GUIA BASICA DE CPP

DESCRIPCIÓN

Con esta guía se busca empezar a manejar el lenguaje de cpp o mejorar en el mismo, esto con ejemplos prácticos e imágenes de apoyo.

Edwin Andres Villarraga Ardila

Contenido

Compiladores

Para poder programar en C++ necesitamos de un compilador el cual ayudara a establecer una comunicación entre el lenguaje y la máquina, de esta manera muchos lenguajes de programación se vuelven más atractivos por ser de alto nivel, esto conlleva una ventaja al ser más entendible para las personas.

Codeblocks:

Codeblocks es una aplicación gratuita de escritorio, desarrollada para poder compilar código de C, C++ y Fortran, para el caso de C++ se puede descargar su versión con Cygwin la cual nos permite adaptar GCC en el compilador. Esta aplicación puede descargarse desde su sitio oficial.
http://www.codeblocks.org/downloads

Ilustración 1 Codeblocks Page

Para seguir con el proceso en el apartado de Download the binary release se descarga el ejecutable

Al abrir el ejecutable se desplegará una ventana como la siguiente:

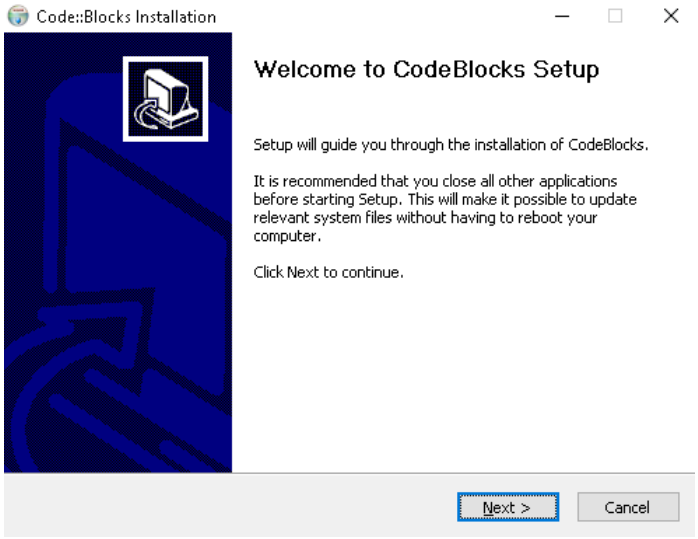

Ilustración 2 Iinstalador de codeblocks

Allí dejaremos todo predeterminado por lo que será presionar el botón de siguiente en todas las ocasiones, al terminar la instalación. Abriremos la aplicación que queda anexa en el escritorio.

Ilustración 3 Logo del compilador

Al ejecutar la aplicación se mostrará una ventana vacía, por lo que para crear un nuevo proyecto vamos al apartado de archivo, nuevo y seleccionar proyecto, en ello se abre una pestaña emergente y buscamos aplicación de consola, y seleccionamos el lenguaje C++.

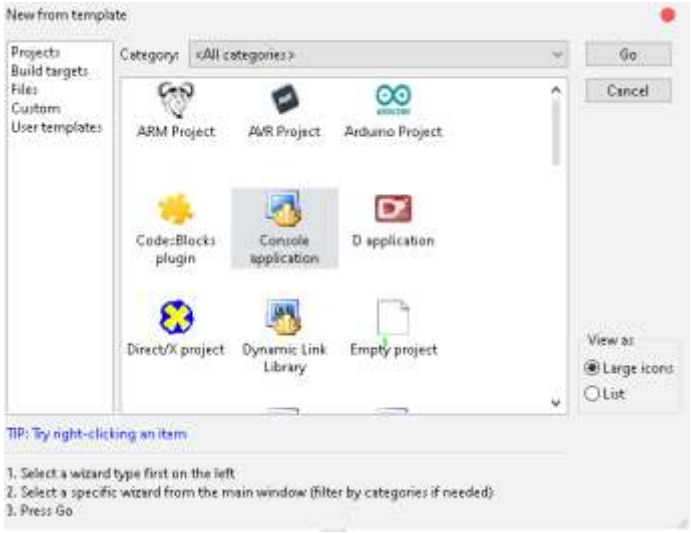

Ilustración 4 Proyecto de consola

Guía rápida de librerías, funciones y código en C/C++

Antes de empezar a declarar variables y crear clases debemos llamar las librerías pertenecientes a **C** y **Cpp** para ello usamos las dos siguientes líneas, estas llamaran todas las funciones estándar del lenguaje.

```
#include <cstdlib>
#include <bits/stdc++.h>
```

Tipos de datos

En C++ al igual que en otros lenguajes existen tipos de datos nativos como lo son, numéricos, lógicos, texto.

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    int numero_entero_corto=123;
    long numero_entero_largo=1234L;
    long long int numero_entero_extra_largo=1234545LL;
    float numero_decimal_corto=3.1415;
    double numero_decimal_largo=3.1415;
    char caracter='C';
    string cadena_de_caracteres="cadena";
    bool booleano=true;
}
```

Esto especifica que los tipos de datos hacen parte de las palabras reservadas del lenguaje, por lo que no podemos nombrar una variable con una de estas palabras., ya que puede causar conflictos al interpretar el código.

-Los tipos de datos booleanos solo pueden tener dos valores posibles **`true`** o **`false.`**

-Los valores de tipo alfanumérico siguen lo siguiente.

Type	Description	Size	Domain
char	Signed character/byte. Characters are enclosed in **single** quotes.	1	-128..127
double	Double precision number	8	ca. $10^{-308}..10^{308}$
int	Signed integer	4	$-2^{31}..2^{31}-1$
float	Floating point number	4	ca. $10^{-38}..10^{38}$
long (int)	Signed long integer	4	$-2^{31}..2^{31}-1$
long long (int)	Signed very long integer	8	$-2^{63}..2^{63}-1$
short (int)	Short integer	2	$-2^{15}..2^{15}-1$
unsigned char	Unsigned character/byte	1	0..255
unsigned (int)	Unsigned integer	4	$0..2^{32}-1$
unsigned long (int)	Unsigned long integer	4	$0..2^{32}-1$
unsigned long long (int)	Unsigned very long integer	8	$0..2^{64}-1$
unsigned short (int)	Unsigned short integer	2	$0..2^{16}-1$

Ilustración 5 7 Introducción a C++ (desy.de)

Salida y Entrada de Datos

cin

Standard input stream
Este objeto proporciona la entrada de datos de forma estándar, por lo que es la manera más sencilla de ingresar valores ya que detecta automáticamente el tipo de dato. La palabra reservada **cin** va acompañada de los signos **>>** indicando donde se van a guardar los valores.
El siguiente ejemplo muestra la entrada de datos para una cadena de caracteres(**string**) y un numero entero(**int**).

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    string cadena;
    cin>>cadena;
    int numero;
    cin>>numero;
}
```

Como podemos apreciar en el ejemplo anterior tenemos que tener las variables declaradas para luego indicarle al objeto **cin** que esa variable tendrá un valor.

cout

Standard output stream
Cout es el método de salida estándar lo que quiere decir que nos ayuda a mostrar en consola. al llamar este objeto indicamos con **<<** queremos que se muestren los valores en consola. Por lo que lo convierte en una manera sencilla de mostrar los datos ya que al igual que **cin**, **cout** detecta automáticamente de que tipo es la variable que queremos imprimir.
Por ejemplo, creamos la variable PI y la mostramos en consola:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    double pi=3.1416;
    cout<<"3.1416 \n";
    cout<<pi<<"\n";
}
```

endl

Insert newline and flush
Endl es una forma de expresar salto de línea o "**\n**", pero este a su vez contiene un flush, este nos sirve para forzar la salida que esta almacenada en el buffer. Por lo que de principio se recomienda utilizar endl y no "**\n**".

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    cout<<"using endl"<<endl;
}
```

using namespace std

namespace std encierra las librerías estándar de esta manera no es necesario indicar en donde residen los objetos.
Por lo que si se olvida esta línea toca indicar de que librería esta saliendo el objeto seguido de ¨: ¨.

```
#include<bits/stdc++.h>
#include<cstdlib>
int main (){
    std::cout<<"sin using"<<endl;
}
```

Para manejar estructuras de datos y entender POO debemos conocer algunas funciones básicas de C++ que se encuentran en sus librerías estándar, las cuales nos ayudaran en casos varios.

Condicionales y Ciclos

If y else

Este tipo de condicional va a cumplirse siempre y cuando la expresión que llegue como parámetro sea verdadera, por lo que el fragmento de código que se encuentra allí solo se ejecuta solo si se cumple esta condición, ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    if(3<4){
        cout<<"3 es menor que 4"<<endl;
    }
    return 0;
}
```

En el ejemplo anterior vemos como la condición de que 3 < 4 es verdadera por lo que llega a ejecutarse el bloque de código de lo contrario existe **else** que se ejecuta en dado caso que la condición del **if** sea falsa.

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    if(3>4){
        cout<<"3 no es menor que 4"<<endl;
    }else{
        cout<<"3 es menor que 4"<<endl;
    }
    return 0;
}
```

Donde comparamos que 3 > 4 condición que es falsa.

while

El ciclo **while** está sujeto a cumplirse siempre y cuando el parámetro que entra sea verdadero. Por lo que el parámetro siempre debe ser verdadero.

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    int numero=3;
    while(numero--){
        cout<<numero<<endl;
    }
    return 0;
}
```

En el ejemplo podemos observar lo siguiente:

- El parámetro número va disminuyendo al terminar una ronda.
- Dentro del ciclo **while** podemos tener entrada o salida de datos.
- El ciclo termina cuando **número** es igual a **0**.

for

El ciclo **for** tiene como parámetros un entero que determina el inicio, seguido de ";" indicamos cual es el final del ciclo y por último decimos el incremento o decremento de la variable de inicio.

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    int fin=5;
    for(int cambio=0;cambio<fin;cambio++){
        cout<<cambio<<" ";
    }
    return 0;
}
```

El ciclo for contiene las siguientes características:

- La variable principal va aumentando o disminuyendo, por lo que es considerada como un contador, además está solo existe dentro del ciclo.
  Contiene en el centro del ciclo un operador indicando hasta cuando termina.

Librerias Basicas

<cassert> (assert.h)

Librería prohibida en competencia
Biblioteca de diagnóstico y aserciones C .
Define una función de macro que se puede usar como una herramienta de depuración estándar:

Funciones:

Assert()

Evalúa una aserción, Por lo que evalúa si la expresión es falsa.

Parámetros:

La función recibe la expresión a evaluar.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
void imprimir(int* miEntero) {
```

```
  assert (miEntero!=NULL);
  printf ("%d\n",*miEntero);
}

int main (){
  int a=10;
  int * b = NULL;
  int * c = NULL;
  b=&a;
  imprimir (b);
  imprimir (c);

  return 0;
}
```

Salida:

```
10

Assertion failed: miEntero!=NULL.
```

<cctype> (ctype.h)

Funciones de manejo de caracteres.

Este header declara un conjunto de funciones para clasificar y transformar caracteres individuales.

Funciones:

Estas funciones toman el equivalente **int** de un carácter como parámetro y devuelven un **int** que puede ser otro carácter o un valor que representa un valor booleano: un valor **int** de 0 significa falso y un valor **int** diferente de 0 representa verdadero.

Isalnum()

Comprueba si el carácter es alfanumérico.

Parámetros:

Carácter a ser verificado, casteado como un entero o **EOF**.

Valor de retorno:

Un valor diferente a cero (**True**) si c es un dígito o una letra, de otro modo cero (**False**).

Ejemplo:

```
#include <cstdlib>
```

```
#include <bits/stdc++.h>
int main (){
    int i;
    char str[]="c3po...";
    i=0;
    while (isalnum(str[i])) i++;
    printf ("The first %d characters are alphanumeric.\n",i);
    return 0;
}
```

Salida:

```
The first 4 characters are alphanumeric.
```

Isalpha()

Comprueba si el carácter es alfabético.

Parámetros:

Carácter a ser verificado, casteado como un entero o **EOF**.

Valor de retorno:

Un valor diferente a cero (**True**) si c es una letra, de otro modo cero (**False**).

Ejemplo:

```
#include <stdio.h>
#include <ctype.h>
int main (){
  int i=0;
  char str[]="C++";
  while (str[i])
  {
    if (isalpha(str[i])) printf ("character %c is alphabetic\n",str[i]);
    else printf ("character %c is not alphabetic\n",str[i]);
    i++;
  }
  return 0;
}
```

Salida:

```
character C is alphabetic
character + is not alphabetic
character + is not alphabetic
```

iscntrl()

Comprueba si el carácter es un carácter de control

Parámetros:

Carácter a ser verificado, casteado como un entero o **EOF**.

Valor de retorno:

Un valor diferente a cero (**True**) si c es un carácter de control, de otro modo cero (**False**).

Ejemplo:

```
#include <stdio.h>
#include <ctype.h>
int main (){
  int i=0;
  char str[]="first line \n second line \n";
  while (!iscntrl(str[i]))
  {
    putchar (str[i]);
    i++;
  }
  return 0;
}
```

Salida:

Este código imprime carácter por carácter hasta que se encuentra un carácter de control que rompe el ciclo **while**. En este caso, solo se imprimirá la primera línea, ya que la línea termina con '**\n**', que es un carácter de control (código ASCII 0x0a).

Isdigit()

Comprueba si el carácter es un dígito decimal

Parámetros:

Carácter a ser verificado, casteado como un entero o **EOF**.

Valor de retorno:

Un valor diferente a cero (**True**) si c es un carácter **digitol**, de otro modo cero (**False**).

Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main (){
  char str[]="1776ad";
  int year;
  if (isdigit(str[0]))
  {
    year = atoi (str);
    printf ("The year that followed %d was %d.\n",year,year+1);
```

```
  }
  return 0;
}
```

Salida:

```
The year that followed 1776 was 1777
```

Islower()

Comprueba si el carácter es una letra en minúsculas

Parámetros:

Carácter a ser verificado, casteado como un entero o EOF.

