
        cout<<it->first<<" "<<it->second<<endl;
    cout<<"\nITERADORES DE REVERSA"<<endl;
    for(rit=mapa.rbegin();rit!=mapa.rend();rit++)
        cout<<rit->first<<" "<<rit->second<<endl;
    return 0;
}
```

`map<tipo_dato,tipo_dato>::iterator it`	Creamos un iterador de mapa denominado it.

`map<tipo_dato,tipo_dato>::reverse_iterator rit`	Creamos un iterador de reversa de mapa denominado rit.
`begin()`	Iterador que apunta al principio del mapa.
`end()`	Iterador que apunta al final del mapa.
`rbegin()`	Iterador reverso que apunta al principio del mapa invertido.
`rend()`	Iterador reverso que apunta al final del mapa invertido.

6.7.7 OPERACIONES DE MAPA

```
#include<iostream>
#include<map>
using namespace std;
void mostrar_mapa(map<int,string>A){
       map<int,string>::iterator it;
    for(it=A.begin();it!=A.end();it++)
        cout<<it->first<<" "<<it->second<<endl;
}
int main(){
    map<int,string>mapa;
    mapa.insert(pair<int,string>(486,"Rosario"));
    mapa.insert(pair<int,string>(125,"Oscar"));
    mapa.insert(pair<int,string>(125,"Rolando"));
    mapa.insert(pair<int,string>(236,"Fermin"));
    mapa.insert(pair<int,string>(218,"Roger"));

    cout<<"FUNCION FIND"<<endl;
    mostrar_mapa(mapa);
    cout<<"El elemento de llave 125 es: ";
    cout<<mapa.find(125)->first<<" "<<mapa.find(125)->second<<endl;
    cout<<"Borrando el elemento de llave 125"<<endl;
    map<int,string>::iterator it;
    it=mapa.find(125);
    mapa.erase(it);
    mostrar_mapa(mapa);

    cout<<"\nFUNCION COUNT"<<endl;
    cout<<"Contando los elementos con llave 236"<<endl;
    cout<<mapa.count(236)<<endl;
    return 0;
}
```

`find(llave)`	Busca un dato determinado de un mapa si es que el dato buscado se encuentra en el contenedor mediante la llave.
`count(llave)`	Retorna un número que representa cuantas veces aparece un elemento mediante la llave. En el caso del map que contiene elementos no repetidos entonces el número será 1 o en el caso que el dato no se encuentre en el contenedor retornara 0.

6.7.8 MULTIMAPA

```
#include<iostream>
#include<map>
using namespace std;
void mostrar_mapa(multimap<int,string> M){
    for(multimap<int,string>::iterator it=M.begin();it!=M.end();it++)
```

```
        cout<<it->first<<" "<<it->second<<endl;
    cout<<endl;
}
int main(){
    multimap<int,string>mapa;
    mapa.insert(pair<int,string>(486,"Rosario"));
    mapa.insert(pair<int,string>(125,"Oscar"));
    mapa.insert(pair<int,string>(125,"Rolando"));
    mapa.insert(pair<int,string>(236,"Fermin"));
    mapa.insert(pair<int,string>(218,"Roger"));
    mapa.insert(pair<int,string>(236,"Jose"));
    mostrar_mapa(mapa);

    cout<<"Longitud del multimapa: "<< mapa.size()<<endl; //Muestra: 6

    cout<<"\nContamos cuantas veces se presenta los elementos con llave '236'"<<endl;
    cout<<mapa.count(236)<<endl;//Muestra: 2

    cout<<"\nBuscamos el elemento de llave '125' para eliminarlo"<<endl;
    multimap<int,string>::iterator it; //creamos un iterador de multiconjunto
    it=mapa.find(125);
    mapa.erase(it);
    mostrar_mapa(mapa);
    return 0;
}
```

```
//Antes de eliminar el dato de llave 125
125 Oscar
125 Rolando
218 Roger
236 Fermin
236 Jose
486 Rosario
```

```
//Despues de eliminar el dato de llave 125
125 Rolando
218 Roger
236 Fermin
236 Jose
486 Rosario
```

El multimapa tiene las mismas funciones que el mapa normal además ambos ordenan las llaves de sus elementos, la diferencia es que un **mapa (map)** contiene una sola copia de cada llave y el **multimapa (multimapa)** puede contener mas de una copia de cada llave.

6.8 BITSET (CONJUNTO DE BITS)

6.8.1 CONTRUCTORES Y OPERACIONES DE BITS

```
#include<iostream>
#include<bitset>
using namespace std;
int main(){
    //crea un conjunto de bits vacio
    bitset<10>primer;
    //crea un conjunto de bits con el valor binario del numero entero 12
    bitset<10>segundo(12);
    //crea un conjunto de bits con la cadena '1001'
    bitset<10>tercero(string("1001"));
    //crea un conjunto de bits con el contenido del segundo conjunto
    bitset<10>cuarto(segundo);
    cout<<"Valor del primer bitset: "<<primer<<endl;//Muestra:0000000000
    cout<<"Valor del segundo bitset: "<<segundo<<endl;//Muestra:0000001100
    cout<<"Valor del tercero bitset: "<<tercero<<endl;//Muestra:0000001001
    cout<<"Valor del cuarto bitset: "<<cuarto<<endl;//Muestra:0000001100
    return 0;
}
```

`bitset<n>cbit`	Crea un conjunto de 10 bits vacio. Los diez bits inicialmente están con cero.
`bitset<n>cbit(numero)`	Dado un numero entero, se contruye un conjunto de 8 bits que contendrá el

	valor binario de dicho numero.
`bitset<n>cbit(string("10101"))`	Dado una cadena, se contruye un conjunto de 8 bits que contendrá la cadena. Los caracteres de dicha cadena deberán ser del valor de 1 o 0.
`bitset<10>cbit1(cbit2)`	Crea un conjunto de bits "cbit1", con el contenido del conjunto de bits "cbit2".
`cin>>Cbit`	Lectura del bitset, esta lectura solo tomara en cuenta los caracteres '0' o '1'.

6.8.2 OPERADORES BINARIOS Y ACCESO DE BITS

```
#include<iostream>
#include<bitset>
using namespace std;
void mostrar_bitset(bitset<8> b){
    for(int i=b.size()-1;i>=0;i--)
        cout<<b[i];
    cout<<endl;
}
int main(){
    bitset<8>primero(14),segundo(11),tercero;
    cout<<"Primero: ";
    mostrar_bitset(primero);//Muestra: 00001110
    cout<<"Segundo: ";
    mostrar_bitset(segundo);//Muestra: 00001011
    cout<<"Tercero: ";
    mostrar_bitset(tercero);//Muestra: 00000000

    cout<<"\nOPERADOR DE ASIGNACION ="<<endl;
    cout<<"Asignando el contenido del segundo bitset al tercer bitset"<<endl;
    tercero=segundo;
    cout<<"Tercero: ";
    mostrar_bitset(tercero);//Muestra: 00001011

    cout<<"\nOPERACIONES DE COMPARACION == y !="<<endl;
    if(tercero==segundo)
        cout<<"Los bitsets segundo y tercero son iguales"<<endl;
    else
        cout<<"Los bitsets segundo y tercero son diferentes"<<endl;
    //Muestra: Los bitsets segundo y tercero son iguales
    if(primero!=segundo)
        cout<<"Los bitsets primero y segundo son diferentes"<<endl;
    else
        cout<<"Los bitsets primero y segundo son iguales"<<endl;
    //Muestra: Los bitsets primero y segundo son diferentes

    cout<<"\nOPERADOR ^ (XOR)"<<endl;
    cout<<"(primero^segundo)= "<<(primero^segundo)<<endl;//Muestra:0000000101

    cout<<"\nOPERADOR & (AND)"<<endl;
    cout<<"(primero&segundo)= "<<(primero&segundo)<<endl;//Muestra:0000001010

    cout<<"\nOPERADOR | (OR)"<<endl;
    cout<<"(primero|segundo)= "<<(primero|segundo)<<endl;//Muestra:0000001111
    cout<<"\nOPERADOR << (RECORRIDO A LA IZQUIERDA)"<<endl;
    cout<<"Tercero: "<<tercero<<endl;//Muestra: 00001011
    cout<<"Recorriendo 1 bit"<<endl;
    tercero=(tercero<<1);
    cout<<"Tercero: "<<tercero<<endl;//Muestra: 00010110
    cout<<"Recorriendo 2 bits"<<endl;
    tercero=(tercero<<2);
    cout<<"Tercero: "<<tercero<<endl;//Muestra: 01011000
```

```
    cout<<"\nOPERADOR >> (RECORRIDO A LA DERECHA)"<<endl;
    cout<<"Recorriendo 4 bits"<<endl;
    tercero=(tercero>>4);
    cout<<"Tercero: "<<tercero<<endl;//Muestra: 00000101

    cout<<"\nOPERADOR ~ (NEGACION)"<<endl;
    tercero=(~tercero);
    cout<<"Tercero: "<<tercero<<endl;//Muestra: 11111010
    return 0;
}
```

`Cbit1^Cbit2`	Operación XOR entre los bitset (conjunto de bits) Cbit1 y Cbit2.
`Cbit1&Cbit2`	Operación AND entre los bitset (conjunto de bits) Cbit1 y Cbit2.
`Cbit1|Cbit2`	Operación OR entre los bitset (conjunto de bits) Cbit1 y Cbit2.
`Cbit<<n`	Recorre **n** bits a la izquierda, es decir que se añaden n bits 0 a la derecha.
`Cbit>>n`	Recorre **n** bits a la derecha, es decir que se añaden n bits 0 a la izquierda.
`~Cbit`	Negacion, es decir que los bits que estén encendidos se apagaran, asi mismo los bits que estén apagados se encenderán. (Los bits de valor 0 tendran el valor de 1. Asi mismo los bits de valor 1 tendran el valor de 0).
`Cbit1==Cbit2`	Es una comparación que devuelve **true** (verdadero) si los bits de ambos conjuntos son iguales. Caso contrario devolverá **false** (falso)**.**
`Cbit1!=Cbit2`	Es una comparación que devuelve **true** (verdadero) si los bits de ambos conjuntos son diferentes. Caso contrario devolverá **false** (falso)**.**
`Cbit1=Cbit2`	Asignacion, es decir que Cbit1 tendra el contenido de Cbit2.
`[i]`	Donde i es un entero no negativo. Este operador accede al i-esimo bit de atrás hacia adelante.

6.8.3 OPERACIONES DE BIT

