

Entrada/salida

Estándar

Salida
<pre>#include <iostream> // Funciones de entrada/salida estándar // Funciones cout, cin using namespace std; // Con esto no haría falta utilizar std:: int main(int argc, char** argv) { // endl fuerza una nueva línea std::cout << "Hello World" << endl; cout << "Hola"; cout << "Hola2"; float f1 = 1.111111; float f2 = 1.111111; printf("f1 + f2 = %.3f\n", f2 + f2); // También en <iostream> int num = 10; bool siONo = (num < 100) ? true : false; // Para que imprima "true" o "false" en lugar de 0s y 1s en la salida cout.setf(ios::boolalpha); cout << num << " < 100? --> " << siONo << endl; return 0; }</pre>
Entrada
<pre>#include <cstdlib> #include <iostream> // Funciones de entrada/salida estándar // Funciones cout, cin using namespace std; int main(int argc, char** argv) { string s = "Introduce un número: "; string num1, num2; cout << s; cin >> num1; cout << "Introduce otro número: "; cin >> num2; // stoi() --> string a entero (función de C++) int iNum1 = stoi(num1); // atoi() es la función equivalente de C, por eso convertimos primero</pre>

```

// el string en un puntero de chars
int iNum2 = atoi(num2.c_str());
// string.c_str() --> convierte un string en un puntero de chars, para usarse
// en las funciones que lo requieren
printf( "%s + %s = %d\n", num1.c_str(), num2.c_str(), (iNum1 + iNum2) );

return 0;
}

```

Tipos de datos

bool	
char	
int	
float	
double	
string	<pre> string s1 = "Hola"; string s2 ("Hola"); string* s3 = new ("Hola"); </pre>

```

auto variable = true; // Asigna el tipo de dato automáticamente

```

Límites

```

#include <iostream>

int main(int argc, char** argv) {
    cout << "SHORT INT MIN: " << numeric_limits<short int>::min() << endl;
    cout << "SHORT INT MAX: " << numeric_limits<short int>::max() << endl;
}

```

string

```

#include <cstdlib>
#include <iostream> // Funciones de entrada/salida estándar

```

```
// Posibilidad de incluir valores por defecto
double AddNumbers(double num1 = 0, double num2 = 0){
    return num1 + num2;
}
```

Lambdas

```
#include <cstdlib>
#include <iostream>
#include <sstream> // Funciones de entrada/salida estándar
#include <ctime>
#include <vector>

// Funciones cout, cin
using namespace std;

vector<int> GenerateRandomVector( int num, int min, int max, bool repeatNums );

int main() {
    vector<int> vec = GenerateRandomVector( 10, 50, 80, false );
    // Primer parámetro: iterador de inicio
    // Segundo parámetro: iterador de fin
    // Tercer parámetro: función lambda para ordenar ascendentemente
    sort( vec.begin(), vec.end(), [](int a, int b){ return a < b; } );
    for( auto val: vec ){
        cout << val << endl;
    }
}

vector<int> GenerateRandomVector( int num, int min, int max, bool repeatNums ){
    vector<int> vec;
    srand(time(NULL));
    int randVal = 0;
    for( int i = 0; i < num; i++ ){
        do{
            randVal = min + rand() % (max+1 - min);
        }
        // find() --> si lo encuentra devuelve un iterador diferente a "vec.end()"
        while( !repeatNums && find(vec.begin(), vec.end(), randVal) != vec.end() );
        vec.push_back(randVal);
    }
    return vec;
}
```

Sobrecarga de operadores

```
#include <cstdlib>
#include <iostream>
#include <sstream> // Funciones de entrada/salida estándar

// Funciones cout, cin
using namespace std;

class Box{
protected:
    double length, width, breadth;

public:
    Box(){
        length = 1, width = 1, breadth = 1;
    }

    Box( double l, double w, double b ){
        length = l, width = w, breadth = b;
    }

    // Sobrecarga del operador de incremento prefijo (ej. ++box)
    Box& operator ++(){
        length++;
        width++;
        breadth++;
        return *this;
    }

    // Sobrecarga del operador de incremento sufijo (ej. box++)
    Box operator ++(int){
        // Guardamos los datos primero
        Box b(length, width, breadth);
        length++;
        width++;
        breadth++;
        // Devolvemos los datos sin incrementar
        return b;
    }

    /*
    * El modificador "friend" permite definir una función que en realidad
    * está fuera de la clase pero le permite acceder a todos los miembros
    * privados y protegidos de la misma.
    *
    * Esta función sobrecarga el operador "<<" para poder imprimir como queramos
    * el objeto utilizando el stream "cout"
    */
};
```