Valor de retorno:

Un valor diferente a cero (True) si c es un carácter en minúscula, de otro modo cero (False).

Ejemplo:

```
#include <stdio.h>
#include <ctype.h>
int main (){
  int i=0;
  char str[]="Test String.\n";
  char c;
  while (str[i])
  {
    c=str[i];
    if (islower(c)) c=toupper(c);
    putchar (c);
    i++;
  }
  return 0;
}
```

Salida:

```
Test String.
```

Isprint()

Comprueba si el carácter es imprimible

Parámetros:

Carácter a ser verificado, casteado como un entero o EOF.

Valor de retorno:

Un valor diferente a cero (True) si c es un carácter imprimible, de otro modo cero (False).

Ejemplo:

```
#include <stdio.h>
#include <ctype.h>
int main (){
  int i=0;
  char str[]="first line \n second line \n";
  while (isprint(str[i]))
  {
    putchar (str[i]);
    i++;
  }
  return 0;
}
```

Salida:

Este código imprime una cadena carácter por carácter hasta que un carácter que no se puede imprimir se verifica y rompe el ciclo while. En este caso, solo se imprimirá la primera línea, ya que la línea termina con un carácter de nueva línea ('\ n'), que no es un carácter imprimible.

Ispunct()

Comprueba si el carácter es signo de puntuación.

Parámetros:

Carácter a ser verificado, casteado como un entero o EOF.

Valor de retorno:

Un valor diferente a cero (True) si c es un carácter signo de puntuación, de otro modo cero (False).

Ejemplo:

```
#include <stdio.h>
#include <ctype.h>
int main ()
{
  int i=0;
  int cx=0;
  char str[]="Hello, welcome!";
  while (str[i])
  {
    if (ispunct(str[i])) cx++;
    i++;
  }
  printf ("Sentence contains %d punctuation characters.\n", cx);
  return 0;
}
```

Salida:

```
Sentence contains 2 punctuation characters.
```

Isspace()

Comprueba si el carácter es un espacio en blanco, tabular, nueva línea o tabulación vertical.

Parámetros:

Carácter a ser verificado, casteado como un entero o EOF.

Valor de retorno:

Un valor diferente a cero (True) si c es un carácter de espacio, de otro modo cero (False).

Ejemplo:

```c
#include <stdio.h>
#include <ctype.h>
int main ()
{
  char c;
  int i=0;
  char str[]="Example sentence to test isspace\n";
  while (str[i])
  {
    c=str[i];
    if (isspace(c)) c='\n';
    putchar (c);
    i++;
  }
  return 0;
}
```

Este código imprime la cadena C carácter por carácter, reemplazando cualquier carácter de espacio en blanco por un carácter de nueva línea.

Salida:

```
Example
sentence
to
test
isspace
```

isupper()

Comprueba si el carácter es una letra en mayúsculas.

Parámetros:

Carácter a ser verificado, casteado como un entero o EOF.

Valor de retorno:

Un valor diferente a cero (True) si c es un carácter en mayúscula, de otro modo cero (False).

Ejemplo:

```
#include <stdio.h>
#include <ctype.h>
int main ()
{
  int i=0;
  char str[]="Test String.\n";
  char c;
  while (str[i])
  {
    c=str[i];
    if (isupper(c)) c=tolower(c);
    putchar (c);
    i++;
  }
  return 0;
}
```

Salida:

```
test string.
```

Isxdigit()

Comprueba si el carácter es un dígito hexadecimal.

Parámetros:

Carácter a ser verificado, casteado como un entero o EOF.

Valor de retorno:

Un valor diferente a cero (True) si c es un dígito hexadecimal, de otro modo cero (False).

Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main ()
{
  char str[]="ffff";
  long int number;
  if (isxdigit(str[0]))
  {
    number = strtol (str,NULL,16);
    printf ("The hexadecimal number %lx is %ld.\n",number,number);
  }
```

```
  return 0;
}
```

isxdigit se usa para verificar si el primer carácter en str es un dígito hexadecimal válido y, por lo tanto, un candidato válido para ser convertido por strtol en un valor integral.

Salida:

```
The hexadecimal number ffff is 65535.
```

Tolower()

Convierte el carácter a minúsculas

Parámetros:

Carácter a ser modificado, casteado como un entero o EOF.

Valor de retorno:

El equivalente en minúsculas de c, si existe dicho valor, o c (sin cambios) de lo contrario.
El valor se devuelve como un valor int que se puede convertir implícitamente en char.

Ejemplo:

```
#include <stdio.h>
#include <ctype.h>
int main ()
{
  int i=0;
  char str[]="Test String.\n";
  char c;
  while (str[i])
  {
    c=str[i];
    putchar (tolower(c));
    i++;
  }
  return 0;
}
```

Salida:

```
test string.
```

Toupper()

Convierte el carácter a mayúsculas.

Parámetros:

Carácter a ser modificado, casteado como un entero o EOF.

Valor de retorno:

El equivalente en mayúsculas de c, si existe dicho valor, o c (sin cambios) de lo contrario.
El valor se devuelve como un valor int que se puede convertir implícitamente en char.

Ejemplo:

```
#include <stdio.h>
#include <ctype.h>
int main ()
{
  int i=0;
  char str[]="Test String.\n";
  char c;
  while (str[i])
  {
    c=str[i];
    putchar (toupper(c));
    i++;
  }
  return 0;
}
```

Salida:

```
TEST STRING.
```

<ciso646> (iso646.h)

Librería prohibida en competencia

Operadores de deletreo alternativos ISO 646

macro	operator
and	&&
and_eq	&=
bitand	&
bitor	|
compl	~
not	!
not_eq	!=
or	||
or_eq	|=

xor	^
xor_eq	^=

Tabla 1 Operadores binarios

<cmath> (math.h)

Librería C de manejo numérico.

Funciones:

pow()

La función pow() es la encargada de realizar una potencia, para ello requiere saber la base y el exponente.

Parámetros:

Necesita dos números teniendo en cuenta que el primer número es la base y el segundo el exponente.

Valor de retorno:

Un entero correspondiente a la potencia.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    int a=2,b=3;
    cout<<pow(a,b)<<"\n";
    return 0;
}
```

Salida:

```
8
```

sqrt()

La función sqrt nos permite hallar la raíz cuadrada de un número.

Parámetros:

Debe recibir un número positivo, al cual se le hallara la raíz.

Valor de retorno:

Un número positivo correspondiente a √x.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    int a=25;
    double raiz=sqrt(25);
    cout<<raiz<<"\n";
    return 0;
}
```

Salida:

```
5
```

modf()

Nos permite separar un numero de tipo float o double en dos partes indicando lo siguiente. el primero será la parte entera y la segunda será la parte decimal.

Parámetros:

La función recibe un numero de tipo decimal seguido de llamar un variable ya declarada.

Valor de retorno:

Retorna el valor de los decimales, además de devolver en otra variable cual es la parte entera del número.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    double numero=3.1415;
    double parte_entera;
    double fractpart = modf(numero,&parte_entera);;
    printf("%f = %f + %f",numero,parte_entera,fractpart);
    return 0;
}
```

Salida:

```
3.141500 = 3.000000 + 0.141500
```

log()

Permite sacar el logaritmo de un número.

Parámetros:

El número debe ser positivo de lo contrario retornará un error.

Valor de retorno:

Retornara un número con el logaritmo del valor dado.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    double numero=3.1415;
    double total = log(numero);
    printf("log(%f) = %f \n",numero,total);
    return 0;
}
```

Salida:

```
log(3.141500) = 1.144700
```

log2()

De igual manera que la función anterior log2() nos permite hallar el logaritmo de un número, pero esta vez se realizará en base 2.

Parámetros:

El número debe ser positivo de lo contrario retornará un error.

Valor de retorno:

Devuelve un número con el logaritmo en base 2 del valor dado.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    double numero=1024;
    double total = log2(numero);
    printf("log(%f) = %f \n",numero,total);
    return 0;
}
```

Salida:

```
log(1024.000000) = 10.000000
```

abs()

Esta función nos permite sacar el valor absoluto de un número.

Parámetros:

El valor debe ser un número retornable.

Valor de retorno:

Retorna el valor absoluto del número |x|.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;

int main() {
    printf("Valor absoluto: %d\n",abs(-322));
    printf("Valor absoluto: %f\n",abs(-3.1416));
    return 0;
}
```

Salida:

```
Valor absoluto: 322
Valor absoluto: 3.141600
```

floor()

Redondea un decimal hacia abajo sin importar si los decimales están cercanos al siguiente entero.

Parámetros:

La función recibe cual es el número.

Valor de retorno:

Devuelve un número el cual se redondeó hacia abajo.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;

int main() {
    float pi=3.1415;
    float num=3.8;
    printf("redondeo hacia abajo %f\n",floor(pi));
    printf("redondeo hacia abajo %f\n",floor(num));
    return 0;
}
```

Salida:

redondeo hacia abajo 3.000000
redondeo hacia abajo 3.000000

ceil()

Redondea el número hacia el siguiente valor entero sin importar si sus decimales son bajos.

Parámetros:

Debe recibir un número.

Valor de retorno:

Devuelve el valor del número con un redondeo hacia arriba.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;

int main() {
    float pi=3.1415;
    float num=3.8;
    printf("redondeo hacia arriba %f\n",ceil(pi));
    printf("redondeo hacia arriba %f\n",ceil(num));
    return 0;
}
```

Salida:

```
redondeo hacia abajo 4.000000
redondeo hacia abajo 4.000000
```

lround()

Nos permite redondear el número a su entero más cercano dependiendo de qué tan grande sean sus decimales.

Parámetros

Debe tener el número que se desea redondear.

Valor de retorno:

Devuelve un número redondeado casteando a un **long int**.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
typedef double lf;
```

```
using namespace std;
int main() {
    lf arriba=3.68,abajo=3.25;
    printf("el redondea hacia arriba de %f es
%lld\n",arriba,lround(arriba));
    printf("el redondea hacia abajo de %f es
%lld\n",abajo,lround(abajo));
    return 0;
}
```

Salida:

```
el redondea hacia arriba de 3.680000 es 4
el redondea hacia abajo de 3.250000 es 3
```

sin() - sinh() - asin()

sin: calcula la función trigonométrica del seno de un ángulo x, sinh: devuelve la función hiperbólica del seno, asin: retorna el arc-seno de x.

Parámetros

Se ingresa el valor del ángulo en radianes.

Valor de retorno:

Retorna el valor de x en radianes.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    double decimal=0.50;
    double r1=sin(decimal);
    double r2=sinh(decimal);
    double r3=asin(decimal);
    cout<<r1<<" "<<r2<<" "<<r3<<endl;
    return 0;
}
```

Salida:

```
0.479426 0.521095 0.523599
```

cos() - cosh() - acos()

cos: calcula la función trigonométrica del coseno de un ángulo x, cosh: devuelve la función hiperbólica del coseno, acos: retorna el arc-coseno de x.

Parámetros

Se ingresa el valor del ángulo en radianes.

Valor de retorno:

Retorna el valor de x en radianes.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    double decimal=0.65;
    double r1=cos(decimal);
    double r2=cosh(decimal);
    double r3=acos(decimal);
    cout<<r1<<" "<<r2<<" "<<r3<<endl;
    return 0;
}
```

Salida:

```
0.796084 1.21879 0.863212
```

tan() - tanh() – atan()

tan: calcula la función trigonométrica del tangente de un ángulo x, tanh: devuelve la función hiperbólica de la tangente, atan: retorna el arc-tangente de x.

Parámetros

Se ingresa el valor del ángulo en radianes.