```
#include<iostream>
#include<bitset>
using namespace std;
int main(){
    bitset<8>bit;
    cout<<bit<<endl;//Muestra: 00000000

    cout<<"\nOPERACION SET"<<endl;
    cout<<"Endendiendo todos los bits"<<endl;
    bit.set();
    cout<<bit<<endl;//Muestra: 11111111
    cout<<"Apagando el bit 2"<<endl;
    bit.set(2,0);
    cout<<bit<<endl;//Muestra: 11111011
    cout<<"Encendiendo el bit 2"<<endl;
    bit.set(2);
    cout<<bit<<endl;//Muestra: 11111111

    cout<<"\nOPERACION RESET"<<endl;
    cout<<"Apagando el bit 6"<<endl;
    bit.reset(6);
    cout<<bit<<endl;//Muestra: 10111111
    cout<<"Apagando todos los bits"<<endl;
    bit.reset();
    cout<<bit<<endl;//Muestra: 00000000

    cout<<"OPERACION FLIP"<<endl;
```

```
    cout<<"Cambiando de valor el bit 2"<<endl;
    bit.flip(2);
    cout<<bit<<endl;//Muestra: 00000100
    cout<<"Cambiando de valor el bit 4"<<endl;
    bit.flip(4);
    cout<<bit<<endl;//Muestra: 00010100
    cout<<"Cambiando de valor todos los bits"<<endl;
    bit.flip();
    cout<<bit<<endl;//Muestra: 11101011
    return 0;
}
```

`set(k)`	Donde k es un número entero no negativo, enciende el k-esimo bit.
`set(k,0)`	Donde k es un número entero no negativo, apaga el k-esimo bit.
`set()`	Enciende todos los bits del conjunto de bits.
`reset(k)`	Donde k es un número entero, apaga el k-esimo bit.
`reset()`	Apaga todos los bits del conjunto de bits.
`flip(k)`	Donde k es un número entero, cambia el valor del k-esimo bit.
`flip()`	Cambia el valor de todos los bits del conjunto.

6.8.4 FUNCIONES DE BITSET

```
#include<iostream>
#include<bitset>
using namespace std;
int main(){
    bitset<7>bit(54);
    cout<<bit<<endl;//Muestra: 0110110

    cout<<"\nFUNCION SIZE"<<endl;
    cout<<"Numero de bits: "<<bit.size()<<endl;//Muestra: 7

    cout<<"\nOPERACION COUNT"<<endl;
    cout<<"Numero de bits encendidos: " <<bit.count()<<endl;//Muestra: 4
    cout<<"Numero de bits apagados: "<<bit.size()-bit.count()<<endl;//Muestra: 3

    cout<<"\nFUNCION TEST"<<endl;
    //Con boolalpha Los bits no se imprimiran como 0's o 1's sino como true o false
    cout<<boolalpha;
    for(int i=0;i<bit.size();i++)
        cout<<bit.test(i)<<endl;

    cout<<"\nFUNCION ANY"<<endl;
    cout<<"Existe algun bit esta encendido?"<<endl;
    if(bit.any())
        cout<<bit.count()<<" bits estan encendidos"<<endl;
    else
        cout<<"Todos los bits estan apagados"<<endl;
    //Muestra: 4 bits estan encendidos

    cout<<"\nFUNCION NONE"<<endl;
    cout<<"Todos los bits estan apagados?"<<endl;
    if(bit.none())
        cout<<"Todos los bits estan apagados"<<endl;
    else
        cout<<"No, "<<bit.count()<<" bits estan encendidos"<<endl;
    //Muestra: No, 4 bits estan encendidos
```

```
    cout<<"\nOPERACION TO_ULONG"<<endl;
    int n=bit.to_ulong();
    cout<<"Numero entero: "<<n<<endl;//Muestra: 54

    cout<<"\nOPERACION TO_STRING"<<endl;
    string cad=bit.to_string();
    cout<<"Cadena: "<<cad<<endl;//Muestra: 0110110
    return 0;
}
```

`size()`	Retorna un numero entero que representa el tamaño (numero de bits) del bitset.
`count()`	Retorna un número entero que representa el número de bits que están encendidos.
`test(k)`	Donde k es un número entero, evalua el k-esimo bit. Retorna **true** (verdadero) si el bit esta encendido, caso contrario retorna **false** (falso).
`any()`	Retorna **true** (verdadero) si algun bit esta encendido, caso contrario retorna **false** (falso).
`none()`	Retorna **true** (verdadero) si todos los bits esta apagados, caso contrario retorna **false** (falso).
`to_ulong()`	Devulve un entero que representa valor decimal del conjunto de bits.
`to_string()`	Devulve una cadena con los elementos del conjunto de bits.

7 ALGORITMOS STL

7.1 OPERACIONES DE SECUENCIA SIN MODIFICACION

7.1.1 ALGORITMO FOR EACH

```
#include<iostream>//Librería para la entrada y salida de datos
#include<algorithm>//Librería para el manejo de algortimos
#include<vector>//Librería para el uso de vectores
using namespace std;
//Funcion para añadir 2 a un numero
void sumar(int &i){i+=2;}//Se utiliza un puntero para que la informacion se almacena
void mostrar(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
void mostrar(int v[],int n){
    for(int i=0;i<n;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    vector<int>x;
    x.push_back(12);
    x.push_back(7);
    x.push_back(24);
    x.push_back(4);
    x.push_back(10);
    int y[]={25, 11, 14, 19};
    for_each(x.begin(),x.end(),sumar);
    mostrar(x);//Muestra: 14 9 26 6 12
    for_each(y,y+4,sumar);
    mostrar(y,4);//Muestra: 27 13 16 21
    return 0;
}
```

`for_each(it1,it2,f(n))`	Donde it1 e it2 son iteradores de un contenedor STL o también pueden ser punteros de un array, f(n) es una función. La función for_each aplica la función f(n) a cada elemento del contenedor o array desde la posición de it1 hasta la posición it2.

7.1.2 ALGORITMOS FIND

```
//ALGORTIMO FIND
#include<iostream>//Librería para la entrada y salida de datos
#include<algorithm>//Librería para el manejo de algortimos
using namespace std;
int main(){
    int y[]={25, 11, 14, 19};
    cout<<"Buscando el numero 14 en el array y"<<endl;
    int *p;
    p=find(y,y+4,14);//Buscando el dato 14
    p++;//Avanamos una posicion del puntero
    cout<<"El siguiente dato de 14 es: "<<*p<<endl;//Muestra: 19
    return 0;
}
```

```
//ALGORTIMO FIND_IF
#include<iostream>//Librería para la entrada y salida de datos
#include<algorithm>//Librería para el manejo de algortimos
#include<vector>//Librería para el uso de vectores
using namespace std;
```

```
//Funcion para evaluar si un numero es impar o no
bool es_impar(int n){return(n&1);}
int main(){
    vector<int>x;
    x.push_back(12);
    x.push_back(7);
    x.push_back(24);
    x.push_back(4);
    x.push_back(10);
    int y[]={25, 11, 14, 19};
    vector<int>::iterator it=find_if(x.begin(),x.end(),es_impar);
    cout<<"El primer numero impar del vector x es: "<<*it<<endl;// Muestra: 7
    int* p=find_if(y,y+4,es_impar);
    cout<<"El primer numero impar del vector y es: "<<*p<<endl;// Muestra: 25
    return 0;
}
```

```
//ALGORITMOS FIND_END Y FIND_FIRST_OF
#include<iostream>//Librería para la entrada y salida de datos
#include<algorithm>//Librería para el manejo de algortimos
#include<vector>//Librería para el uso de vectores
using namespace std;
bool comparar(int a,int b){return(a==b+2);}
int main(){
    vector<int>x;
    int y[]={6,8,10,12};
    for(int i=2;i<=16;i+=2)
        x.push_back(i);
    vector<int>:: iterator it;
    it=find_end(x.begin(),x.end(),y,y+4);
    cout<<"La secuencia comienza a partir de la posicion: "<<it-x.begin()<<endl;//2
    it=find_end(x.begin(),x.end(),y,y+4,comparar);
    cout<<"La secuencia comienza a partir de la posicion: "<<it-x.begin()<<endl;//3
    it=find_first_of(x.begin(),x.end(),y,y+4);
    cout<<"La secuencia comienza con el elemento: "<<*it<<endl;//Muestra: 6
    it=find_first_of(x.begin(),x.end(),y,y+4,comparar);
    cout<<"La secuencia comienza con el elemento: "<<*it<<endl;//Muestra: 8
    return 0;
}
```

```
//ALGORITMO ADJACENT_FIND
#include<iostream>//Librería para la entrada y salida de datos
#include<algorithm>//Librería para el manejo de algortimos
#include<vector>//Librería para el uso de vectores
using namespace std;
int main(){
    vector<int>x;
    x.push_back(100);
    x.push_back(12);
    x.push_back(12);
    x.push_back(24);
    x.push_back(7);
    x.push_back(7);
    vector<int>::iterator it;
    it=adjacent_find(x.begin(),x.end());
    cout<<"El primer par de elementos repetidos es "<<*it<<endl;//12
    it=adjacent_find(++it,x.end());
    cout<<"El segundo par de elementos repetidos es "<<*it<<endl;//7
    return 0;
}
```

`find(it1,it2,dato)`	Donde it1 e it2 son iteradores de un contenedor STL o también pueden ser punteros de un array. La función find busca el dato y devuelve ese elemento.

`find_if(it1,it2,f(n))`	Donde it1 e it2 son iteradores de un contenedor STL o también pueden ser punteros de un array, f(n) es una función. La función find_if busca al primer elemento que cumple la función f(n).
`find_end(a1,a2,b1,b2)`	Donde a1 e a2 son iteradores de un contenedor STL o también pueden ser punteros de un array, b1 e b2 son iteradores de otro contenedor STL o también pueden ser punteros de otro array. La función find_end busca el final de una secuencia.
`find_end(a1,a2,b1,b2,f(n))`	Donde a1 e a2 son iteradores de un contenedor STL o también pueden ser punteros de un array, b1 e b2 son iteradores de otro contenedor STL o también pueden ser punteros de otro array y f(n) es una función. La función find_end busca el final de una secuencia dada la condición f(n).
`find_first_of(a1,a2,b1,b2)`	Donde a1 e a2 son iteradores de un contenedor STL o también pueden ser punteros de un array, b1 e b2 son iteradores de otro contenedor STL o también pueden ser punteros de otro array. La función find_first_of busca el primer dato de una secuencia.
`find_first_of(a1,a2,b1,b2,f(n))`	Donde a1 e a2 son iteradores de un contenedor STL o también pueden ser punteros de un array, b1 e b2 son iteradores de otro contenedor STL o también pueden ser punteros de otro array y f(n) es una función. La función find_first_of busca el primer dato de una secuencia dada la condición f(n).
`adjacent_find(it1,it2)`	Donde it1 e it2 son iteradores de un contenedor STL o también pueden ser punteros de un array. La función adjacent_find busca al primer elemento que este repetido y lo devuelve si es que es que su copia es adiacente (a su derecha o izquierda).

7.1.3 ALGORITMOS COUNT

```
#include<iostream>
#include<algorithm>
using namespace std;
bool es_impar(int n){return (n&1);}
int main(){
    int x[]={25,11,7,25,16,14,25};
    cout<<"Cuantas veces aparece el dato 25?"<<endl;
    cout<<"Aparece: "<<count(x,x+7,25)<<" veces"<<endl;//3
    cout<<"Cuantas numeros impares hay?"<<endl;
    cout<<"Aparece: "<<count_if(x,x+7,es_impar)<<" veces"<<endl;//5
    return 0;
}
```

`count(it1,it2,dato)`	Donde it1 e it2 son iteradores de un contenedor STL o punteros de un array, la función count devuelve un entero que representa las veces que un dato se presenta.
`count_if(it1,it2,f(n))`	Donde it1 e it2 son iteradores de un contenedor STL o punteros de un array, f(n) es una función. La función count_if devuelve un entero que representa la cantidad de datos que cumplen la condición de f(n).

7.1.4 ALGORITMO MISMATCH

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
int main(){
    int x[]={10,20,30,17,50,10};
          vector<int>y;
    for(int i=10;i<=100;i+=10)
```

```
        y.push_back(i);
    pair<vector<int>::iterator,int*>par;
    par=mismatch(y.begin(),y.end(),x);
cout<<"Los pares difieren en: "<<*par.first<<" y "<<*par.second<<endl;//Muestra:40 y 17
    par.first++;
    par.second++;
    par=mismatch(par.first,y.end(),par.second);
cout<<"Los pares difieren en: "<<*par.first<<" y "<<*par.second<<endl;//Muestra:60 y 10
    return 0;
}
```

`mismatch(it1,it2,x)`	Donde it1 e it2 son iteradores de un contenedor STL o punteros de un array y x es un puntero o iterador de otra estructura, la función mismatch devuelve un par que compara las dos estructuras y encuentra el primer par donde las estructuras difieren. La comparación se realiza en el rango it1, it2 y x.
`mismatch(it1,it2,x,f(n))`	Donde it1 e it2 son iteradores de un contenedor STL o punteros de un array y x es un puntero o iterador de otra estructura, f(n) es una función. La función mismatch devuelve un par que compara las dos estructuras y encuentra el primer par donde las estructuras no cumplen la condición de f(n).

7.1.5 ALGORITMO EQUAL

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
int main(){
    int x[]={10,20,30,40,50,60,70};
          vector<int>y;
    for(int i=10;i<=70;i+=10)
        y.push_back(i);
    if(equal(x,x+7,y.begin()))
        cout<<"La estructuras son iguales"<<endl;
    else
        cout<<"La estructuras son diferentes"<<endl;
    //Muestra: La estructuras son iguales
    if(equal(y.begin(),y.end(),x))
        cout<<"La estructuras son iguales"<<endl;
    else
        cout<<"La estructuras son diferentes"<<endl;
    //Muestra: La estructuras son iguales
    return 0;
}
```

`equal(it1,it2,p)`	Donde it1 e it2 son iteradores de un contenedor STL o punteros de un array y p es un puntero o iterador de otra estructura. La función equal devuelve **true** (verdadero), si las estructuras son iguales, caso contrario devuelve **false** (falso). La comparación se realiza en el rango it1, it2 y p.
`equal(it1,it2,p,f(n))`	Donde it1 e it2 son iteradores de un contenedor STL o punteros de un array y p es un puntero o iterador de otra estructura, f(n) es una función. La función equal devuelve **true** (verdadero), si las estructuras cumplen con la condición de f(n), caso contrario devuelve **false** (falso). La comparación se realiza en el rango it1, it2 y p.

7.1.6 ALGORITMOS SEARCH

```
//ALGORITMO SEARCH
#include<iostream>
#include<algorithm>
#include<vector>
```

```
using namespace std;
bool comparar(int a,int b){return (a==b+20);}
int main(){
    int x[]={30,40,50};
    vector<int>y;
    for(int i=10;i<=70;i+=10)
        y.push_back(i);
    vector<int>::iterator it;
    it=search(y.begin(),y.end(),x,x+3);
    if(it!=y.end())
     cout<<"Secuencia de x encontrada en y, en la posicion: "<<(it-y.begin())<<endl;//2
    it=search(y.begin(),y.end(),x,x+3,comparar);
    if(it!=y.end())
     cout<<"Secuencia de x encontrada en y, en la posicion: "<<(it-y.begin())<<endl;//4
    return 0;
}
```

```
//ALGORITMO SEARCH_N
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    int x[]={1,3,5,6,7,7,7,8,9};
    int *p;
    p=search_n(x,x+9,3,7);
    if(p!=x+9)
        cout<<"La secuencia 7 7 7 se encuentra en la posicion: "<<p-x<<endl;//4
    return 0;
}
```

`search(a1,a2,b1,b2)`	Donde a1, a2 son iteradores de un contenedor STL o punteros de un array, b1 y b2 son iteradores de otro contenedor STL o punteros de un array. La función search devuelve un puntero o iterador que apunta a la posición donde comienza una secuencia de la estructura b que se encuentra en la estructura a.
`search(it1,it2,n,m)`	Donde it1 e it2 son iteradores de un contenedor STL o punteros de un array, n y m son números enteros. La función search_n devuelve un puntero o iterador que apunta a la posición donde comienza una secuencia repetida del dato m, n veces.

7.2 OPERACIONES DE SECUENCIA CON MODIFICACION

7.2.1 ALGORITMOS COPY

```
//ALGORITMO COPY
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    int x[]={12,34,56,27,64};
    int y[5];
    copy(x,x+5,y);
    for(int i=0;i<5;i++)
        cout<<y[i]<<" ";//Muestra: 12 34 56 27 64
    return 0;
}
```

```
//ALGORITMO COPY_BACKWARD
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
int main(){
    vector<int>vec;
```

```
    vec.push_back(12);
    vec.push_back(10);
    vec.push_back(25);
    vec.push_back(9);
    vec.push_back(20);
    vec.push_back(13);
    vec.resize(vec.size()+3);//Preparando 3 campos mas para nuevos datos
    copy_backward(vec.begin(),vec.begin()+5,vec.end());
    for(int i=0;i<vec.size();i++)
        cout<<vec[i]<<" ";//Muestra: 12 10 25 9 12 10 25 9 20
    return 0;
}
```

`copy(a1,a2,b)`	Donde a1, a2 son iteradores de un contenedor STL o punteros de un array A, b es iterador de otro contenedor STL o puntero de un array B. La función copy toma los elemento de A desde la posición de a1 hasta la posición a2 y los copia en la posición de b de la estructura B.
`copy_backward(it1,it2,it3)`	Donde it1, it2 e it3 son iteradores de un contenedor STL o punteros de un array. La función copy_backward copia los datos de una misma estructura desde it1 hasta it2, i los copia en la posición de it3 respetando el tamaño actual de la estructura.

7.2.2 ALGORITMOS SWAP

```
//ALGORITMO SWAP
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
void mostrar_vector(int v[],int m){
    for(int i=0;i<m;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
void mostrar_vector(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    int num1,num2;
    num1=35;
    num2=47;
    swap(num1,num2);//Intercambiando el valor de los enteros num1 y num2
    cout<<"num1: "<<num1<<endl; //Muestra: 47
    cout<<"num2: "<<num2<<endl; //Muestra: 35
    int x[]={12,45,67};
    int y[]={1,36,28};
    swap(x,y);//Intercambiando los vectores x[] e y[]
    cout<<"x[]= ";
    mostrar_vector(x,3);//Muestra: 1 36 28
    cout<<"y[]= ";
    mostrar_vector(y,3);//Muestra: 12 45 67

    vector<int>v1(5,12), v2(7,25);
    swap(v1,v2);//Intercambiando los contenedores v1 y v2
    cout<<"vector v1: ";
    mostrar_vector(v1);//Muestra: 25 25 25 25 25 25 25
    cout<<"vector v2: ";
    mostrar_vector(v2);//Muestra: 12 12 12 12 12
    return 0;
}
```

```
//ALGORITMO SWAP_RANGES
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
void mostrar_vector(int v[],int m){
    for(int i=0;i<m;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
void mostrar_vector(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    int x[]={12,45,67,23,58};
    int y[]={1,36,28,102,81};
    //Intercambiando los 3 elementos del medio de x por los tres primeros de y
    swap_ranges(x+1,x+4,y);
    cout<<"x[]= ";
    mostrar_vector(x,5);//Muestra: 12 1 36 28 58
    cout<<"y[]= ";
    mostrar_vector(y,5);//Muestra: 45 67 23 102 81

    vector<int>v1(7,12), v2(10,25);
    //Intercambiando los 3 elementos del medio de v1 por los 3 elementos de v2
    swap_ranges(v1.begin()+2,v1.end()-2,v2.begin()+1);
    cout<<"vector v1: ";
    mostrar_vector(v1);//Muestra: 12 12 25 25 25 12 12
    cout<<"vector v2: ";
    mostrar_vector(v2);//Muestra: 25 12 12 12 25 25 25 25 25 25
    return 0;
}
```

```
//ALGORITMO ITER_SWAP
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
void mostrar_vector(int v[],int m){
    for(int i=0;i<m;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
void mostrar_vector(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    int x[]={12,45,67,23,58};
    int y[]={1,36,28,102,81};
    iter_swap(x+1,y+4);//Intercambiando el 2do elemento de x por el 5to elemento de y
    cout<<"x[]= ";
    mostrar_vector(x,5);//Muestra: 12 81 67 23 58
    cout<<"y[]= ";
    mostrar_vector(y,5);//Muestra: 1 36 28 102 45

    vector<int>v1(7,12), v2(10,25);
    //Intercambiando el 4to elemento de v1 por el 6to elemento de v2
    iter_swap(v1.begin()+3,v2.begin()+5);
    cout<<"vector v1: ";
    mostrar_vector(v1);//Muestra: 12 12 12 25 12 12 12
```

```
    cout<<"vector v2: ";
    mostrar_vector(v2);//Muestra: 25 25 25 25 25 12 25 25 25 25

    //Intercambiando el 3er elemento de x por el 4to elemento de v1
    iter_swap(x+2,v1.begin()+3);
    cout<<"x[]= ";
    mostrar_vector(x,5);//Muestra: 12 81 25 23 58
    cout<<"vector v1: ";
    mostrar_vector(v1);//Muestra: 12 12 12 67 12 12 12
    return 0;
}
```

`swap(A,B)`	Donde A y B pueden ser variables, estructuras, objetos, arrays o contenedores STL. La función swap intercambia el valor de A y B. En el caso de que A y B fueran arrays[], deben tener la misma longitud de elementos, requisito que no es necesario si A y B fueran contenedores STL.
`swap_ranges(it1A,it2A,itB)`	Donde A y B son estructuras, arrays o contenedores STL, it1A, it2A e itB son iteradores o punteros. La función swap_range toma los datos de A desde la posición de itA1 hasta itA2 y los intercambia por la misma cantidad de datos de B a partir de la posición de itB. En el caso de que A y B fueran arrays[], deben tener la misma longitud de elementos, requisito que no es necesario si A y B fueran contenedores STL.
`iter_swap(a,b)`	Donde a y b son iteradores o punteros. La función iter_swap intercambia los datos que se encuentran en la posición a y b. Es decir que el dato que se encuentra en la posición a ocupara la posición b, asi mismo el dato que se encuentra en la posición b, ocupara la posición a. Los punteros o iteradores pueden pertenecer a distintas estructuras o ambos pueden pertenecer a la misma.

7.2.3 ALGORITMOS REPLACE