Valor de retorno:

Retorna el valor de x en radianes.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    double decimal=0.45;
    double r1=tan(decimal);
    double r2=tanh(decimal);
    double r3=atan(decimal);
    cout<<r1<<" "<<r2<<" "<<r3<<endl;
    return 0;
}
```

Salida:

```
0.900447 1.10297 1.10403
```

hypot()

Calcula el valor de la suma y luego aplica la raíz (teorema de Pitágoras).

Parámetros

Se ingresan dos valores de a y b para luego computar la hipotenusa.

Valor de retorno:

Devuelve la raíz de (x^2+y^2).

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    double x=3,y=4;
    double h=hypot(x,y);
    cout<<h<<endl;
    return 0;
}
```

Salida:

```
5
```

cbrt()

Calcula el valor de la raíz cubica de x.

Parámetros

Ingresa el valor del número para calcular la raíz.

Valor de retorno:

Retorna la raíz cubica = ($\sqrt[3]{x}$).

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    double numero=125;
    double total=cbrt(numero);
    cout<<total<<endl;
    return 0;
}
```

Salida:

5

<cstdint> (stdint.h)

Tipos enteros.

Esta librería nos da el conjunto de tipos de datos con sus macros específicos que especifican los límites de los tipos.

Macros

INTMAX_MIN	**Minimum value of intmax_t**	**$-(2^{63}-1)$, or lower**
INTMAX_MAX	Maximum value of intmax_t	$2^{63}-1$, or higher
UINTMAX_MAX	Maximum value of uintmax_t	$2^{64}-1$, or higher
INT*N*_MIN	Minimum value of exact-width signed type	Exactly $-2^{(N-1)}$
INT*N*_MAX	Maximum value of exact-width signed type	Exactly $2^{(N-1)}-1$
UINT*N*_MAX	Maximum value of exact-width unsigned type	Exactly $2^{N}-1$
INT_LEAST*N*_MIN	Minimum value of minimum-width signed type	$-(2^{(N-1)}-1)$, or lower
INT_LEAST*N*_MAX	Maximum value of minimum-width signed type	$2^{(N-1)}-1$, or higher
UINT_LEAST*N*_MAX	Maximum value of minimum-width unsigned type	$2^{N}-1$, or higher
INT_FAST*N*_MIN	Minimum value of fastest minimum-width signed type	$-(2^{(N-1)}-1)$, or lower
INT_FAST*N*_MAX	Maximum value of fastest minimum-width signed type	$2^{(N-1)}-1$, or higher
UINT_FAST*N*_MAX	Maximum value of fastest minimum-width unsigned type	$2^{N}-1$, or higher
INTPTR_MIN	Minimum value of intptr_t	$-(2^{15}-1)$, or lower
INTPTR_MAX	Maximum value of intptr_t	$2^{15}-1$, or higher
UINTPTR_MAX	Maximum value of uintptr_t	$2^{16}-1$, or higher

Tabla 2 Limites de los Tipos de datos

<cstdio> (stdio.h)

Librería de manejo de operaciones de entrada y salida de C.

Funciones:

printf()

Nos permite dar un formato de salida, impresión rápida.

Parámetros:

Se debe especificar qué tipo de objeto es con una nomenclatura especial.

Especificación	Salida
%i o %d	Número entero.
%u	Número entero.
%o	Número octal.
%x	Entero Hexadecimal.
%X	Entero Hexadecimal en mayúsculas.
%a	Decimal Hexadecimal en minúsculas.
%A	Decimal Hexadecimal en mayúsculas.
%f	Números decimales.
%s	String de caracteres.
%e	Notación científica en minúsculas.
%E	Notación científica en mayúsculas.
%c	Carácter.

Tabla 3 Formatos de impresión para los tipos de datos

Valor de retorno:

Este devuelve el tipo de dato especificado para luego escribirlo.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    printf ("caracteres: %c %c \n", 'a', 65);
    printf ("decimales: %d %ld\n", 1977, 650000L);
    printf ("decimales por notacion: %4.2f %E \n", 3.1416, 3.1416);
    printf ("%s \n", "AAA string");
    return 0;
}
```

Salida:

```
caracteres: a A
decimales: 1977 650000
decimales por notación: 3.14 3.141600E+000
AAA string
```

gets()

Este permite leer caracteres de manera estándar, de esta manera lee hasta que no encuentre más caracteres.

Parámetros:

Solo es posible leer para un vector de caracteres.

Valor de retorno:

Retorna un string hasta que encuentre el fin de archivo.

Ejemplo:

```
#include <cstdlib>
#include<bits/stdc++.h>
//cantidad maxima de caracteres
#define MAX 256
using namespace std;
int main() {
    char str[MAX];
    gets(str);
    cout<<str<<"\n";
    return 0;
}
```

Salida:

```
Entrada por teclado usando gets
```

scanf()

scanf nos permite leer los diferentes tipos de datos, es una lectura más rápida que el std.

Parámetros:

De igual manera que el printf debemos especificar el tipo de dato que estamos leyendo además de colocar el signo '&' antes del nombre de la variable

Especificación	Salida
%i o **%d**	Número entero.

%u	Número entero.
%o	Número octal.
%x	Entero Hexadecimal.
%X	Entero Hexadecimal en mayúsculas.
%a	Decimal Hexadecimal en minusculas.
%A	Decimal Hexadecimal en mayúsculas.
%f	Números decimales.
%s	String de caracteres.
%e	Notación científica en minúsculas.
%E	Notación científica en mayúsculas.
%c	Carácter.

Tabla 4 Formato de lectura de datos

Valor de retorno:

Devuelve el valor a la variable asignada.

Ejemplo:

```
#include <cstdlib>
#include<bits/stdc++.h>

using namespace std;
int main() {
    int entero;
    float decimal;
    string s;
    scanf("%d",&entero);
    scanf("%f",&decimal);
    scanf("%s",&s);
    cout<<"Este es un entero "<<entero<<"\n";
    cout<<"Este es un decimal "<<decimal<<"\n";
    cout<<"Este es un string "<<s<<"\n";
    return 0;
}
```

Salida:

```
11
1.21
this a string
Este es un entero 11
Este es un decimal 1.21
Este es un string
```

sscanf()

sscanf lee los datos almacenados en un string para poder resignar los valores en diferentes variables, por lo que se debe especificar el objeto y formato del valor final.

Parámetros:

Se ingresa un tipo de dato string el cual contiene los datos a procesar, el formato final de los tipos de datos, y de ultimo donde van a quedar alojados los valores.

Valor de retorno:

Retorna los nuevos valores a los objetos asignados de no poder asignar el valor lo interpreta como nulo.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    string complete,nombre;
    int edad;
    printf("Cual es su nombre y edad : ");
    getline(cin,complete);
    sscanf(complete.c_str(),"%s %d",nombre,&edad);
    printf("%s \n%d \n",nombre,edad);
}
```

Salida:

```
Cual es su nombre y edad : Edwin 18

Edwin

18
```

getchar()

Esta función nos permite leer un carácter sin tener que especificar.

Parámetros:

Ninguno.

Valor de retorno:

Retorna el que se ingresó por consola.

Ejemplo:

```
#include <cstdlib>
#include<bits/stdc++.h>
using namespace std;
int main() {
    char c;
    c=getchar();
    printf("%c",c);
    return 0;
}
```

Salida:

```
@
```

putchar ()

Esta es capaz de imprimir solo el carácter al igual que getchar() no tenemos que especificarlo.

Parámetros:

El valor que se asigna internamente debe existir.

Valor de retorno:

Retorna el carácter mientras sea diferente de EOF

Ejemplo:

```
#include <cstdlib>
#include<bits/stdc++.h>

using namespace std;
int main() {
    char c;
    for(c='A';c<='Z';c++){
        putchar(c);
    }
}
```

Salida:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

<cstdlib> (stdlib.h)

Liberia general de utilidades estándar de C

Funciones:

atoi()

La función atoi nos permite pasar una cadena de caracteres numéricos a entero.

Parámetros:

El string debe contener números enteros en base 10.

Valor de retorno:

Si el entero está en el rango de un MAX INTEGER este se devolverá un número, si por un motivo sobrepasa el máximo se puede intentar con la función atol().

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
//cantidad maxima de caracteres
#define MAX 256
using namespace std;
int main() {
    char num[MAX]="123";
    string num1="112";
    int toint=atoi(num);
    int toints=atoi(num1.c_str());
    cout<<toint<<" "<<toints<<"\n";
    return 0;
}
```

Salida:

123 112

atol()

Al igual que la función atoi esta busca convertir un string a un valor numérico esta función es usada por si el número es mayor al **MAX_INT**.

Parámetros:

El string debe contener números enteros en base 10.

Valor de retorno:

Devuelve el string convertido a un entero.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
//cantidad maxima de caracteres
#define MAX 256
using namespace std;
int main() {
    char num[MAX]="12456463";
    string num1="1232323";
```

```
    long toll=atol(num);
    long toll1=atol(num1.c_str());
    cout<<toll<<" "<<toll1<<"\n";
    return 0;
}
```

Salida:

```
12456463 1232323
```

atof()

Igual que las funciones anteriores este busca que un string sea un número para este caso atof() acepta decimales.

Parámetros:

El string debe contener números en base 10 y puede contener decimales.

Valor de retorno:

Devuelve el string convertido a un **double**.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
//cantidad maxima de caracteres
#define MAX 256
using namespace std;
int main() {
    char num[MAX]="12.456463";
    string num1="3.1416";
    double todou=atof(num);
    double todou1=atof(num1.c_str());
    cout<<todou<<" "<<todou1<<"\n";
    return 0;
}
```

Salida:

```
12.456463 3.1416
```

labs()

Saca el valor absoluto de un número de tipo **long int**.

Parámetros:

Entra el número al que se le aplicara el valor absoluto.

Valor de retorno:

El valor absoluto del número que se ingresó.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main() {
    long num=-1000L;
    cout<<labs(num)<<endl;
}
```

Salida:

```
1000
```

div()

para esta función se necesita un tipo de dato **div_t** este guardara entero de la división y el residuo

Parámetros:

c --- La función recibe el numerador y denominador.

Valor de retorno:

El resultado de la división se retorna en un objeto **div_t**, este tipo de dato contiene dos valores el cociente y el residuo.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    div_t total;
    int a=201;
    int b=5;
    total=div(a,b);
    cout<<"Cociente: "<<total.quot<<"\nResiduo: "<<total.rem<<endl;
}
```

Salida:

```
Cociente: 40
Residuo: 1
```

rand()

Rand nos permite generar números aleatorios desde 0 hasta un numero que es el máximo.
Rand() % 10 – el número máximo puede ser 10.
Rand() % 10 + 1 – Los numero aleatorio se generan desde 1 hasta 10.

Parámetros:

c --- Ninguno.

Valor de retorno:

Devuelve un numero entre 0 y un máximo.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main (){
    int a=rand() %10;
    int b=rand() %1 + 10;
    while(a!=b){
        cout<<a<<" "<<b<<endl;
        a=rand() % 10;
        b=rand() % 10 + 1;
    }
    cout<<"a == b : "<<a<<endl;
}
```

Salida:

```
1 10
4 1
9 5
8 9
2 5
5 6
1 8
1 2
5 3
a == b : 7
```

bsearch()

realiza una búsqueda binaria en el arreglo, devuelve un puntero si se encuentra el elemento, para esta búsqueda el arreglo debe estar arreglado de menor a mayor.

Parámetros:

Se debe ingresar el elemento que se va a buscar, seguido de ello va el arreglo en donde se va a buscar, la cantidad de datos del arreglo, el tamaño en bites del arreglo, y la función que va a comparar dos datos.

Valor de retorno:

Si el elemento está presente devolverá un valor de lo contrario si no se encuentra devuelve un puntero vacío.

Ejemplo:

```
#include<bits/stdc++.h>
```

```
#include<cstdlib>
using namespace std;
int compareints (const void * a, const void * b){
  return ( *(int*)a - *(int*)b );
}
int main (){
    int arr[]={0,1,5,10,100,200,500};
    int n=sizeof(arr)/sizeof(arr[0]);
    int key=200;
    auto *it = bsearch(&key,arr,n,sizeof(int),compareints);
    if(it!=NULL){
        cout<<key<<" esta presente en el arreglo"<<endl;
    }else{
        cout<<key<<" NO esta presente en el arreglo"<<endl;
    }
}
```

Salida:

```
200 esta presente en el arreglo
```

<cstring> (string.h)

Manejo de strings en C

La librería de string.h contiene funciones para manipular estas cadenas incluyendo algunas para comparar, cambiar caracteres, copiar y demás.

Funciones:

strcmp()

strcmp() compara que dos vectores de string sean iguales, retorna un entero 0 si las cadenas son iguales ,de lo contrario devuelve un entero 1 diciendo que son diferentes.

Parámetros:

c --- El string debe estar declarado como un vector de caracteres.

Valor de retorno:

retorna un entero 1 y 0 indicando 1 que son diferentes y 0 que son iguales.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
//cantidad maxima de caracteres
#define MAX 256
using namespace std;
int main() {
    char a[MAX]="AAA";
```

```
    char b[MAX]="AAA";
    if(strcmp(a,b)==0){
        cout<<"la cadena a y b Son iguales"<<"\n";
    }
    return 0;
}
```

Salida:

```
la cadena a y b Son iguales
```

strlen()

la función strlen devuelve el tamaño del string , de esta manera podemos guardar el número en un entero por si se desea utilizar más adelante.