```
//ALGORITMO REPLACE Y REPLACE_IF
#include<iostream>
#include<algorithm>
using namespace std;
void mostrar_vector(int v[],int m){
    for(int i=0;i<m;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
bool es_impar(int num){return (num&1);}
int main(){
    int x[]={25,11,16,11,8,14,11};

    cout<<"FUNCION REPLACE"<<endl;
    replace(x,x+7,11,15);//Remplazando el numero 11 por el numero 15
    cout<<"x[]= ";
    mostrar_vector(x,7);//Muestra: 25 15 16 15 8 14 15

    cout<<"\nFUNCION REPLACE IF"<<endl;
    replace_if(x,x+7,es_impar,24);//Remplazando los numeros impares por el numero 24
    cout<<"x[]= ";
    mostrar_vector(x,7);//Muestra: 24 24 16 24 8 14 24
    return 0;
}
```

```
//ALGORITMO REPLACE_COPY Y REPLACE_COPY_IF
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
```

```
void mostrar_vector(int v[],int m){
    for(int i=0;i<m;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
void mostrar_vector(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
bool es_impar(int num){return (num&1);}
int main(){
    int x[]={25,11,16,11,8,14,11};
    vector<int>v1(7),v2(7);
    //FUNCION REPLACE_COPY
    replace_copy(x,x+8,v1.begin(),11,15);//Remplazando el numero 11 por el numero 15
    //FUNCION REPLACE_COPY_IF
    replace_copy_if(x,x+8,v2.begin(),es_impar,24);//Remplazando números impares por 24
    //MOSTRANDO LOS DATOS
    cout<<"x[]= ";
    mostrar_vector(x,7);//Muestra: 25 11 16 11 8 14 11
    cout<<"vector v1: ";
    mostrar_vector(v1);//Muestra: 25 15 16 15 8 14 15
    cout<<"vector v2: ";
    mostrar_vector(v2);//Muestra: 24 24 16 24 8 14 24
    return 0;
}
```

`replace(it1,it2,dato1,dato2)`	Donde it1 e it2 son iteradores o punteros, la función replace busca los dato1 desde la posición de it1 hasta la posición de it2. Si dato1 esta en la estructura los reemplaza por dato2.
`replace_if(it1,it2,f(n),dato)`	Donde it1 e it2 son iteradores o punteros, f(n) es una función de condición, la función replace_if busca los datos que cumplen la condición f(n) desde la posición de it1 hasta la posición de it2 y los reemplaza por el nuevo dato.
`replace_copy(itA1,itA2,itB, dato1, dato2)`	Donde itA1 e itA2 son iteradores o punteros de A e itB es puntero o iterador de B, la función replace_copy busca los dato1 desde la posición de itA1 hasta la posición de itA2. Si dato1 esta en la estructura los reemplaza por dato2 y almacena el resultado en B, en la posición de itB.
`replace_copy_if(itA1,itA2,itB, f(n),dato)`	Donde itA1 e itA2 son iteradores o punteros, f(n) es una función de condición e itB es puntero o iterador de B, la función replace_copy_if busca los datos que cumplen la condición f(n) desde la posición de itA1 hasta la posición de itA2 y los reemplaza por el nuevo dato y almacena el resultado en B, en la posición de itB.

7.2.4 ALGORITMOS FILL

```
#include<iostream>
#include<algorithm>
using namespace std;
void mostrar(int v[],int m){
    for(int i=0;i<m;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    const int n=10;
    int x[n];
    //ALGORITMO FILL
    fill(x,x+n,0);//Llenando todos los espacios con cero
```

```
    mostrar(x,n);//Muestra: 0 0 0 0 0 0 0 0 0 0
    fill(x,x+5,2);//Llenando los cinco primeros espacios con 5
    mostrar(x,n);//Muestra: 2 2 2 2 2 0 0 0 0 0
    //ALGORITMO FILL_N
    fill_n(x+2,5,8);//Llenando 5 espacios con 8 a partir de la posicion 2
    mostrar(x,n);//Muestra: 2 2 8 8 8 8 8 0 0 0
    return 0;
}
```

`fill(it1,it2,dato)`	Donde it1 e it2 son iteradores o punteros, la función fill llena los espacios con un dato, desde la posición it1 hasta la posición it2.
`fill_n(it,n,dato)`	Donde it es iterador o puntero, la función fill_n llena los n espacios con un dato, desde la posición it.

7.2.5 ALGORITMOS GENERATE

```
#include<iostream>
#include<algorithm>
using namespace std;
int num=1;
void mostrar(int v[],int m){
    for(int i=0;i<m;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int numero(){return num+=2;}
int main(){
    const int n=10;
    int x[n];
    //ALGORITMO GENERATE
    generate(x,x+n,numero);
    mostrar(x,n);//Muestra: 3 5 7 9 11 13 15 17 19 21
    num=0;
    //ALGORITMO GENERATE_N
    generate_n(x+3,5,numero);
    mostrar(x,n);//Muestra: 3 5 7 2 4 6 8 10 19 21
    return 0;
}
```

`generate(it1,it2,f(n))`	Donde it1 e it2 son iteradores o punteros y f(n) es una función, la función generate genera los espacios con f(n) desde la posición it1 hasta la posición it2.
`generate_n(it,n,f(n))`	Donde it es iterador o puntero, la función generate_n genera los n espacios con f(n), desde la posición it.

7.2.6 ALGORITMOS REMOVE

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
bool es_impar(int i){return (i&1);}
int main(){
    int x[]={10,58,10,13,4,10,2,49};
    vector<int>::iterator ini,fin;
    //ALGORITMO REMOVE
    vector<int>vec1(x,x+8);
    ini=vec1.begin();
    fin=remove(vec1.begin(),vec1.end(),10);
    while(ini!=fin){
```

```
        cout<<*ini<<" ";
        ini++;
    }//Muestra: 58 13 4 2 49
    cout<<endl;
    //ALGORITMO REMOVE_IF
    vector<int>vec2;
    for(int i=1;i<=10;i++)
        vec2.push_back(i);
    ini=vec2.begin();
    fin=remove_if(vec2.begin(),vec2.end(),es_impar);
    while(ini!=fin){
        cout<<*ini<<" ";
        ini++;
    }//Muestra: 2 4 6 8 10
    cout<<endl;
    return 0;
}
```

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
void mostrar(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
bool es_impar(int i){return (i&1);}
int main(){
    int x[]={10,58,10,13,4,10,2,49};
    int y[]={25,16,7,11,2,14,19,10};
    vector<int>v1(8),v2(8);
    //ALGORITMO REMOVE_COPY
    remove_copy(x,x+8,v1.begin(),10);
    mostrar(v1);//Muestra: 58 13 4 2 49 0 0 0
    //ALGORITMO REMOVE_COPY_IF
    remove_copy_if(y,y+8,v2.begin()+3,es_impar);
    mostrar(v2);//Muestra: 0 0 0 16 2 14 10 0
    return 0;
}
```

`remove(it1,it2,dato)`	Donde it1 e it2 son iteradores o punteros, la función remove transforma el rango (it1, it2) en una secuencia que no contiene el dato y devuelve un iterador donde marca el final de la secuencia.
`remove_if(it1,it2,f(n))`	it1 e it2 son iteradores o punteros y f(n) es una funcion, la función remove_if transforma el rango (it1, it2) en una secuencia que no contiene f(n) y devuelve un iterador donde marca el final de la secuencia.
`remove_copy(itA1,itA2,itB,dato)`	Donde itA1 e itA2 son iteradores o punteros de la estructura A, itB es un iterador o puntero de B, la función remove_copy transforma el rango (itA1, itA2) en una secuencia que no contiene el dato. Dicha secuenia se copia en la posición itB de B.
`remove_copy_if(itA1,itA2,itB,f(n))`	Donde it es iterador o puntero y f(n) es una funcion, la función remove_copu_if transforma el rango (it1, it2) en un rango que no contiene f(n). Dicha secuenia se copia en la posición itB de B.

7.2.7 ALGORITMOS UNIQUE

```
//ALGORITMO UNIQUE
#include<iostream>
```

```
#include<vector>
#include<algorithm>
using namespace std;
bool compara(int i,int j){return i+10==j;}
int main(){
    int vec1[] = {10,20,20,20,30,30,20,20,10};
    vector<int>vec2(vec1,vec1+9);

    int* p=unique(vec1,vec1+9);
    for(int* i=vec1;i!=p;i++)
        cout<<*i<<" ";//Muestra: 10 20 30 20 10
    cout<<endl;
    vector<int>::iterator it,pf;

    pf=unique(vec2.begin(),vec2.end(),compara);
    for(it=vec2.begin();it!=pf;it++)
        cout<<*it<<" ";//Muestra: 10 30 30 20 20 10
    return 0;
}
```

```
//ALGORITMO UNIQUE_COPY
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
void mostrar(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
bool compara(int i,int j){return i+10==j;}
int main(){
    int x[] = {10,20,20,20,30,30,20,20,10};
    vector<int>vec1(9);
    vector<int>vec2(9);
    vector<int>::iterator it;

    unique_copy(x,x+9,vec1.begin()+2);
    mostrar(vec1);//Muestra: 0 0 10 20 30 20 10 0 0

    unique_copy(x,x+9,vec2.begin()+3,compara);
    mostrar(vec2);//Muestra: 0 0 0 10 30 30 20 20 10
    return 0;
}
```

`unique(it1,it2)`	Donde it1 e it2 son iteradores o punteros, la función unique elimina los datos repetidos adyacentes (por la izquierda o derecha) enel rango (it1, it2) y devuelve un iterador donde marca el final de la secuencia.
`unique(it1,it2,f(x,y))`	it1 e it2 son iteradores o punteros y f(x, y) es una función. La función unique elimina los datos adyacentes (por la izquierda o derecha) que cumplen la condición f(x, y) en el rango (it1, it2) y devuelve un iterador donde marca el final de la secuencia.
`unique_copy(itA1,itA2,itB)`	Donde itA1 e itA2 son iteradores o punteros de la estructura A, itB es un iterador o puntero de B, la función unique_copy transforma el rango (itA1, itA2) en una secuencia que no contiene datos repetidos adyacentes. Dicha secuenia se copia en la posición itB de B.
`unique_copy(itA1,itA2,itB,f(x,y))`	Donde itA1 e itA2 son iteradores o punteros de la estructura A, itB es un iterador o puntero de B y f(x, y) es una función. La función unique_copy transforma el rango (itA1, itA2) en una secuencia que no contiene datos adyacentes que cumplen la condición f(x, y). Dicha secuenia se copia en la posición itB de B.

7.2.8 ALGORITMOS REVERSE

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
void mostrar(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    vector<int>v1,v2(9);
    for(int i=1;i<10;i++)
        v1.push_back(i);
    //ALGORITMO REVERSE
    reverse(v1.begin(),v1.end());
    mostrar(v1);//Muestra: 9 8 7 6 5 4 3 2 1

    //ALGORITMO REVERSE_COPY
    reverse_copy(v1.begin(),v1.end(),v2.begin());
    mostrar(v2);//Muestra: 1 2 3 4 5 6 7 8 9
    return 0;
}
```

`reverse(it1,it2)`	Donde it1 e it2 son iteradores o punteros, la función reverse invierte el contenido del vector en el rango de (it1,it2) .
`reverse_copy(itA1,itA2,itB)`	Donde itA1 e itA2 son iteradores o punteros de A, itB es iterador o puntero de B. La función la función reverse_copy invierte el contenido del vector en el rango de (it1,it2) y lo almacena en la posición de itB2 de B.

7.2.9 ALGORITMOS ROTATE

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
void mostrar(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    vector<int>v1,v2(9);
    for(int i=1;i<10;i++)
        v1.push_back(i);
    //ALGORITMO ROTATE
    rotate(v1.begin(),v1.begin()+3,v1.end());
    mostrar(v1);//Muestra: 4 5 6 7 8 9 1 2 3

    //ALGORITMO ROTATE_COPY
    rotate_copy(v1.begin(),v1.begin()+3,v1.end(),v2.begin());
    mostrar(v2);//Muestra: 7 8 9 1 2 3 4 5 6
    return 0;
}
```

`rotate(it1,it2,it3)`	Donde it1, it2 e it3 son iteradores o punteros, la función rotate rota el contenido de A desde el rango de (it1,it2) hasta it3 .
`rotate(itA1,itA2,itA3,itB)`	Donde itA1, itA2 e itA3 son iteradores o punteros de A, itB es un puntero o iterador de B, la función rotate rota el contenido del A desde el rango de (it1,it2) hasta it3 y lo almacena en la posición itB en B.

7.2.10 ALGORITMO RANDOM SHUFFLE

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
void mostrar(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int func(int i) { return std::rand()%i;}
int main(){
    vector<int>v1;
    int w=10;
    for(int i=1;i<10;i++)
        v1.push_back(i);
    random_shuffle(v1.begin(),v1.end());//Acomoda los datos aletoriamente
    mostrar(v1);
    random_shuffle(v1.begin(),v1.end(),func);//Acomoda los datos aletoriamente
    mostrar(v1);
    return 0;
}
```

`ramdom_shuffle(it1,it2)`	Donde it1 e it2 son iteradores o punteros, la función random_shuffle acomoda los datos aleatoriamente de la estructura en el rango de (it1,it2).
`ramdom_shuffle(it1,it2,f(n))`	Donde it1 e it2 son iteradores o punteros y f(n) es una funcion, la función random_shuffle acomoda los datos acorde a la función f(n), de la estructura en el rango de (it1,it2).

7.3 ALGORITMOS DE PARTICION

7.3.1 ALGORITMO PARTITION

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
bool es_impar(int i){return (i&1);}
int main(){
    vector<int>vec;
    for(int i=1;i<=10;i++)
        vec.push_back(i);//Cargamos el vector son numeros del 1 al 10
    vector<int>::iterator particion,it;
    particion=partition(vec.begin(),vec.end(),es_impar);

    cout<<"Numeros impares:"<<endl;
    for(it=vec.begin();it!=particion;it++)
        cout<<*it<<" ";//Muestra: 1 9 3 7 5
    cout<<endl;

    cout<<"Numeros pares:"<<endl;
    for(it=particion;it!=vec.end();it++)
        cout<<*it<<" ";//Muestra: 6 4 8 2 10
    return 0;
}
```

`partition(it1,it2,f(n))`	Donde it1 e it2 son iteradores o punteros y f(n) es una funcion, la función partition acomoda en las primeras posiciones los datos que cumplen la función f(n) en el rango de (it1,it2).

7.3.2 ALGORITMO STABLE PARTITION

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
bool es_impar(int i){return (i&1);}
int main(){
    vector<int>vec;
    for(int i=1;i<=10;i++)
        vec.push_back(i);//Cargamos el vector son numeros del 1 al 10
    vector<int>::iterator particion,it;
    particion=stable_partition(vec.begin(),vec.end(),es_impar);

    cout<<"Numeros impares:"<<endl;
    for(it=vec.begin();it!=particion;it++)
        cout<<*it<<" ";//Muestra: 1 3 5 7 9
    cout<<endl;

    cout<<"Numeros pares:"<<endl;
    for(it=particion;it!=vec.end();it++)
        cout<<*it<<" ";//Muestra: 2 4 6 8 10
    return 0;
}
```

`stable_partition(it1,it2,f(n))`	Donde it1 e it2 son iteradores o punteros y f(n) es una funcion, la función stable_partition acomoda en las primeras posiciones los datos que cumplen la función f(n) en el rango de (it1,it2). A diferencia de partition, stable_partition, acomoda los datos según el orden en el que estaban los datos inicialmente.

7.4 ALGORITMOS DE ORDENAMIENTO

7.4.1 ALGORITMO SORT

```
#include<iostream>
#include<algorithm>
using namespace std;
bool mayor_a_menor(int i,int j){return (i>j);}
struct micomp{    bool operator() (int i,int j) {return (i<j);} }objeto;
void mostrar(int v[],int n){
    for(int i=0;i<n;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    int vec[] = {32,71,12,45,26,80,33,53};
    sort(vec,vec+4);//Ordena las cuatro primeras posiciones ascendentemente
    mostrar(vec,8);//Muestra: 12 32 45 71 26 80 33 53
    sort(vec+4,vec+8,mayor_a_menor);//Ordena las 4 ultimas posiciones descendentemente
    mostrar(vec,8);//Muestra: 12 32 45 71 80 53 33 26
    sort(vec,vec+8,objeto);//Ordena todas las posiciones del vector ascendentemente
    mostrar(vec,8);//Muestra: 12 26 32 33 45 53 71 80
    return 0;
}
```

`sort(it1,it2)`	Donde it1 e it2 son iteradores o punteros. La función sort ordena ascendentemente los elementos del vector en el rango (it1,it2).
`sort(it1,it2,f(n))`	Donde it1 e it2 son iteradores o punteros y f(n) es una funcion. La función sort ordena elementos del vector en el rango (it1,it2), acorde a las codiciones de f(n).
`sort(it1,it2,A)`	Donde it1 e it2 son iteradores o punteros y A es un objeto. La función sort ordena elementos del vector en el rango (it1,it2) acorde a las codiciones de A.

7.4.2 ALGORITMO STABLE SORT

```
#include<iostream>
#include<algorithm>
using namespace std;
bool comp_enteros(double i,double j){return ((int)i<(int)j);}
void mostrar(double v[],int n){
    for(int i=0;i<n;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    double vec1[] = {3.14, 1.41, 2.72, 4.67, 1.73, 1.32, 1.62, 2.58};
    double vec2[sizeof(vec1)];
    copy(vec1,vec1+8,vec2);
    stable_sort(vec1,vec1+8);//Ordena las cuatro primeras posiciones ascendentemente
    mostrar(vec1,8);//Muestra: 1.32 1.41 1.62 1.73 2.58 2.72 3.14 4.67
    stable_sort(vec2,vec2+8,comp_enteros);
    mostrar(vec2,8);//Muestra: 1.41 1.73 1.32 1.62 2.72 2.58 3.14 4.67
    return 0;
}
```

`stable_sort(it1,it2)`	Donde it1 e it2 son iteradores o punteros. La función stable_sort ordena ascendentemente los elementos del vector en el rango (it1,it2).
`stable_sort(it1,it2,f(n))`	Donde it1 e it2 son iteradores o punteros y f(n) es una funcion. La función stable_sort ordena elementos del vector en el rango (it1,it2), acorde a las codiciones de f(n).

7.4.3 ALGORITMO QSORT (LIBRERÍA CSTDLIB)

```
#include<iostream>
#include<cstdlib>
using namespace std;
int comparar(const void *a,const void *b){return ( *(int*)a - *(int*)b );}
void mostrar(int v[],int n){
    for(int i=0;i<n;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    int vec[] = {32,71,12,45,26,80,33,53};
    qsort(vec,8,sizeof(int),comparar);
    mostrar(vec,8);//12 26 32 33 45 53 71 80
    return 0;
}
```

```
//Ordenando cadenas
#include<iostream>
#include<cstdlib>
#include<cstring>
using namespace std;
void mostrar(char v[][20],int n){
    for(int i=0;i<n;i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    char vec[][20] = {"este","es","un","ejemplo","de","cadenas"};
    qsort(vec,6,20,(int(*)(const void*,const void*))strcmp);
    mostrar(vec,6);//cadenas de ejemplo es este un
    return 0;
}
```

`qsort(it,n,size,f(n))`	Donde: **it** es iterador o puntero. **n** es un entero positivo que representa el numero de elementos. **size** es el tamaño de bits del tipo de variable de los datos. **f(n)** es la función de condición. La función qsort ordena ascendentemente los elementos del vector.

7.5 ALGORITMOS DE BUSQUEDA

7.5.1 ALGORITMOS LOWER BOUND Y UPPER BOUND

```
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    int vec[] = {10,20,30,30,20,10,10,20};
    sort(vec,vec+8);// 10 10 10 20 20 20 30 30
    int *low,*up;
    low=lower_bound(vec,vec+8,20);
    up=upper_bound(vec,vec+8,20);
    cout<<"Lower bound de 20 esta en la posicion: "<<(low-vec)<<endl;//Muestra: 3
    cout<<"Upper bound de 20 esta en la posicion: "<<(up-vec)<<endl;//Muestra: 6
    return 0;
}
```

`lower_bound(it1,it2,dato)`	Donde it1 e it2 son iteradores o punteros. La función lower_bound busca la primera aparición del dato en el rango (it1,it2). Debemos ordenar la estructura antes.
`upper_bound(it1,it2,dato)`	Donde it1 e it2 son iteradores o punteros. La función lower_bound busca la ultima aparición del dato en el rango (it1,it2). Debemos ordenar la estructura antes.