Parámetros:

c --- El string debe estar declarado como un vector de caracteres.

Valor de retorno:

la función devuelve un entero que indica el largo de la cadena.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
//cantidad maxima de caracteres
#define MAX 256
using namespace std;
int main() {
    char s[MAX]="example";
    int len=strlen(s);
    cout<<len<<"\n";
    return 0;
}
```

Salida:

```
7
```

memset()

Le asigna un valor a un bloque de memoria, usado para inicializar un **vector**

Parámetros:

indicamos cual es el objeto que vamos a cambiar seguido de ello le asignamos el valor por último le diremos en qué posición.

Valor de retorno:

retorna los nuevos valores del objeto que se cambió.

Ejemplo:

```
#include <cstdlib>
#include<bits/stdc++.h>
using namespace std;
int main() {
    int vec[15];
    memset(vec, -1, sizeof(vec));
    //for each para recorrer
    //la estructura(arreglo)
    for(int a:vec){
        cout<<a<<" ";
    }
    return 0;
}
```

Salida:

```
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

strcpy()

traslada la información que se encuentra en un string a un segundo string.

Parámetros:

se debe escoger cual es el destinatario y luego de ello cuál es el string que se desea copiar.

Valor de retorno:

esta función retorna el string destinatario una vez ya le asignó el nuevo valor.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
#define MAX 256
using namespace std;
int main() {
    char str1[MAX]="this a sample";
    char str2[MAX];
    strcpy(str2,str1);
    //strcpy(str2,"example");
    //en caso de no tener el valor dentro de un string
    cout<<str2<<"\n";
    return 0;
}
```

Salida:

```
this a sample
```

strcat()

strcat nos permite concatenar dos string, incluyendo el valor al primer string.

Parámetros:

debemos tener el string destinatario y el segundo string que se concatena con el primero, se debe tener en cuenta que string debe ser un vector de caracteres.

Valor de retorno:

Un string en el cual tendrá su valor más el del nuevo valor.

Ejemplo:

```
#include <cstdlib>
#include<bits/stdc++.h>
//cantidad maxima de caracteres
#define MAX 256
using namespace std;
int main() {
    char s[MAX]="this";
    strcat(s," is a ");
    strcat(s," example");
    //s ahora tiene el valor de this is a example
    printf("%s",s);
    return 0;
}
```

Salida:

```
this is a example
```

memcmp()

compara dos bloques de memoria o string

Parámetros:

Se debe decir cuál es el primer bloque, el segundo bloque, y el número de caracteres a comparar

Valor de retorno:

esta función retorna un entero x>0, x<0, o x==0 .si el número es mayor a cero es que tiene letras en mayúsculas o son parecidos, si es menor que cero indica que tiene letras minúsculas o son parecidos, y por último que son iguales

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    char str1[]="0x000007b";
    char str2[]="0X000007B";
    int n=memcmp(str1,str2,sizeof(str1));
    if(n==0){
        cout<<str1<<" son iguales "<<str2<<"\n";
    }else if(n>0){
```

```
        cout<<str1<<" letras mayusculas "<<str2<<"\n";
    }else{
        cout<<str1<<" letras minisculas "<<str2<<"\n";
    }
    return 0;
}
```

Salida:

```
0x000007b letras mayusculas 0X000007B
```

<ctime> (time.h)

Liberia de tiempo de C

ctime()

Interpreta el tiempo de la máquina y la convierte a el tiempo de calendario siguiendo el siguiente formato.

```
Www Mmm dd hh:mm:ss yyyy
```

Parámetros:

Un puntero de objeto que contiene el tiempo aritmético

Valor de retorno:

Retorna un C-string con que contiene la información legible.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main() {
    time_t rawtime;
    time (&rawtime);
    printf ("The current local time is: %s", ctime (&rawtime));
}
```

Salida:

```
The current local time is: Tue Jul 28 16:33:41 2020
```

Librerías estándar y headers misceláneos de C++ 11

<algorithm>

Librería estándar de algoritmos pre-compilados.

replace()

Esta función nos permitirá cambiar el valor de un carácter que se encuentre dentro del vector.

Parámetros:

Se debe tener un objeto vector, además en la función replace le indicamos cual es el inicio y el final del vector seguido de ello elegimos el valor que se quiere cambiar y por último el nuevo valor que tendrá este.

Valor de retorno:

Devuelve el nuevo valor del carácter seleccionado.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    int repla[]={1,2,3,4,5,6,7,8,9};
    int n=sizeof(repla)/sizeof(repla[0]);
    vector<int>vec(repla,repla+n);
    replace(vec.begin(),vec.end(),5,0);
    for(int i=0;i<vec.size();++i){
        printf("%d ",vec[i]);
    }
}
```

Salida:

```
1 2 3 4 0 6 7 8 9
```

reverse()

Esta función nos permite voltear el vector.

Parámetros:

Debe ser un vector bidimensional, se debe indicar cual es inicio del vector y el final, además debe ser un objeto vector.

Valor de retorno:

Retorna los diferentes valores que se encuentran en el vector después de aplicarle el reverse.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(int argc, char** argv) {
    vector<int>vec;
    //0 1 2 3 4 5 6 7 8 9
    for(int i=0;i<10;++i){
        vec.push_back(i);
    }
    //9 8 7 6 5 4 3 2 1 0
    reverse(vec.begin(),vec.end());
    for(int i=0;i<vec.size();++i){
        printf("%d ",vec[i]);
    }
}
```

Salida:

```
9 8 7 6 5 4 3 2 1 0
```

min()

compara dos valores y devuelve el valor menor.

Parámetros:

los valores que se van a comparar deben ser del mismo tipo.

Valor de retorno:

retorna el valor mínimo de los dos valores.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;

int main() {
    float pi=3.1415;
    float num=3.8;
    printf("el numero menor es %f\n",min(pi,num));
    return 0;
}
```

Salida:

```
el numero menor es 3.141500
```

count()

verifica cuántas veces está repetido el dato.

Parámetros:

debemos determinar cuál es el inicio y el final del arreglo y por último de vamos a indicar cual es el dato a buscar.

Valor de retorno:

devuelve un número con la cantidad de veces que se encontró el valor.

Ejemplo 1:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int arr[]={2,4,5,2,1,1,3,5,6,3,2};
    int len=sizeof arr / sizeof arr[0];
    int c;
    c=count(arr,arr+len,1);
    cout<<"el numero 1 se repite "<<c<<endl;
    c=count(arr,arr+len,8);
    cout<<"el numero 8 se repite "<<c<<endl;
    return 0;
}
```

Salida:

```
el numero 1 se repite 2
el numero 8 se repite 0
```

Ejemplo 2:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    char arr[]={'e','d','c','c','e','e'};
    int len=sizeof arr / sizeof arr[0];
    int c;
    c=count(arr,arr+len,'c');
    cout<<"la letra c se repite "<<c<<endl;
    c=count(arr,arr+len,'e');
    cout<<"la letra e se repite "<<c<<endl;
    return 0;
}
```

Salida:

```
la letra c se repite 2
la letra e se repite 3
```

<iterator>

Librería definidora de iteradores

Operaciones numéricas generalizadas

<regex>

Manejo de expresiones regulares

<utility>

Componentes de utilidad

Pair

pair es una estructura que puede agrupar dos valores de retorno, estos pueden ser de diferente tipo.

make_pair()

nos permite agregar un nuevo par de datos.

Parámetros:

indicamos cuales son los dos valores que vamos a agregar al **pair**.

Valor de retorno:

el **pair** obtiene los valores de **X** y **Y** respectivamente.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main() {
    pair <int,string>pareja;
    pareja= std::make_pair(18,"edwin");
    cout<<pareja.first<<" "<<pareja.second;
}
```

Salida:

```
18 edwin
```

<valarray>

Librería para **arrays** con valores numéricos

Valarray permite realizar operaciones matemáticas a todo un vector de manera fácil.

sum()

La función sum que se encuentra en valarray nos permite sumar todos los datos del arreglo.

Parámetros:

Ninguna.

Valor de retorno:

Devuelve el valor de la suma.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    valarray<int>varr{3,4,23,4,4,5,12,13};
    cout<<varr.sum()<<endl;
    return 0;
}
```

Salida:

```
68
```

max()

max me retorna el número más grande que encuentre en el arreglo.

Parámetros:

Ninguna.

Valor de retorno:

Devuelve el valor más grande.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    valarray<int>varr{3,4,23,4,4,5,12,13};
    cout<<varr.max()<<endl;
    return 0;
}
```

Salida:

```
23
```

min()

min me retorna el número más pequeño que encuentre en el arreglo.

Parámetros:

Ninguna.

Valor de retorno:

Devuelve el valor más pequeño.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    valarray<int>varr{3,4,23,4,4,5,12,13};
    cout<<varr.min()<<endl;
    return 0;
}
```

Salida:

```
3
```

apply()

Nos permite aplicar una función a cada elemento del arreglo.

Parámetros:

Se debe ingresar la función que se va a aplicar a cada uno de los valores.

Valor de retorno:

Devuelve el nuevo arreglo con la operación realizada a los elementos.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int fiveAdd(int x){
    return (x+5);
}
int main() {
    int arr[]={1,2,3,4,5};
    int n= sizeof(arr)/sizeof(arr[0]);
    valarray<int>varr (arr,n);
    varr=varr.apply(fiveAdd);
    for(int i:varr){
        cout<<i<<" ";
    }
```

```
    return 0;
}
```

Salida:

```
6 7 8 9 10
```

size()

Calcula la cantidad de datos que se encuentran dentro del arreglo.

Parámetros:

Ninguna.

Valor de retorno:

Ninguna.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    valarray<int>varr{2,343,4,3,2,3,4};
    cout<<varr.size()<<endl;
    return 0;
}
```

Salida:

```
7
```

Crear Librerias

Para poder definir nuestra propia librería, creamos un archivo en blanco y lo guardamos con la extensión “miLibreria**.h**” esto determina que es una librería. Este archivo no contiene la función **main** ya que no queremos que ejecute todo el archivo si no que solo ejecute cierto bloque de código que llamemos desde el “.**cpp**”.

Para poder llamar nuestra librería usamos #include “miLibreria.h”,al ser un archivo que se encuentra remotamente lo llamamos con la dirección de la carpeta, si esta se encuentra en el mismo lugar que el **main** solo escribimos el nombre entre comillas dobles.

Ejemplo de libreria:

```
#include <bits/stdc++.h>
#include <cstdlib>
//miLibreria.h
using namespace std;
void imprimir(){
    cout<<"Esta es una funcion de mi libreria"<<endl;
}
```

Ejemplo del archivo main

```
#include <bits/stdc++.h>
#include <cstdlib>
#include "miLibreria.h"
using namespace std;
int main() {
    imprimir();
    return 0;
}
```

De esta manera podemos crear nuestras propias librerías para no saturar el archivo principal, también podemos indicar los parámetros si la función lo requiere por devolver la suma de dos números

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
void imprimir(){
    cout<<"Esta es una funcion de mi libreria"<<endl;
}
int suma(int a,int b){
    return (a+b);
}
```

Estructuras de Datos

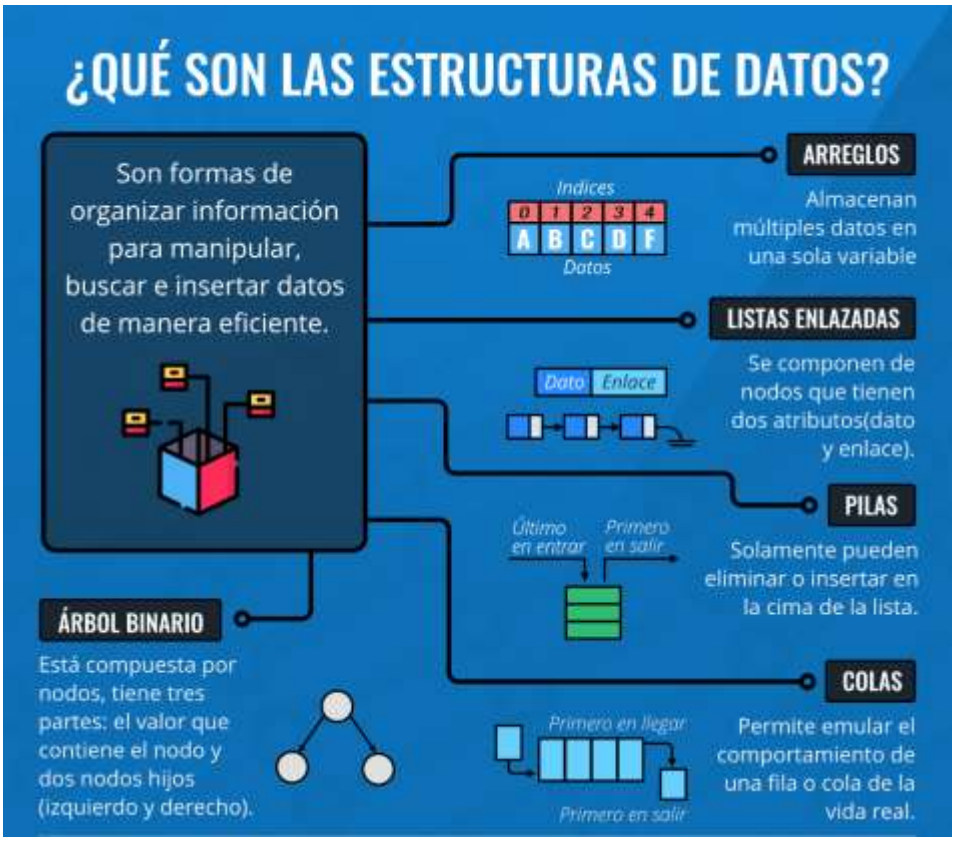

Ilustración 6 (174) Pinterest

Array

Array es una forma de reescribir un vector fijo, con la diferencia que este es un objeto ya nativo del lenguaje (a partir de C++ 11). este objeto conlleva la siguiente estructura:
array<OBJECT, SIZE>name;

Podemos apreciar que debemos ingresar dos parámetros dentro de los diamantes, el primero corresponde al tipo de dato del vector, y el segundo el tamaño que determina que el arreglo es fijo.

size()

Parámetros:

Ninguno.

Valor de retorno:

Devuelve el tamaño del arreglo.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    array<int,10>arr;
    cout<<arr.size()<<endl;
}
```

Salida:

```
10
```

front()

Permite acceder al dato que se encuentra de primeras.

Parámetros:

Ninguno.

Valor de retorno:

Devuelve el primer dato.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    array<int,10>arr;
    ///{2,3,4,5,6,7,8,9,10,11}
    for(int i=0;i<arr.size();i++){
        arr[i]=i+2;
    }
    cout<<arr.front()<<endl;
}
```

Salida:

```
2
```

back()

Permite acceder al dato que se encuentra de ultimas en el arreglo.

Parámetros:

Ninguno.

Valor de retorno:

Devuelve último dato(n-1) correspondiente el último dato.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main() {
    array<int,10>arr;
    ///{2,3,4,5,6,7,8,9,10,11}
    for(int i=0;i<arr.size();i++){
        arr[i]=i+2;
    }
    cout<<arr.back()<<endl;
}
```

Salida:

```
11
```

fill()

Permite llenar el vector con un solo dato repetido n veces.

Parámetros:

El elemento el cual será asignado a todas las posiciones del vector.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    array<int,5>arr;
    arr.fill(8);
    for(auto i:arr){
        cout<<i<<" ";
    }
    return 0;
}
```

Salida:

```
8 8 8 8 8
```

vector

Vector es un arreglo dinámico.
vector<OBJECT>name;

insert()

Permite agregar nuevos elementos después de una posición específica.

Parámetros:

Para agregar un dato al objeto debemos tener un iterador y la función insert no pedirá el iterador y después el dato que queremos agregar.
como segundo constructor podemos agregar un arreglo al vector indicándole en la función donde comienza en vector, cual es el vector y por último el vector sumado con el largo del mismo.

Valor de retorno:

El iterador nos ayudará a retornar el valor que estamos indicando.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    int arr[]={2,3,4,2,5};
    int len=sizeof(arr)/sizeof(arr[0]);
    vector<int> myvec2;
    myvec2.insert(myvec2.begin(),arr,arr+len);
    std::vector<int>::iterator it2=myvec2.begin();
    myvec2.insert(it2,10);
    for(auto it:myvec2){
        cout<<it<<" ";
    }
  return 0;
}
```

Salida:

```
10 2 3 4 2 5
```

front()

esta función nos permite seleccionar el dato se agregó de primeras en el vector.

Parámetros:

ninguno.

Valor de retorno:

retorna el valor que se encuentra en la primera posición.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
```

```
int main(){
    int arr[]={2,3,4,5,10};
    int len=sizeof(arr)/sizeof(arr[0]);
    vector<int> myvec2;
    myvec2.insert(myvec2.begin(),arr,arr+len);
    cout<<myvec2.front()<<"\n";
  return 0;
}
```

Salida:

```
2
```

back()

esta función nos permite seleccionar el dato se agregó de últimas en el vector.

Parámetros:

ninguno.

Valor de retorno:

retorna el valor que se encuentra en la última posición.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    int arr[]={2,3,4,5,10};
    int len=sizeof(arr)/sizeof(arr[0]);
    vector<int> myvec2;
    myvec2.insert(myvec2.begin(),arr,arr+len);
    cout<<myvec2.back()<<"\n";
  return 0;
}
```

Salida:

```
10
```

erase()

esta función nos permite eliminar un dato que se encuentre dentro del vector.

Parámetros:

se debe indicar la posición y/o especificar el inicio y el final de datos a eliminar.

Valor de retorno:

una vez encuentra el dato se eliminará del vector.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    int arr[]={2,3,4,5,10};
    int len=sizeof(arr)/sizeof(arr[0]);
    vector<int> myvec2;
    myvec2.insert(myvec2.begin(),arr,arr+len);
    //elimina la posicion tres ya que cuenta desde 0
    myvec2.erase(myvec2.begin()+2);
    std::vector<int>::iterator it=myvec2.begin();
    for(unsigned i=0;i<myvec2.size();++i){
        cout<<myvec2[i]<<" ";
    }
  return 0;
}
```

Salida:

```
2 3 5 10
```

push_back()

Con push back podremos ingresar datos, detrás del último dato.

Parámetros:

Ingresamos el elemento que queremos ingresar, este es añadido al final del vector.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    vector<int>vec;
    vec.push_back(4);
    vec.push_back(12);
    cout<<vec.back()<<endl;
}
```

Salida:

```
12
```

deque

deque es una cola con doble lado, por lo que podemos ingresar datos por delante o atrás, y esto es posible gracias a su tamaño dinámico.
Deque< OBJECT >name;

back()

accede al dato que se encuentra al final.

Parámetros:

Ninguno.

Valor de retorno:

Devuelve el dato n-simo.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    deque<int>dq={1,2,3,5,4,987};
    cout<<dq.back();
}
```

Salida:

```
987
```

front()

accede al dato que se encuentra al principio.

Parámetros:

Ninguno.

Valor de retorno:

Devuelve el primer dato de la cola.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    deque<int>dq={1,2,3,5,4,987};
    cout<<dq.front();
}
```

Salida:

```
1
```

size()

Determina la cantidad de datos que se encuentran en la cola.

Parámetros:

Ninguno.

Valor de retorno:

Devuelve el tamaño de la cola.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    deque<int>dq={1,2,3,5,4,987};
    cout<<dq.size();
}
```

Salida:

```
6
```

push_back()

Permite ingresar datos al final de la cola.

Parámetros:

Indicar cual es el nuevo elemento que se quiere agregar.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    deque<int>dq;
    dq.push_back(12);
    dq.push_back(10);
    cout<<dq.back();
}
```

Salida:

```
10
```

push_front()

Permite ingresar datos al inicio de la cola.

Parámetros:

Indicar cual es el nuevo elemento que se quiere agregar.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    deque<int>dq;
    dq.push_front(10);
    dq.push_front(12);
    cout<<dq.front();
}
```

Salida:

```
12
```

pop_back()

Elimina el dato que se encuentra en la última posición de la cola.

Parámetros:

Ninguno.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    deque<int>dq={1,2,3,5,4,987};
    dq.pop_back();
    cout<<dq.back();
}
```

Salida:

```
4
```

pop_front()

Elimina el dato que se encuentra de primeras en la cola.

Parámetros:

Ninguno.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
int main(){
    deque<int>dq={1,2,3,5,4,987};
    dq.pop_front();
    cout<<dq.front();
}
```

Salida:

2

list

List representa una lista doblemente enlazada por lo que podremos insertar valores por los dos lados izquierda o derecha.
list<OBJECT>name;

back()

nos permite acceder al dato que se encuentra en la parte de atrás de la lista.

Parámetros:

ninguno.

Valor de retorno:

nos retorna el valor que se encuentra de últimas en la lista.

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13};
  list<int> mylist (myints,myints+5);
  cout<<mylist.back()<<endl;
```

```
}
```

Salida:

```
13
```

front()

nos permite acceder al dato que se encuentra en la parte de adelante de la lista.

Parámetros:

ninguno.

Valor de retorno:

nos retorna el valor que se encuentra de primeras en la lista.

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13};
  list<int> mylist (myints,myints+5);
  cout<<mylist.front()<<endl;
}
```

Salida:

```
75
```

size()

nos permite saber lo largo de la lista.

Parámetros:

ninguno.

Valor de retorno:

devuelve el número de datos que se encuentra en la lista

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13};
  list<int> mylist (myints,myints+5);
  cout<<mylist.size()<<endl;
}
```

Salida:

```
5
```

pop_front()

elimina el primer elemento que se encuentra en la lista.

Parámetros:

ninguno.

Valor de retorno:

elimina el dato que este en el principio.

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
    int myints[] = {75,23,65,42,13};
    list<int> mylist (myints,myints+5);
    mylist.pop_front();
    cout<<mylist.front()<<endl;
}
```

Salida:

```
23
```

pop_back()

elimina el último elemento que se encuentra en la lista.

Parámetros:

ninguno.

Valor de retorno:

elimina el dato que esté al final.

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13};
  list<int> mylist (myints,myints+5);
  mylist.pop_back();
  cout<<mylist.back()<<endl;
}
```

Salida:

```
42
```

push_back()

con esta función podremos insertar datos a la lista por la parte de atrás.

Parámetros:

el valor que vamos a insertar a la lista, este debe coincidir con el tipo de dato de la lista.

Valor de retorno:

guarda el valor en la lista.

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13};
  list<int> mylist (myints,myints+5);
  mylist.push_back(12);
  cout<<mylist.back()<<endl;
}
```

Salida:

```
12
```

push_front()

con esta función podremos insertar datos a la lista por la parte de adelante.

Parámetros:

el valor que vamos a insertar a la lista, este debe coincidir con el tipo de dato de la lista.

Valor de retorno:

guarda el valor en la lista.

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13};
  list<int> mylist (myints,myints+5);
  mylist.push_front(123);
  cout<<mylist.front()<<endl;
}
```

Salida:

```
123
```

empty()

muestra si la lista está vacía.

Parámetros:

ninguno

Valor de retorno:

retorna un true si el tamaño de la lista es diferente de 0.

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13,12};
  list<int> mylist (myints,myints+6);
    //verifica que no este vacia
  while(!mylist.empty()){
    cout<<mylist.front()<<" ";
    mylist.pop_front();
    cout<<mylist.back()<<" ";
    mylist.pop_back();
    }
}
```

Salida:

```
75 12 23 13 65 42
```

sort()

organiza la lista.

Parámetros:

ninguno

Valor de retorno:

retorna la lista con los valores ordenados de menor a mayor

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13,12};
```

```
  list<int> mylist (myints,myints+6);
  mylist.sort();//ordena los valores
  while(!mylist.empty()){
    cout<<mylist.front()<<" ";
    mylist.pop_front();
    }
}
```

Salida:

```
12 13 23 42 65 75
```

remove()

elimina un valor que indiquemos, siempre que se encuentre en la lista

Parámetros:

le indicamos que valor es el que se desea eliminar.

Valor de retorno:

ninguno

Ejemplo:

```
#include<bits/stdc++.h>
using namespace std;
int main ()
{
  int myints[] = {75,23,65,42,13,12};
  list<int> mylist (myints,myints+6);
  mylist.remove(65); //dato a eliminar
  while(!mylist.empty()){
    cout<<mylist.front()<<" ";
    mylist.pop_front();
    }
}
```

Salida:

```
75 23 42 13 12
```

stack

LIFO stack
stack sigue la dinámica de LIFO (Last In First Out) esto quiere decir que el último que entra es el primero que sale.
stack<OBJECT>name;

push()

podremos añadir un dato a la pila(stack).

Parámetros:

Debemos colocar el valor correspondiente al tipo de dato del stack.