7.5.2 ALGORITMO EQUAL RANGE

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main(){
    int x[] = {10,20,30,30,20,10,10,20};
    vector<int> v(x,x+9);
    pair<vector<int>::iterator,vector<int>::iterator>rango;
    sort(v.begin(),v.end());//10 10 10 20 20 20 30 30
    rango=equal_range(v.begin(),v.end(),20);
    cout<<"Primera posicion del rango: "<<rango.first-v.begin()<<endl;//Muestra: 3
    cout<<"Segunda posicion del rango: "<<rango.second-v.begin()<<endl;//Muestra: 6
    return 0;
}
```

`equal_range(it1,it2,dato)`	Donde it1 e it2 son iteradores o punteros. La función equal_range busca el rango en el que se encuentra una sucesión de datos repetidos. Devuelve un par que contiene la posición inicial y la posición final. Ordenamos la estructura para la búsqueda.

7.5.3 ALGORITMO BINARY SEARCH

```
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    int x[] = {1,2,3,4,5,4,3,2,1};
    sort(x,x+9);//1 1 2 2 3 3 4 4 5
```

```
    cout<<"Buscando el dato 3: ";
    if(binary_search(x,x+9,3))
        cout<<"Se encontro el dato!!"<<endl;
    else
        cout<<"No se encuentra el dato"<<endl;
    //Muestra: Se encontro el dato!!

    cout<<"Buscando el dato 6: ";
    if(binary_search(x,x+9,6))
        cout<<"Se encontro el dato!!"<<endl;
    else
        cout<<"No se encuentra el dato"<<endl;
    //Muestra: No se encuentra el dato
    return 0;
}
```

`binary_search(it1,it2,dato)`	Donde it1 e it2 son iteradores o punteros. La función binary_search devuelve true si encuentra el dato en el rango it1 e it2. Es necesario que el vector o contenedor este ordenado.

7.6 ALGORITMOS MERGE

7.6.1 ALGORITMO MERGE

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main(){
    int x[] = {5,10,15,20,25};
    int y[] = {50,40,30,20,10};
    vector<int>v(10);
    sort(x,x+5);//5 10 15 20 25
    sort(y,y+5);//10 20 30 40 50
    merge(x,x+5,y,y+5,v.begin());
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";//Muestra: 5 10 10 15 20 20 25 30 40 50
    cout<<endl;
    return 0;
}
```

`merge(itA1,itA2,itB1,itB2,itC)`	Donde itA1 e itA2 son iteradores o punteros de A, itB1 e itAB2 son iteradores o punteros de B e itC es un puntero o iterador de C. La función merge combina en orden el contenido de A y B en el rango de (itA1, itA2) y (itB1, itB2) respectivamente, y lo coloca en la posición de itC de C.

7.6.2 ALGORITMO INCLUDES

```
#include<iostream>
#include<algorithm>
using namespace std;
bool compara(int i,int j){return i==j+10;}
int main(){
    int x[] = {5,10,15,20,25,30,35,40,45,50};
    int y[] = {40,30,20,10};
    sort(x,x+10);//5 10 15 20 25 30 35 40 45 50
    sort(y,y+4);//10 20 30 40
    if(includes(x,x+10,y,y+4))
        cout<<"X contiene a Y"<<endl;
    else
        cout<<"X NO contiene a Y"<<endl;
```

```
    //Muestra: X contiene a Y

    if(includes(x,x+10,y,y+4,compara))
        cout<<"X contiene a Y"<<endl;
    else
        cout<<"X NO contiene a Y"<<endl;
    //Muestra: X NO contiene a Y
    return 0;
}
```

includes(itA1,itA2,itB1,itB2)	Donde itA1 e itA2 son iteradores o punteros de A, itB1 e itAB2 son iteradores o punteros de B. La función includes devuelve true (verdadero) si encuentra el contenido de B en el rango (it1B, itB2) en la secuencia de (itA1, itA2) de A.

7.6.3 ALGORITMO DE OPERACIONES DE CONJUNTOS.

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
void mostrar(vector<int> v){
    for(int i=0;i<v.size();i++)
        cout<<v[i]<<" ";
    cout<<endl;
}
int main(){
    int x[] = {5,10,15,20,25};
    int y[] = {50,40,30,20,10};
    vector<int>U(10),I(10),D(10),DS(10);
    sort(x,x+5);//5 10 15 20 25
    sort(y,y+5);//10 20 30 40 50
    vector<int>::iterator op;

    cout<<"UNION"<<endl;
    op=set_union(x,x+5,y,y+5,U.begin());
    mostrar(U);//Muestra: 5 10 15 20 25 30 40 50 0 0
    U.resize(op-U.begin());
    mostrar(U);//Muestra: 5 10 15 20 25 30 40 50

    cout<<"\nINTERSECCION"<<endl;
    op=set_intersection(x,x+5,y,y+5,I.begin());
    mostrar(I);//Muestra: 10 20 0 0 0 0 0 0 0 0
    I.resize(op-I.begin());
    mostrar(I);//Muestra: 10 20

    cout<<"\nDIFERENCIA"<<endl;
    op=set_difference(x,x+5,y,y+5,D.begin());
    mostrar(D);//Muestra: 5 15 25 0 0 0 0 0 0 0
    D.resize(op-D.begin());
    mostrar(D);//Muestra: 5 15 25

    cout<<"\nDIFERENCIA SIMETRICA"<<endl;
    op=set_symmetric_difference(x,x+5,y,y+5,DS.begin());
    mostrar(DS);//Muestra: 5 15 25 30 40 50 0 0 0 0
    DS.resize(op-DS.begin());
    mostrar(DS);//Muestra: 5 15 25 30 40 50
    return 0;
}
```

set_union(itA1,itA2,itB1,itB2,itC)	Donde itA1 e itA2 son iteradores o punteros de A, itB1 e itAB2 son iteradores o punteros de B e itC es un puntero o iterador de C. La función set_union realiza la unión de A y B en el rango de (itA1, itA2) y

	(itB1, itB2) respectivamente, y lo coloca en la posición de itC de C.
`set_interseccion(itA1,itA2,itB1, itB2,itC)`	Donde itA1 e itA2 son iteradores o punteros de A, itB1 e itAB2 son iteradores o punteros de B e itC es un puntero o iterador de C. La función set_intersection realiza la intersección de A y B en el rango de (itA1, itA2) y (itB1, itB2) respectivamente, y lo coloca en la posición de itC de C.
`set_difference(itA1,itA2,itB1, itB2,itC)`	Donde itA1 e itA2 son iteradores o punteros de A, itB1 e itAB2 son iteradores o punteros de B e itC es un puntero o iterador de C. La función set_difference realiza la diferencia de A y B en el rango de (itA1, itA2) y (itB1, itB2) respectivamente, y lo coloca en la posición de itC de C.
`set_symmetric_difference(itA1, itA2, itB1, itB2,itC)`	Donde itA1 e itA2 son iteradores o punteros de A, itB1 e itAB2 son iteradores o punteros de B e itC es un puntero o iterador de C. La función set_symmetric_difference realiza la diferencia simetrica de A y B en el rango de (itA1, itA2) y (itB1, itB2) respectivamente, y lo coloca en la posición de itC de C.

7.7 ALGORITMOS HEAP

7.7.1 PUSH HEAP, MAKE HEAP, Y POP HEAP

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main(){
    int x[]={10,5,30,20,15};
    vector<int>v(x,x+5);
    cout<<"Primer elemento: "<<v.front()<<endl;//Muestra: 10

    cout<<"\nMAKE HEAP"<<endl;
    make_heap(v.begin(),v.end());
    cout<<"Primer elemento: "<<v.front()<<endl;//Muestra: 30

    cout<<"\nPOP HEAP"<<endl;
    pop_heap(v.begin(),v.end());
    cout<<"Primer elemento: "<<v.front()<<endl;//Muestra: 20

    cout<<"\nPUSH HEAP"<<endl;
    v.push_back(50);
    push_heap(v.begin(),v.end());
    cout<<"Primer elemento: "<<v.front()<<endl;//Muestra: 50
    return 0;
}
```

`make_heap(it1,it2)`	Donde it1 e it2 son iteradores o punteros. La función make_heap organiza los elementos de una estructura en el rango (it1, it2), de tal manera que se pueda obtener rápidamente el elemento de mayor valor el cual es situado en la primera posición.
`pop_heap(it1,it2`	Donde it1 e it2 son iteradores o punteros. La función pop_heap organiza los elementos de una estructura en el rango (it1, it2), de tal manera que el elemento de mayor valor es colocado a la posición final. Despues el segundo mayor valor es situado al principio de la estructura.
`push_heap(it1,it2)`	Donde it1 e it2 son iteradores o punteros. La función pop_heap organiza los elementos de una estructura en el rango (it1, it2), de tal manera que un nuevo dato pueda ser incluido en la organización heap.

7.8 ALGORITMOS MIN/MAX

7.8.1 MIN Y MAX

```
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    cout<<"MAX"<<endl;
    cout<<"El mayor de 5 y 7 es: "<<max(5,7)<<endl;//Muestra: 7
    cout<<"El mayor de 2.56 y 2.55 es: "<<max(2.56,2.55)<<endl;//Muestra: 2.56

    cout<<"MIN"<<endl;
    cout<<"El menor de 5 y 7 es: "<<min(5,7)<<endl;//Muestra: 5
    cout<<"El menor de 2.56 y 2.55 es: "<<min(2.56,2.55)<<endl;//Muestra: 2.55
    return 0;
}
```

`max(dato1,dato2)`	Compara y devuelve el dato que sea de mayor valor.
`min(dato1,dato2)`	Compara y devuelve el dato que sea de menor valor.

7.8.2 MIN ELEMENT Y MAX ELEMENT

```
#include<iostream>
#include<algorithm>
using namespace std;
bool comp_impar(int i,int j){return(i<j && (i&1));}
int main(){
    int v[] = {3,7,2,5,6,4,9};
    cout<<"MIN ELEMENT"<<endl;
    cout<<"Menor elemento : "<<*min_element(v,v+7)<<endl;//Muestra: 2
    cout<<"Menor elemento impar: "<<*min_element(v,v+7,comp_impar)<<endl;//Muestra: 3

    cout<<"MAX ELEMENT"<<endl;
    cout<<"Mayor elemento es: "<<*max_element(v,v+7)<<endl;//Muestra: 9
    cout<<"Mayor elemento impar: "<<*max_element(v,v+7,comp_impar)<<endl;//Muestra: 9
    return 0;
}
```

`max_element(it1, it2)`	Donde it1 e it2 son iteradores o punteros. La función max_element busca la el elemento de mayor valor en el rango (it1, ti2).
`max_element(it1, it2,f(x,y))`	Donde it1 e it2 son iteradores o punteros y f(x,y) es una función. La función max_element busca la el elemento de mayor valor en el rango (it1, ti2) según la condición de f(x,y).
`min_element(it1, it2)`	Donde it1 e it2 son iteradores o punteros. La función max_element busca la el elemento de mayor valor en el rango (it1, ti2).
`min_element(it1, it2,f(x,y))`	Donde it1 e it2 son iteradores o punteros y f(x,y) es una función. La función max_element busca la el elemento de mayor valor en el rango (it1, ti2) según la condición de f(x,y).