Valor de retorno:

ninguno.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    stack<int>mystack;
    mystack.push(1);
    mystack.push(3);
    mystack.push(5);
}
```

Salida:

```
NONE
```

top()

accede al elemento que se encuentra de primeras.

Parámetros:

ninguno

Valor de retorno:

devuelve el valor del dato.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    stack<int>mystack;
    mystack.push(1);
    mystack.push(3);
    mystack.push(5);
    cout<<mystack.top()<<endl;
}
```

Salida:

```
5
```

pop()

remueve el dato que se encuentra de primeras en la pila.

Parámetros:

ninguno.

Valor de retorno:

ninguno.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    stack<int>mystack;
    mystack.push(1);
    mystack.push(3);
    mystack.push(5);
    mystack.pop();
    cout<<mystack.top()<<endl;
}
```

Salida:

```
3
```

size()

determina la cantidad de datos que se encuentran en la pila.

Parámetros:

ninguno.

Valor de retorno:

devuelve un número con la cantidad de elementos.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    stack<int>mystack;
    mystack.push(1);
    mystack.push(3);
    mystack.push(5);
    mystack.pop();
```

```
    cout<<mystack.size()<<endl;
}
```

Salida:

```
2
```

empty()

verifica que no tenga elementos la pila.

Parámetros:

ninguno.

Valor de retorno:

retorna un true si el tamaño de la pila es 0.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    stack<int>mystack;
    mystack.push(1);
    mystack.push(3);
    mystack.push(5);
    mystack.push(2);
    while(!mystack.empty()){
        cout<<mystack.top()<<" ";
        mystack.pop();
    }
}
```

Salida:

```
2 5 3 1
```

queue

FIFO queue
queue sigue la filosofía de FIFO (First In First Out) de manera que el primero que entra es el primero que sale.
Queue< OBJECT >name;

push()

añade un elemento a la cola.

Parámetros:

el dato que queremos agregar.

Valor de retorno:

ninguno.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    queue<int>myqueue;
    myqueue.push(1);
    myqueue.push(2);
    myqueue.push(3);
    myqueue.push(4);
}
```

Salida:

```
NONE
```

front()

revisa el dato que se encuentra de primeras por orden de llegada.

Parámetros:

ninguno.

Valor de retorno:

accede al dato que llegó en primer lugar mientras no sea eliminado.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    queue<int>myqueue;
    myqueue.push(1);
    myqueue.push(2);
    myqueue.push(3);
    myqueue.push(4);
    cout<<myqueue.front()<<endl;
}
```

Salida:

```
1
```

back()

revisa el dato que se encuentra de últimas en la cola.

Parámetros:

ninguno.

Valor de retorno:

referencia al último dato que se agregó.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    queue<int>myqueue;
    myqueue.push(1);
    myqueue.push(2);
    myqueue.push(3);
    myqueue.push(4);
    cout<<myqueue.back()<<endl;
}
```

Salida:

```
4
```

size()

revisa la cantidad de datos que contiene la cola.

Parámetros:

ninguno.

Valor de retorno:

devuelve la cantidad de datos que se encuentran en la cola.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    queue<int>myqueue;
    myqueue.push(1);
    myqueue.push(2);
    myqueue.push(3);
    myqueue.push(4);
```

```
    cout<<myqueue.size()<<endl;
}
```

Salida:

```
4
```

pop()

remueve el dato que se encuentra de primera según su orden de llegada.

Parámetros:

ninguno.

Valor de retorno:

devuelve la cantidad de datos que se encuentran en la cola.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main() {
    queue<int>myqueue;
    myqueue.push(1);
    myqueue.push(2);
    myqueue.push(3);
    myqueue.push(4);
    myqueue.pop();
    cout<<myqueue.front()<<endl;
}
```

Salida:

```
2
```

empty()

verifica que la cola no tenga ningún elemento esto quiere decir que está vacía.

Parámetros:

ninguno.

Valor de retorno:

retorna un true si el tamaño de la cola es 0.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
```

```
using namespace std;

int main() {
    queue<int>myqueue;
    myqueue.push(1);
    myqueue.push(2);
    myqueue.push(3);
    myqueue.push(4);
    while(!myqueue.empty()){
        cout<<myqueue.front()<<" ";
        myqueue.pop();
    }
}
```

Salida:

```
1 2 3 4
```

Priority_queue

Priority queue

set

set organiza los datos a medida que van siendo insertados además de que no permite elementos repetidos dentro de él.
Set< OBJECT > name;

insert()

podremos agregar elementos al set.

Parámetros:

necesitamos el un dato correspondiente al tipo de objeto.

Valor de retorno:

inserta el elemento mientras no esté repetido.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main() {
    int arr[]={1,2,3,4,5,4,2,3,1,6};
    int len=sizeof arr / sizeof arr[0];
    set<int>myset;
    myset.insert(arr,arr+len);
    vector<int> myvec ;
    myvec.insert(myvec.begin(),myset.begin(),myset.end());
    for(int i=0;i<myvec.size();i++){
        cout<<myvec[i]<<" ";
```

```
    }
}
```

Salida:

```
1 2 3 4 5 6
```

count()

verifica que cuerto dato se encuentre dentro del objeto.

Parámetros:

requiere el valor que se desea buscar dentro del set.

Valor de retorno:

devuelve un 1 si el valor está presente de lo contrario es un 0.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main() {
    int arr[]={1,2,4,5,4,2,1,6};
    int len=sizeof arr / sizeof arr[0];
    set<int>myset;
    myset.insert(arr,arr+len);
    for(int i=0;i<4;i++){
        if(myset.count(i)!=0){
            cout<<i<<" se encuentra dentro del set"<<endl;
        }else  cout<<i<<" NO se encuentra dentro del set"<<endl;
    }
}
```

Salida:

```
0 NO se encuentra dentro del set
1 se encuentra dentro del set
2 se encuentra dentro del set
3 NO se encuentra dentro del set
```

erase()

podremos eliminar un dato que esté dentro del set.

Parámetros:

le indicamos que valor es el que se desea eliminar.

Valor de retorno:

ninguno

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main() {
    int arr[]={1,2,4,5,4,2,1,6};
    int len=sizeof arr / sizeof arr[0];
    set<int>myset;
    myset.insert(arr,arr+len);
    myset.erase(1);
    vector<int> myvec;
    myvec.insert(myvec.begin(),myset.begin(),myset.end());
    for(int i=0;i<myvec.size();i++){
        cout<<myvec[i]<<" ";
    }
}
```

Salida:

```
2 4 5 6
```

lower_bound()

esta función es un binary search y devuelve el número si se encuentra en el set.

Parámetros:

el valor a comparar con el contenedor.

Valor de retorno:

el iterador con el primer elemento que contiene el dato ingresado.

Ejemplo:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int main() {
    set<int>myset;
    for(int i=1;i<10;i++){
        myset.insert(i*5);
    }
    set<int>::iterator it=myset.lower_bound(15);
    myset.erase(it);
    for(int i:myset){
        cout<<i<<" ";
    }
}
```

Salida:

```
5 10 20 25 30 35 40 45
```

multiset

Multiple-key set

map

Map hace referencia a un diccionario ya que debemos crear la clave y su valor correspondiente.
Map < OBJECT, OBJECT >Nombre;

insert()

nos permite añadir elementos en el map.

Parámetros:

Crear un make_pair con la key y el valor : “map.insert(make_pair(key,value))”.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    map<char,string>mapa;
    mapa.insert(make_pair('a',"Angie"));
    mapa.insert(make_pair('b',"Bob"));
    mapa.insert(make_pair('d',"Daniel"));
    return 0;
}
```

Salida:

```
Ninguna.
```

First , Second

Set al ser una estructura de tipo pair, requiere un par de apuntadores para acceder a la key y los valores.

Parámetros:

Ninguno.

Valor de retorno:

Devuelve la key o el value si esta presente en la key.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
```

```
int main() {
    map<char,string>mapa;
    mapa.insert(make_pair('a',"Angie"));
    mapa.insert(make_pair('b',"Bob"));
    mapa.insert(make_pair('d',"Daniel"));
    for(auto it:mapa){
        cout<<it.first<<" : "<<it.second<<endl;
    }
    return 0;
}
```

Salida:

```
a : Angie
b : Bob
d : Daniel
```

find()

Permite buscar un valor en el set teniendo como parámetro la key.

Parámetros:

Recibe un tipo de dato correspondiente a la key.

Valor de retorno:

Devuelve el valor que se encuentra en la key.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    map<char,string>mapa;
    mapa.insert(make_pair('a',"Angie"));
    mapa.insert(make_pair('b',"Bob"));
    mapa.insert(make_pair('d',"Daniel"));
    cout<<"Key: a"<<" Value: "<<(mapa.find('a')->second)<<endl;
    cout<<"Key: d"<<" Value: "<<(mapa.find('d')->second)<<endl;
    return 0;
}
```

Salida:

```
Key: a Value: Angie
Key: d Value: Daniel
```

erase()

erase nos permite eliminar un valor que sea existente en el map.

Parámetros:

Recibe un tipo de dato correspondiente a la key que queremos eliminar, también podemos indicar cual es la posición del elemento con un iterator.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    map<char,string>mapa;
    mapa.insert(make_pair('a',"Angie"));
    mapa.insert(make_pair('b',"Bob"));
    mapa.insert(make_pair('d',"Daniel"));
    mapa.erase('a');
    map<char,string>::iterator it = mapa.find('b');
    mapa.erase(it);
    for(auto i:mapa){
        cout<<i.first<<" "<<i.second<<endl;
    }
    return 0;
}
```

Salida:

```
d Daniel
```

count()

count nos permite saber si una key contiene valor.

Parámetros:

La función recibe una key a la cual verifica su existencia.

Valor de retorno:

Devuelve un 1 si existe un valor para la key ingresada, de lo contrario si no se encuentra un valor retorna 0.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    map<char,string>mapa;
    mapa.insert(make_pair('a',"Angie"));
    mapa.insert(make_pair('b',"Bob"));
    mapa.insert(make_pair('e',"Edwin"));
    for(char c='a';c<='e';c++){
        if(mapa.count(c)>0){
            cout<<mapa.find(c)->second<<endl;
```

```
        }else{
            cout<<c<<" sin elementos"<<endl;
        }
    }
    return 0;
}
```

Salida:

Angie
Bob
c sin elementos
d sin elementos
Edwin

size()

esta función nos devuelve el número con la cantidad de keys que existen.

Parámetros:

Ninguno.

Valor de retorno:

Retorna la cantidad de datos que existen en el **map**.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    map<char,string>mapa;
    mapa.insert(make_pair('a',"Angie"));
    mapa.insert(make_pair('b',"Bob"));
    mapa.insert(make_pair('e',"Edwin"));
    cout<<mapa.size()<<endl;
    return 0;
}
```

Salida:

3

clear()

Con **clear** podemos eliminar todos los datos que se encuentran en el **map**.

Parámetros:

Ninguno.

Valor de retorno:

Ninguno.

Ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    map<char,string>mapa;
    mapa.insert({'a',"Angie"});
    mapa.insert({'e',"Edwin"});
    mapa.insert({'d',"Diego"});
    mapa.clear();
    cout<<mapa.size()<<endl;
    return 0;
}
```

Salida:

0

multimap

Multiple-key map

fstream

Input/output file stream class

stringstream

Input/output string stream

es un buffer en memoria que busca acceder a diferentes datos de forma homogénea.
permite la manipulación de un string nativo, además de poder convertir el string a diferentes objetos nativos. Esto es una ventaja dado que de manera interna el string no es capaz de hacerlo.

str()

nos permite convertir el objeto a un tipo string.

Parámetros:

necesita un objeto de tipo string donde aloja el contenido.

Valor de retorno:

devuelve el valor que se aloja en el stream buffer a un string

Ejemplo:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    stringstream ss;
    ss.str ("esto es un string");
    string str= ss.str();
    cout<<str<<endl;
    return 0;
}
```

Salida:

esto es un string

string to int

Ejemplo:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    string number = "123";
    stringstream ss;
    ss<<number;
    int n;
    ss>>n;
    cout<<n;
    return 0;
}
```

Salida:

123

string to double

Ejemplo:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    string number = "3.1416";
    stringstream ss;
    ss<<number;
    double n;
    ss>>n;
    cout<<n;
    return 0;
}
```

Salida:

3.1416

Uso de Macros

El uso de Macros en C++ nos permite ahorrar líneas de código renombrando algunas líneas de código, por lo que sirve para declarar y optimizar la cantidad de código (el uso de macros **no es obligatorio**).

Existen dos tipos de macros, la primera la utilizamos exclusivamente para dar un nuevo nombre a los objetos que existen o que nosotros creamos, esto lo hacemos con la palabra reservada **typedef Object newName** por ejemplo:

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
typedef long long int ll;
typedef vector<int> vI;
typedef pair<int,int> pii;
int main() {
    ll a;
    pii nuevoPar;
    vI vec;
    return 0;
}
```

En el ejemplo podemos apreciar como un nuevo vector de enteros podemos declararlo como **`vI`** y el nombre del objeto.