7.9 ALGORITMOS DE ORDEN LEXICOGRAFICO

7.9.1 LEXICOGRAFICAL COMPARE

```
#include<iostream>
#include<algorithm>
using namespace std;
bool compara(char c1,char c2){return (tolower(c1)<tolower(c2));}
int main(){
```

```
    string cad1="Apellido", cad2="apartamento";
    if(lexicographical_compare(cad1.begin(),cad1.end(),cad2.begin(),cad2.end()))
        cout<<"La cadena 'Apellido' es menor que la cadena 'apartamento'"<<endl;
    else
        cout<<"La cadena 'apartamento' es menor que la cadena 'Apellido'"<<endl;
    //Muestra: La cadena 'Apellido' es menor que la cadena 'apartamento'

   if(lexicographical_compare(cad1.begin(),cad1.end(),cad2.begin(),cad2.end(),compara))
        cout<<"La cadena 'Apellido' es menor que la cadena 'apartamento'"<<endl;
    else
        cout<<"La cadena 'apartamento' es menor que la cadena 'Apellido'"<<endl;
    //Muestra: La cadena 'apartamento' es menor que la cadena 'Apellido'
    return 0;
}
```

`lexicographical_compare(itA1,itA2, itB1,itB2)`	Donde itA1 e itA2 son iteradores o punteros de A, itB1 e itB2 son iteradores o punteros de B. La función lexicographical_compare compara en el rango (itA1, itA2) de A, con la secuencia del rango (itB1,itB2) de B. Devuelve **true** (verdadero) si A es menor que B.
`lexicographical_compare(itA1,itA2, itB1,itB2,f(x,y))`	Donde itA1 e itA2 son iteradores o punteros de A, itB1 e itB2 son iteradores o punteros de B y f(x,y) es una función. La función lexicographical_compare compara acorde la condición f(x,y) en el rango (itA1, itA2) de A, con la secuencia del rango (itB1,itB2) de B. Devuelve **true** (verdadero) si A es menor que B.

7.9.2 ALGORITMO NEXT_PERMUTATION

```
#include<iostream>
#include<algorithm>
using namespace std;
bool compara(char c1,char c2){return (tolower(c1)<tolower(c2));}
int main(){
    string cad="baA";
    sort(cad.begin(),cad.end());//Aab

    do{
        cout<<cad<<endl;
    }
    while(next_permutation(cad.begin(),cad.end()));

    cout<<"\nUtilizando la funcion compara"<<endl;
    sort(cad.begin(),cad.end());
    do{
        cout<<cad<<endl;
    }
    while(next_permutation(cad.begin(),cad.end(),compara));
    return 0;
}
```

`La primera permutacion simple muestra:`
`Aab`
`Aba`
`aAb`
`abA`
`bAa`
`baA`
	`La segunda permutacion con la función compara muestra:`
	`Aab`
	`Aba`
	`baA`

`next_permutation(it1,it2)`	Donde it1 e it2 son iteradores o punteros. La función next_permutation acomoda los elementos de una estructura según el orden de la siguiente permutación.
`next_permutation(it1,it2,f(x,y))`	Donde it1 e it2 son iteradores o punteros y f(x,y) es una función de

	comparación. La función next_permutation acomoda los elementos de una estructura según el orden de la siguiente permutación y acorde a la condición de f(x, y).

7.9.3 PREV PERMUTATION

```
#include<iostream>
#include<algorithm>
using namespace std;
bool compara(char c1,char c2){return (tolower(c1)<tolower(c2));}
int main(){
    string cad="bAa";
    sort(cad.begin(),cad.end());//Aab
    reverse(cad.begin(),cad.end());//baA

    do{
        cout<<cad<<endl;
    }
    while(prev_permutation(cad.begin(),cad.end()));

    cout<<"\nUtilizando la funcion compara"<<endl;
    sort(cad.begin(),cad.end());
    reverse(cad.begin(),cad.end());//baA
    do{
        cout<<cad<<endl;
    }
    while(prev_permutation(cad.begin(),cad.end(),compara));
    return 0;
}
```

La primera permutacion simple muestra: baA bAa abA aAb Aba Aab	La segunda permutacion con la función compara muestra: baA Aba Aab

`prev_permutation(it1,it2)`	Donde it1 e it2 son iteradores o punteros. La función prev_permutation acomoda los elementos de una estructura según el orden de la anterior permutación.
`prev_permutation(it1,it2,f(x,y))`	Donde it1 e it2 son iteradores o punteros y f(x,y) es una función de comparación. La función prev_permutation acomoda los elementos de una estructura según el orden de la anterior permutación y acorde a la condición de f(x, y).

8. MISCELANEA

8.1 LIBRERÍA CASSERT (ASSERT.H)

8.1.1 EJEMPLO DE INTERRUPCION DE UNA DIVISION ENTRE CERO

```
#include<iostream>//Libreria Para la lectura e impresion de datos
#include<cassert>//Libreria para las assercciones
using namespace std;
int main(){
    int a,b;
    cin>>a>>b;//Lectura de datos
    assert(b!=0);//La division no debe ser entre cero
    cout<<a/b<<endl;//Si se cumple la asserccion procede la division
    return 0;
}
```

```
La ejecución del programa se interrumpe mientras no se cumpla la assercion b!=0
```

8.2 LIMITES DE VARIABLES

8.2.1 CLIMITS (LIMITS.H)

```
#include<iostream>//Libreria Para la lectura e impresion de datos
#include<climits>//Libreria para los limites
using namespace std;
int main(){
    cout<<"Valor de un bit: "<<CHAR_BIT<<endl;//8
    cout<<"Minimo valor de un char: "<<CHAR_MIN<<endl;//-128
    cout<<"Maximo valor de un char: "<<CHAR_MAX<<endl;//127
    cout<<"Maximo valor de un unsigned char: "<<UCHAR_MAX<<endl;//255
    cout<<"Minimo valor de un short: "<<SHRT_MIN<<endl;//-32768
    cout<<"Maximo valor de un short: "<<SHRT_MAX<<endl;//32767
    cout<<"Maximo valor de un unsigned short: "<<USHRT_MAX<<endl;//65535
    cout<<"Minimo valor de un int: "<<INT_MIN<<endl;//-2147483648
    cout<<"Maximo valor de un int: "<<INT_MAX<<endl;//2147483647
    cout<<"Maximo valor de un unsigned int: "<<UINT_MAX<<endl;//4294967295
    cout<<"Minimo valor de un long: "<<LONG_MIN<<endl;//-2147483648
    cout<<"Maximo valor de un long: "<<LONG_MAX<<endl;//2147483647
    cout<<"Maximo valor de un unsigned long: "<<ULONG_MAX<<endl;//4294967295
    cout<<"Minimo valor de un long long: "<<LLONG_MIN<<endl;//-9223372036854775808
    cout<<"Maximo valor de un long long: "<<LLONG_MAX<<endl;//9223372036854775807
cout<<"Maximo valor de un unsigned long long:"<<ULLONG_MAX<<endl;//18446744073709551615
    return 0;
}
```

8.2.2 CFLOAT (FLOAT.H)

```
#include<iostream>//Libreria Para la lectura e impresion de datos
#include<cfloat>//Libreria para los limites de relaes
using namespace std;
int main(){
    cout<<"Minimo valor de float: "<<FLT_MIN<<endl;//1.17549 e-38
    cout<<"Maximo valor de float: "<<FLT_MAX<<endl;//3.40282e+38
    cout<<"Minimo valor de double: "<<DBL_MIN<<endl;//2.22507e-308
    cout<<"Maximo valor de double: "<<DBL_MAX<<endl;//1.79769e+308
    cout<<"Minimo valor de long double: "<<LDBL_MIN<<endl;//3.3621e-4932
    cout<<"Maximo valor de long double: "<<LDBL_MAX<<endl;//1.79769e+308
    return 0;
}
```

8.3 ITERADORES

8.3.1 OPERACIONES DE ITERADOR

```
#include<iostream>
#include<list>
#include<iterator>
using namespace std;
int main(){
    list<int>L;
    for(int i=1;i<=10;i++)
        L.push_back(i);//Cargando la lista con numeros del 1 al 10
    list<int>::iterator it=L.begin();//Apunta al inicio de la lista
    cout<<"Primer elemento de la lista: "<<*it<<endl;// 1
    advance(it,4);//Avanzamos cuatro posiciones el iterador
    cout<<"Quinto elemento de la lista: "<<*it<<endl;// 5
    cout<<"Distancia entre el primer y el quinto elemento :<<distance(L.begin(),it);//4
    return 0;
}
```

advance(it,n)	Donde it es un iterador y n es numero entero no negativo. Avanza la posición del iterador n posiciones.
distance(it1,it2)	Donde it1 e it2 son posiciones. La función distance devuelve un entero que representa la distancia entre it1 e it2.

8.3.2 GENERADORES

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<deque>//Libreria para las bicolas
#include<iterator>//Libreria para manejar los iteradores
#include<algorithm>//Libreria para algoritmos STL
using namespace std;
void mostrar(deque<int> bc){
    for(int i=0;i<bc.size();i++)
        cout<<bc[i]<<" ";
    cout<<endl;
}
int main(){
    deque<int>A,B,C,D(2);
    for(int i=1;i<=10;i++)
        A.push_back(i);//Cargando la bicola A con numeros del 1 al 10
    copy(A.begin(),A.end(),back_inserter(B));
    copy(A.begin(),A.end(),front_inserter(C));
    copy(A.begin(),A.end(),inserter(D,D.begin()+1));
    mostrar(B);//Muestra: 1 2 3 4 5 6 7 8 9 10
    mostrar(C);//Muestra: 10 9 8 7 6 5 4 3 2 1
    mostrar(D);//Muestra: 0 1 2 3 4 5 6 7 8 9 10 0
    return 0;
}
```

8.4 LIBRERÍA UTILITY

8.4.1 MAKE_PAIR

```
#include<iostream>//Libreria para la entrada y salida de datos
#include<utility>//Libreria para el manejo de pares
using namespace std;
int main(){
    //CONSTRUCTORES DE PARES
    pair<char,int>par1,par2;//Creando par1 y par 2 vacios
    pair<char,int>par3('B',34);//Creando el par3 con los datos iniciandos
    pair<char,int>par4(par3);//Crea el par4 con el contenido del pa3
```

```
    //ASIGNACION DE DATOS A LOS PARES
    par1=make_pair('A',45);//Asignando un par con la funcion make pair
    par2=par1;//Asignando el par1 a par2

    cout<<"par1: "<<par1.first<<" "<<par1.second<<endl;//Muestra: A 45
    cout<<"par2: "<<par2.first<<" "<<par2.second<<endl;//Muestra: A 45
    cout<<"par3: "<<par3.first<<" "<<par3.second<<endl;//Muestra: B 34
    cout<<"par4: "<<par4.first<<" "<<par4.second<<endl;//Muestra: B 34

    //OPERADORES DE COMPARACION DE PARES
    if(par1==par2)
        cout<<"par1 y par2 son iguales"<<endl;
    else
        cout<<"par1 y par2 son diferentes"<<endl;
    //Muestra: par1 y par2 son iguales

    if(par1!=par3)
        cout<<"par1 y par3 son diferentes"<<endl;
    else
        cout<<"par1 y par3 son iguales"<<endl;
    //Muestra: par1 y par3 son diferentes

    if(par1<par3)
        cout<<"par1 es menor que par3"<<endl;
    else
        cout<<"par1 NO es menor que par3"<<endl;
    //Muestra: par1 es menor que par3

    if(par1>par3)
        cout<<"par1 es mayor que par3"<<endl;
    else
        cout<<"par1 NO es mayor que par3"<<endl;
    //Muestra: par1 NO es mayor que par3
    return 0;
}
```

`pair<tipo_dato,tipo_dato>P`	Crea un par P de dos tipos de datos.
`pair<tipo_dato,tipo_dato>P(dato1,dato2)`	Crea un par P de datos. Este contructor crea el par con los datos inicializados.
`pair<titpo_dato,tipo_dato>PA(PB)`	Crea un par PA de datos con el contenido del par PB.
`PA=PB`	El operador de asignación =, asigna el par PB al par PA.
`make_pair(dato1,dato2)`	Crea un par de datos.
`first`	Accede al primer dato de un par.
`second`	Accede al segundo dato de un par.
`==`	Esta comparación devuelve **true** (verdadrero) si los pares son iguales. Caso contrario la comparación devolverá **false** (falso).
`!=`	Esta comparación devuelve **true** (verdadrero) si los pares son diferentes. . Caso contrario la comparación devolverá **false** (falso).
`>`	Esta comparación devuelve **true** (verdadrero) si el primer par es mayor lexicográficamente al segundo par. Caso contrario devuelve **false** (falso).
`<`	Esta comparación devuelve **true** (verdadrero) si el primer par es menor lexicográficamente al segundo par. Caso contrario devuelve **false** (falso).

>=	Esta comparación devuelve **true** (verdadrero) si el primer par es mayor o igual lexicográficamente al segundo par. Caso contrario devuelve **false** (falso).
<=	Esta comparación devuelve **true** (verdadrero) si el primer par es menor o igual lexicográficamente al segundo par. Caso contrario devuelve **false** (falso).

8.5 PREPROCESADORES

8.5.1 PREPROCESADOR #INCLUDE

```
#include<nombre_archivo>//Llama a las librería estándar
#include"nombre_archivo"//Busca el archivo que fue definido por el propio programador
```

8.5.2 PREPROCESADOR #DEFINE

```
#include<iostream>//Libreria para la entrada/salida de datos
#include<cmath>//Libreria para las funciones matematicas
#define PI 3.14159//Define una constante simbolica
#define AREA_CIRCULO(r) (PI*r*r)//Define una funcion
using namespace std;
int main(){
    double num=45;
    cout<<"coseno en grados: "<<cos(num*PI/180.0)<<endl;//0.707107
    cout<<"Area de un circulo: "<<AREA_CIRCULO(num)<<endl;//6361.72
    return 0;
}
```

El preprocesador sustituye código por un texto que el programdor define, sirve para simplificar la codificación.

8.5.3 COMPILACION CONDICIONAL

```
#include<iostream>
#define x 34
using namespace std;
int main(){
    #if !defined (x) //Pregunta si x no se ha definido
        cout<<"No definido"<<endl;
    #else
        cout<<"Definido"<<endl;
    #endif
    //Muestra: Definido
    return 0;
}
```

```
#include<iostream>
#define x 34
using namespace std;
int main(){
    #ifdef x
        cout<<"Definido"<<endl;
    #else
        cout<<"No definido"<<endl;
    #endif
    //Muestra: Definido

    #ifndef y
        cout<<"No definido"<<endl;
    #else
```

```
        cout<<"Definido"<<endl;
    #endif
    //Muestra: No definido
    return 0;
}
```

El preprocesador condicional pregunta si una constante simbolica fue definida o no mediante #define. Las directivas ifdef y ifndef son abreviaciones de if defined y de if !defined respectivamente. La directiva #else no es mas que el camino por falso. Finalmente #elif es una abreviación de #else if.

8.5.4 OPERADOR

```
#include<iostream>//Libreria para la entrada/salida de datos
#define HOLA(x) cout<<"HOLA "#x<<endl;
using namespace std;
int main(){
    HOLA("fermin");//Muestra: Hola "fermin"
    return 0;
}
```

8.6 TYPEDEF Y SIZEOF

8.6.1 TYPEDEF

```
#include<iostream>
#include<map>
using namespace std;
typedef map<string,int>mapa;
typedef pair<string,int>par;
int main(){
    mapa M;//Contruyendo un mapa
    //Insertando datos
    M.insert(par("Fermin",10));
    M.insert(par("Jose",25));
    M.insert(par("Juan",30));
    //Mostrando datos
    mapa::iterator it;
    for(it=M.begin();it!=M.end();it++)
        cout<<it->first<<" "<<it->second<<endl;
    return 0;
}
```

La palabra **typedef** proporciona un mecanismo para la creación de alias.

8.6.2 SIZE_OF

```
#include<iostream>
using namespace std;
int main(){
    int x[]={12,3,11,2,23,45};
    char y[]="esta es una cadena";
    cout<<"Tamaño en bytes de x: "<<sizeof(x)<<endl;//24
    cout<<"Tamaño en bytes de y: "<<sizeof(y)<<endl;//19
    cout<<"Numero de elementos de x: "<<sizeof(x)/sizeof(int)<<endl;//6
    cout<<"Numero de elementos de y: "<<sizeof(y)<<endl;//19
    return 0;
}
```

Sizeof determina el tamaño de bytes en los que se almacenan los distintos tipos de datos.

8.7 OPERACIONES DE BITS

```
#include<cstdio>
int main(){
    int a=12,b=10;
    printf("%d and %d = %d\n",a,b,a&b);//Muestra: 12 and 10 = 8
    printf("%d or %d = %d\n",a,b,a|b);//Muestra: 12 or 10 = 14
    printf("%d xor %d = %d\n",a,b,a^b);//Muestra: 12 xor 10 = 6
    printf("negacion %d\n",(~b));//Muestra: negacion -11
    printf("%d << %d = %d\n",a,1,a<<1);//Muestra: 12 << 1 = 24
    printf("%d << %d = %d\n",a,2,a<<2);//Muestra: 12 << 2 = 48
    printf("%d << %d = %d\n",a,3,a<<3);//Muestra: 12 << 3 = 96
    printf("%d >> %d = %d\n",a,1,a>>1);//Muestra: 12 >> 1 = 6
    printf("%d >> %d = %d\n",a,2,a>>2);//Muestra: 12 >> 2 = 3
    printf("%d >> %d = %d\n",a,3,a>>3);//Muestra: 12 >> 3 = 1
    return 0;
}
```

8.8 MANEJO DEL TIEMPO EN C++

8.8.1 OBTENER EL TIEMPO EXACTO

```
#include<cstdio>//Libreria para la entrada y salida
#include<ctime>//Libreria para la obtencion del tiempo
int main(){
    time_t tiempo = time(0);
    struct tm *tlocal = localtime(&tiempo);
    char output[128];
    strftime(output,128,"%d/%m/%y %H:%M:%S",tlocal);
    printf("%s\n",output);//Muestra: 20/12/13 11:40:06
    return 0;
}
```

8.8.2 CALCULAR EL TIEMPO DE EJECUCION DE UN PROGRAMA

```
//Programa que genera numeros primos menores a 99999. (Fuerza Bruta)
#include<cstdio>
#include<ctime>
#define MAX 99999
int main(){
    clock_t t;
    t=clock();
    int primo=2;
    int div=2;
    while(primo<MAX){
        if(primo%div==0){
            if(div>(primo>>1))
                printf("%d, ",primo);
            primo++;
            div=2;
        }
        else
            div++;
    }
    t = clock() - t;
    printf ("\nTarda %d clicks (%f segundos).\n",t,((float)t)/CLOCKS_PER_SEC);
//MUESTRA: Tarda 5953 clicks (5.953000 segundos).
    return 0;
}
```

```
//Programa que genera numeros primos menores a 99999. (Criba Eratostenes)
```

```
#include<cstdio>
#include<ctime>
#include<cstring>
#define MAX 99999
bool numero[MAX];
void criba(){
    memset(numero,1,MAX);
    numero[0]=numero[1]=0;
    for(int i=2;i<MAX;i++){
        if(numero[i]){
            printf("%d, ",i);
            for(int j=(i<<1);j<MAX;j+=i){
                numero[j]=0;
            }
        }
    }
}
int main(){
    clock_t t;
    t=clock();
    criba();
    t = clock() - t;
    printf ("\nTarda %d clicks (%f segundos).\n",t,((float)t)/CLOCKS_PER_SEC);
    //MUESTRA: Tarda 2031 clicks (2.031000 segundos).
    return 0;
}
```

8.9 PLANTILLA PARA COMPETENCIAS

```
#include<iostream>
#include<cstdio>
#include<iomanip>
#include<sstream>
#include<cstdlib>
#include<cmath>
#include<string>
#include<cstring>
#include<cctype>
#include<vector>
#include<stack>
#include<queue>
#include<deque>
#include<list>
#include<set>
#include<map>
#include<bitset>
#include<algorithm>
#include<cassert>
#include<climits>
#include<cfloat>
#include<iterator>
#include<utility>
#include<ctime>
#define FOR(i,a,b) for(int i=(int)a;i<(int)b;i++)
#define RFOR(i,a,b) for(int i=(int)a;i>=(int)b;i--)
#define FORC(i,a,b,c) for(int i=(int)a;i<(int)b;i+=c)
#define FORCOND(i,a,b,cond) for(int i=(int)a;i<(int)b && (bool)cond;i++)
#define FORCCOND(i,a,b,c,cond) for(int i=(int)a;i<(int)b && (bool)cond;i+=c)
#define MAX 100
#define PB push_back
#define PF push_front
#define MK make_pair
```

```
#define cerear(v) memset(v,0,sizeof(v))
using namespace std;
typedef unsigned int UI;
typedef long long LL;
typedef unsigned long long ULL;
typedef priority_queue<int>pqueue;
typedef pair<int,int>par;
typedef map<int,int>mapa;
typedef vector< pair<int,int> >vp;
int main(){
    //ios_base::sync_with_stdio(false);
    //freopen("input.txt","r",stdin);
    //freopen("output.txt","w",stdout);
    return 0;
}
```