La segunda manera puede ser utilizada de igual manera que **typedef**, crear variables globales, renombrar ciclos, objetos, parámetros de objetos, para esto usamos **#define newName Object**.

```
#include <bits/stdc++.h>
#include <cstdlib>
#define fI(x,y,z) for(int i=x;i<y;i+=z)
#define x first
#define y second
#define point pair
using namespace std;
int main() {
    point<int,int>punto{3,4};
    cout<<punto.x<<" "<<punto.y;
    cout<<endl;
    fI(0,5,1){
        cout<<i<<" ";
    }
    return 0;
}
```

Esta manera es muy útil para fragmentos de código que utilizamos constantemente. De igual manera esto solo es utilizado para que el código se vea más corto o sencillo de entender.

Algoritmos útiles de java en C++

Tokenizer

```
#include<bits/stdc++.h>
#include <cstdlib>
using namespace std;
int main() {
    string str;
    getline(cin,str);
    string intermediate;
    vector<int> vec;
    stringstream check1(str);
    //tokenizer el cual podemos cambiar en el tercer parametro
      while(getline(check1, intermediate, ' ')) {
      vec.push_back(atoi(intermediate.c_str()));
    }
    for(int i=0;i<vec.size();++i){
        printf("%d ",vec[i]);
    }
    return 0;
}
```

ReplaceAll

```
#include <bits/stdc++.h>
using namespace std;
string ReplaceAll(string str, const string& from, const string& to) {
    size_t start_pos = 0;
    while((start_pos = str.find(from, start_pos)) != std::string::npos) {
        str.replace(start_pos, from.length(), to);
        start_pos += to.length();
    }
    return str;
}
int main(){
    string str="string antiguo";
    cout<<str<<endl;
    str=ReplaceAll(str,"antiguo","nuevo");
    cout<<str<<endl;
    str=ReplaceAll(str," ","");
    cout<<str<<endl;
    return 0;
}
```

Programación Orientada a Objetos (POO)

La Programación orientada a objetos nos permite organizar y agrupar diferentes objetos, esto supone que debe soportar herencias y polimorfismo. Por lo que nos da la facilidad para modificar, crear nuevos objetos.

Para la creación de objetos en C++ se utiliza **class** o **struct**, las dos funcionan perfectamente pero el grado de dificultad difiere entre una y otra.

El siguiente ejemplo contiene un objeto persona uno creado con **class** y otro con **struct** para saber cómo se usan.

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

///struct
struct Spersona{
    string nombre;
    int edad;
};

///class
class Cpersona{
public:
    string nombre;
    int edad;
    Cpersona(string,int);
};
///constructor
Cpersona::Cpersona(string name,int old){
    nombre=name;
    edad=old;
}
int main (){
    Spersona sp{"Edwin",18};
    Cpersona cp{"Andres",19};
    cout<<"Struct\n"<<sp.nombre<<" "<<sp.edad<<endl;
    cout<<"Class \n"<<cp.nombre<<" "<<cp.edad<<endl;
}
```

Como podemos ver **struct** genera automáticamente el constructor, este nos permite asignar datos en los atributos del objeto. Por lo que el "ahorro de líneas" es notorio.

```
Struct
Edwin 18
Class
Andres 19
```

La salida para ambos objetos es exitosa, para poder ingresar los datos se puede hacer de distintas maneras, en el ejemplo apreciamos la encuentra en <**initializer_list**> esta encargada de inicializar

el objeto, teniendo como parámetros los distintos atributos del objeto (disponible desde **C++11**). También podemos llenar el objeto indicando atributo por atributo de la siguiente manera.

```
Spersona sp;
sp.nombre="Edwin";
sp.edad=18;
Cpersona cp;
cp.nombre="Andres";
cp.edad=19;
```

Pero como sucede en diferentes lenguajes para poder asignar uno por uno debemos hacer un constructor vacío en el caso de utilizar **class**.

Siguiendo esta norma el objeto queda de la siguiente manera

```
///struct
struct Spersona{
    string nombre;
    int edad;
};

///class
class Cpersona{
public:
    string nombre;
    int edad;
    Cpersona();
    Cpersona(string,int);
};
Cpersona::Cpersona(){

}
Cpersona::Cpersona(string name,int old){
    nombre=name;
    edad=old;
}
```

Los objetos se pueden utilizar como si fuera un tipo de dato nativo del lenguaje, esto sugiere que podremos hacer un vector que contendrá el objeto.

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
struct person{
    string nombre;
    int edad;
};
vector<person>personas;
int main (){
    personas.push_back({"Edwin",19});
    personas.push_back({"Diana",20});
    personas.push_back({"Diego",23});
    personas.push_back({"Angie",18});
    for(int i=0;i<personas.size();i++){
        cout<<personas[i].nombre<<" "<<personas[i].edad<<endl;
```

```
    }
}
```

Existen formas de ordenar el vector de objetos para este caso podemos ordenarlo por la edad, podemos hacer una función externa o interna. Para ello utilizaremos la función sort que se encuentra en <vector> y por último parámetro le indicamos la función externa.

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
struct person{
    string nombre;
    int edad;
};
bool order(person a,person b){
    return a.edad<b.edad;
}
vector<person>personas;
int main (){
    personas.push_back({"Edwin",18});
    personas.push_back({"Diana",19});
    personas.push_back({"Angie",17});
    personas.push_back({"Diego",23});
    sort(personas.begin(),personas.end(),order);
    for(int i=0;i<personas.size();i++){
        cout<<personas[i].nombre<<" "<<personas[i].edad<<endl;
    }

}
```

```
Angie 17
Edwin 18
Diana 19
Diego 23
```

Graphical User Interface

GUI por sus siglas en ingles se refiere a (**graphical user interface**) esto refiere a que ahora el código es más amigable con el usuario con ayuda de formulario, objetos, imágenes y demás objetos gráficos para representar, para el caso específico de C++ es cómodo utilizar el entorno de **visual studio** con la extensión denominada **C++/CLR Windows Forms**.

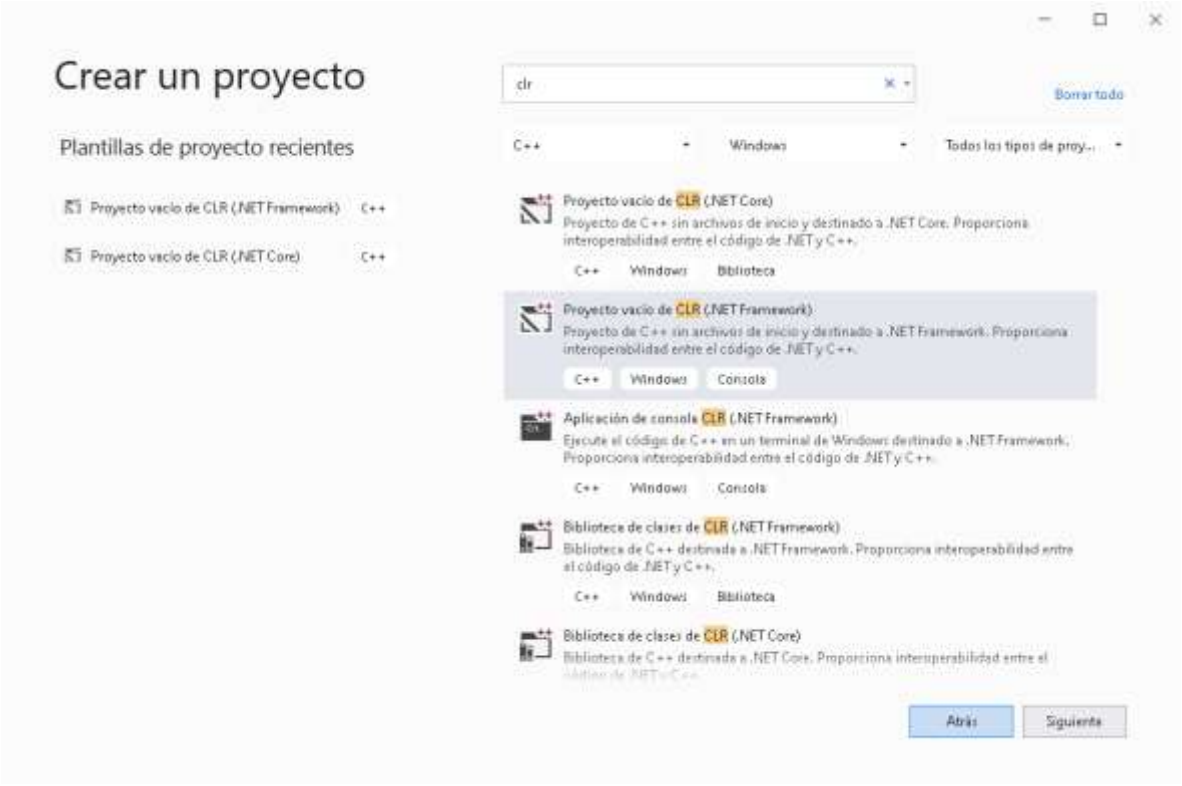

Ilustración 7 Creación de proyecto Visual Studio

Escogemos un proyecto CLR vacío y procedemos a colocarle el nombre del mismo, para este caso el nombre es “GUI” y click en crear.

Nos dirigimos a la barra superior y seleccionamos **proyecto**-**Propiedades**-**Vinculador**-**Sistema** y en **subsistema** seleccionamos Windows.

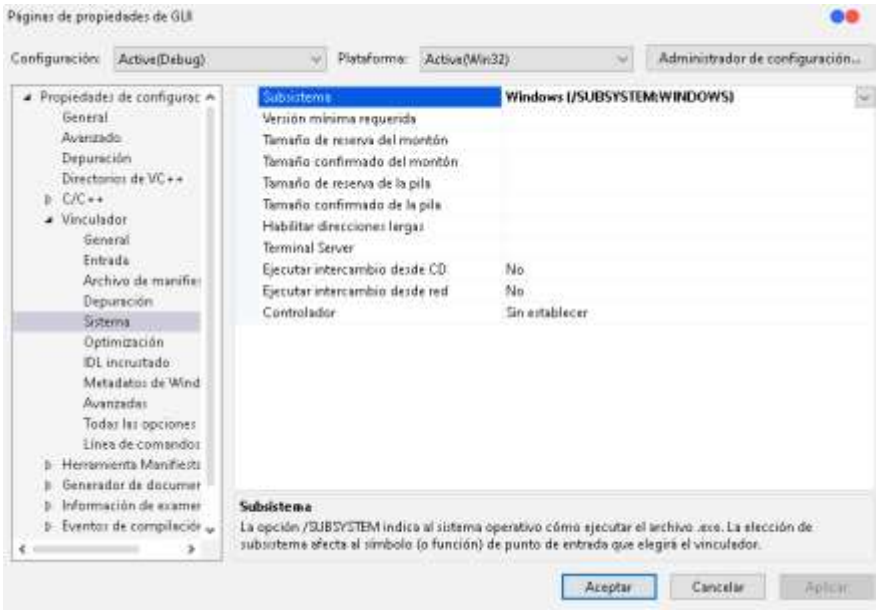

Ilustración 8 Ajustes del GUI I

En avanzadas seleccionamos punto de entrada y escribimos **main**.

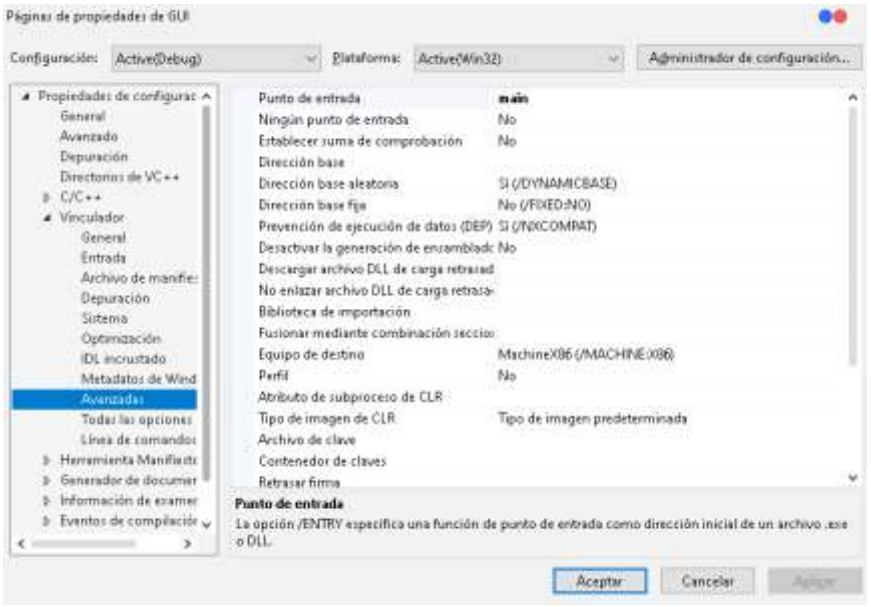

Ilustración 9 Ajustes de GUI II

Aplicamos y aceptamos. Volvemos a seleccionar el apartado de **proyecto** y buscamos **Agregar nuevo Elemento**.

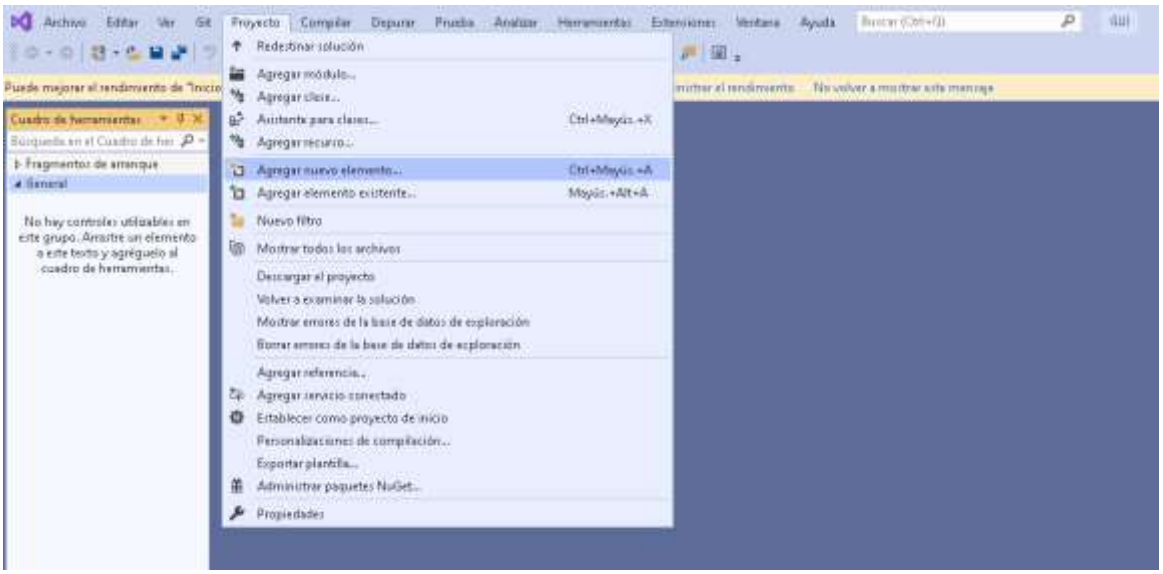

Ilustración 10Nuevos Elementos

En la siguiente pestaña seleccionamos **UI** y por último **Windows Forms**.

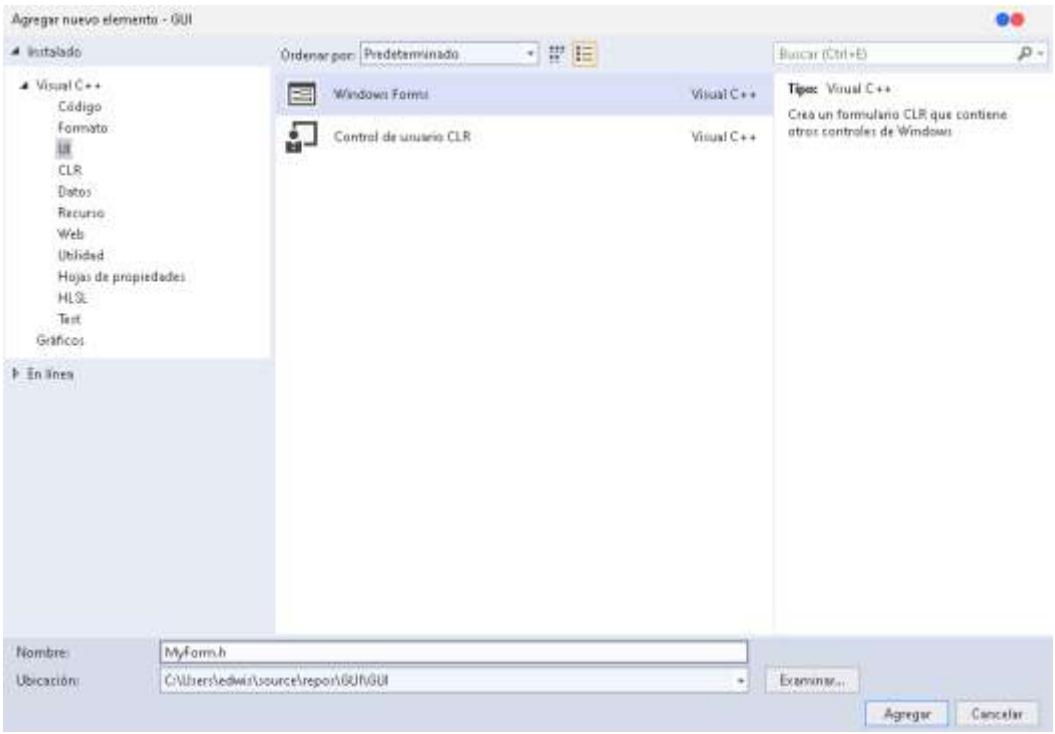

Ilustración 11Agregar Windows Form

Una vez creado nos aparecerá un error bastará con cerrar la pestaña, abrimos el "MyForm.cpp" y copiamos el siguiente código y reemplazamos "**NameForm**" y" **NameProject**" por los nombres del form y el nombre del proyecto:

```
#include "NameForm.h"
using namespace System;
using namespace System::Windows::Forms;
```

```
[STAThreadAttribute]
void main(array<String^>^ args) {
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);
	NameProject::MyForm form;
	Application::Run(% form);
}
```

Reiniciamos visual studio y podemos abrir “MyForm.h” en modo diseño, allí podemos agregar los elementos para el form arrastrándolos desde la **toolbox** (Ctlr+Alt+X).

Para el primer ejemplo agregamos: 1 Botón, 1 Label. seguido damos doble click al botón creado y copiamos lo siguiente.

```
	label1->Text = "Hello World";
```

De esta manera cambiamos el valor que se encuentra en el label.

Ahora para realizar una suma creamos: 2 TextBox , 1 Boton , 1 Label, doble click sobre el botón y realizamos lo siguiente.

```
	int numero1 =Convert::ToInt16(textBox1->Text);
	int numero2 = Convert::ToInt16(textBox2->Text);
	int resultado = numero1 + numero2;
	label1->Text = Convert::ToString(resultado);
```

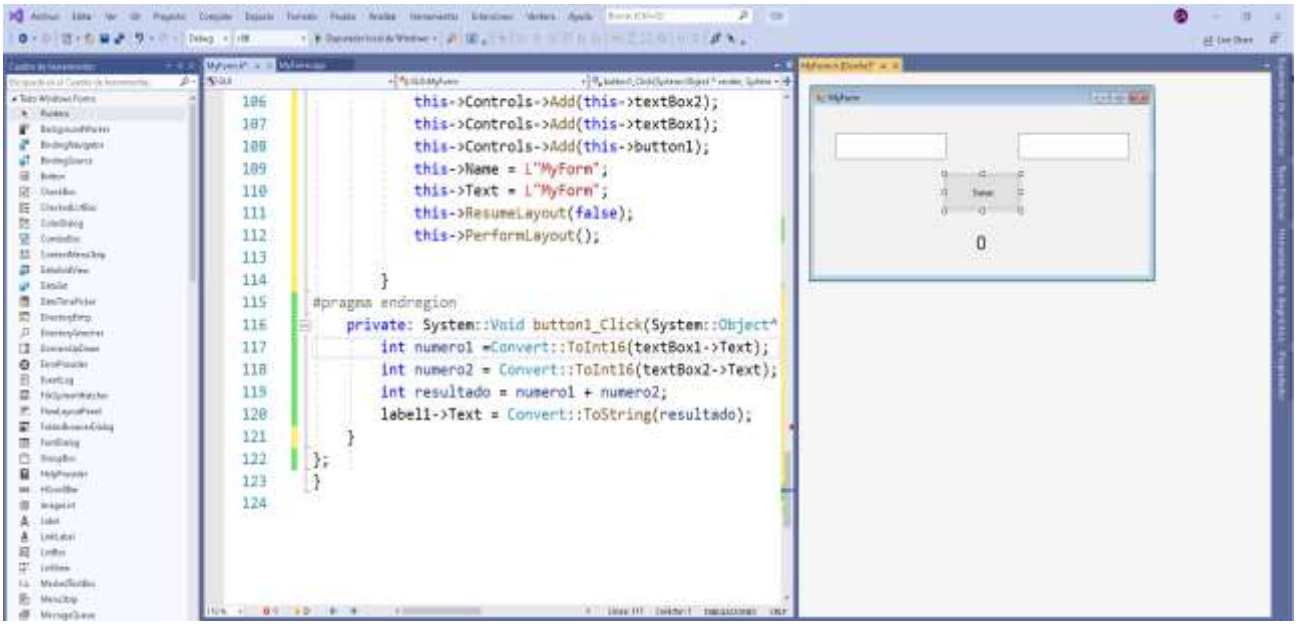

Ilustración 12 Ejemplo de Form I

Podemos editar los valores del botón y el label desde sus propiedades.

Repitiendo el paso anterior creamos varios botones con las operaciones aritméticas y solo cambiamos el signo dependiendo de la operación que estamos realizando.Para que el código sea un poco amigable y menos complicado crearemos la siguietes macros:

```
#define CtoDobuble(a) Convert::ToDouble(a)
#define CtoInt(a) Convert::ToInt16(a)
#define CtoString(a) Convert::ToString(a)
```

Usando las macros ya creadas el código quedaría de la siguiente manera:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
	int numero1 = CtoInt(textBox1->Text);
	int numero2 = CtoInt(textBox2->Text);
	int resultado = numero1 + numero2;
	label1->Text = CtoString(resultado);
}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
	int numero1 = CtoInt(textBox1->Text);
	int numero2 = CtoInt(textBox2->Text);
	int resultado = numero1 - numero2;
	label1->Text = CtoString(resultado);
}
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
	int numero1 = CtoInt(textBox1->Text);
	int numero2 = CtoInt(textBox2->Text);
	int resultado = numero1 * numero2;
	label1->Text = CtoString(resultado);
}
private: System::Void button4_Click(System::Object^ sender, System::EventArgs^ e) {
	Double numero1 = CtoDobuble(textBox1->Text);
	Double numero2 = CtoDobuble(textBox2->Text);
	Double resultado = numero1 / numero2;
	label1->Text = CtoString(resultado);
}
```

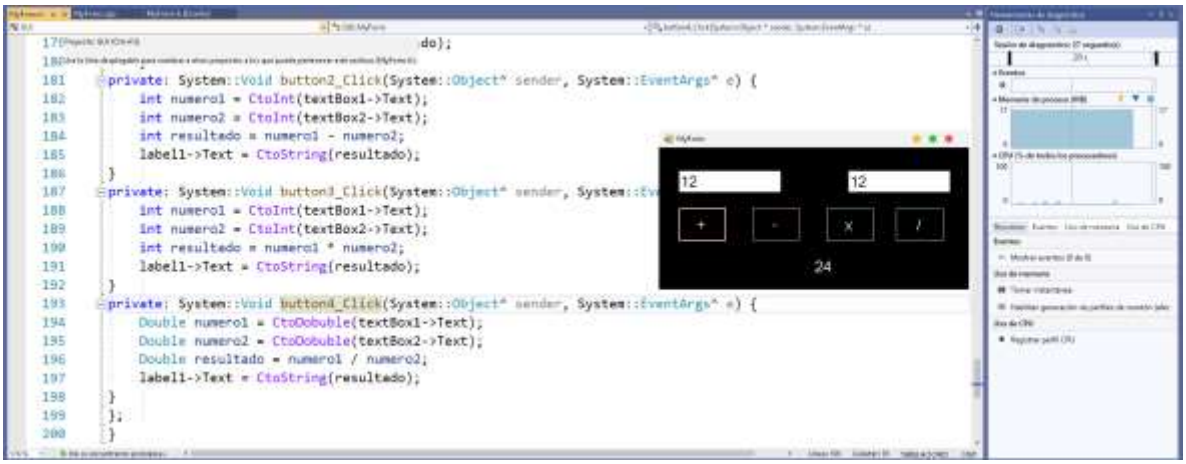

Ilustración 13 Ejemplo de Form II

Para poder que los botones tengan el contorno y el valor del mismo color vamos a la propiedad de flatStyle y escogemos flat, seguido procedemos a escoger el color

Lista de tablas

Lista de ilustraciones