Consejos de estilo

Jose Oncina, José Luis Verdú y Mikel Forcada

15 de marzo de 2023

1. Consejos

1.1. Usa nombres significativos para tus variables y funciones

Los nombres deben indicar claramente el propósito y el uso de la variable o función. Evita abusar de la abreviación en los nombres de las variables. Aunque es posible que conozcas el significado de la abreviación, otros programadores pueden tener dificultades para entenderla.

Mal:

```
bool f(int n) {
   if (n < 2)
     return false;

   for (int i = 2; i < n; ++i)
      if (n % i == 0)
        return false;

   return true;
}
```

Mejor

```
bool isPrime(int number)
{
   if (number < 2)
     return false;
   for (int i = 2; i < number; ++i)
      if (number % i == 0)
        return false;
   return true;
}
```

1.2. Usa comentarios para explicar el código cuando sea necesario

Sobre todo si estás haciendo algo que no es obvio o si el código es complicado.
Mal:

```
// Function to add two numbers
int add(int a, int b)
{
   return a + b; // Returns the sum of a and b
}
```

En este ejemplo, el comentario simplemente repite lo que la línea de código ya explica claramente. El comentario no es necesario y solo agrega ruido al código.

1.3. Divide el código en funciones más pequeñas y manejables

Esto hace que el programa sea más fácil de leer y también facilita el mantenimiento del código.

1.4. Usa la biblioteca estándar de C++ cuando sea posible

La mayoría de las veces, la biblioteca estándar ya proporciona soluciones optimizadas y seguras para problemas comunes.

1.5. Sangra bien tu código

Existen muchas normas de sangrado, elige una y sé consistente.
Hay algunos editores que redefinen el tabulador, por tanto, lo que a ti te parece bien sangrado puede no estarlo en otro editor. Para estar seguro de que tu código está bien sangrado comprueba que la salida de

```
cat fichero.cc
```

también también está bien sangrada.
Usa espacios en blanco para separar claramente los elementos del código. Esto hace que el código sea más fácil de leer.

1.6. Usa de forma correcta las estructuras de programación

No uses un bucle `while` cuando es un bucle `for`.
Mal:

```
...
int i = 0;
int acc = 0;
while( i < 10 ) {
```

```
  acc += v[i];
  i++;
}
// "i" is never used again
...
```

Mejor:

```
...
int acc = 0;
for( int i = 0; i < 10; ++i )
  acc += v[i]
...
```

En general, cuando se está contando y la variable de control no la vas a volver a usar, es mejor usar un `for`.

1.7. Usa de forma correcta los tipos de datos

No uses `float` o `double` para almacenar valores enteros. Si necesitas mas rango del que te da `int` o `long` usa `long long`.

1.8. Declara las variables cuando se necesitan, no antes

En `C` es obligatorio declarar las variables al principio de una función. En `C++` se introdujo la posibilidad de declararlas en cualquier punto para mejorar la comprensión de los programas.

Mal:

```
int acc = 0;
... // lots of code
for( int i = 0; i < n; ++i)
  acc += v[i];
```

Mejor:

```
...
int acc = 0;
for( int i = 0; i < n; ++i )
  acc += v[i];
```

1.9. No pases variables "grandes" por valor a menos que sea absolutamente necesario

Mal:

```
int acc( vector<int> v ) {
  int acc = 0;
  for( int i = 0; i < v.size(); ++i )
```

```
    acc += v[i]
  return acc;
}
```

Mejor:

```
int acc( const vector<int> &v ) {
  int acc = 0;
  for( int i = 0; i < v.size(); ++i )
    acc += v[i]
  return acc;
}
```

1.10. Analizando argumentos

Ejemplo 1.1. *Hacer un programa que busque coches que cumplan ciertas condiciones en una base de de datos almacenada en un fichero. El programa se llamará* `search_cars` *y admitirá los siguientes argumentos:*

- `-f file`*: fichero donde está la base de datos (obligatorio)*
- `-l length`*: longitud mínima del coche (*`double`*, obligatorio)*
- `-a age`*: edad mínima del coche (*`int`*, opcional, 0 por defecto)*
- `-s`*: muestra solo coches deportivos (opcional)*
- `-h`*: imprime un mensaje de ayuda y termina*

```
#include <fstream>
#include <string>
#include <limits>

using namespace std;

const double SENTINEL = numeric_limits<double>::max();

void show_usage() {
    cout << "Usage: search_car [-s] [-a age] -l length -f file\n"
            "\n"
            "Search for a car in a database.\n"
            "\n"
            "  -s          print only sport cars\n"
            "  -a age      print only sports older than 'age'\n"
            "  -l length   print only cars longer than 'length'\n"
            "  -f file     set database to 'file'\n"
            "  -h          print this help message\n" ;
}
```

```
int main( int argc, char *argv[] ) {

    bool is_sport_car = false;
    int age = 0;       // default value for age
    double length = SENTINEL;
    string file_name;

// Parsing arguments

    for( int i = 1; i < argc; i++ ) {
        string arg = argv[i];

        if( arg == "-s" ) {
            is_sport_car = true;

        } else if( arg == "-a" ) {
            i++;
            if( i >= argc ) {
                cerr << "ERROR: can't read the age." << endl;
                exit(EXIT_FAILURE);
            }
            try {
                age = stoi(argv[i]);
            } catch( ... ) {
                cerr << "ERROR: invalid age" << endl;
                exit(EXIT_FAILURE);
            }

        } else if( arg == "-l" ) {
            i++;
            if( i >= argc ) {
                cerr << "ERROR: can't read the length." << endl;
                exit(EXIT_FAILURE);
            }
            try {
                length = stod(argv[i]);
            } catch( ... ) {
                cerr << "ERROR: invalid length" << endl;
                exit(EXIT_FAILURE);
            }

        } else if( arg == "-f" ) {
            i++;
            if( i >= argc ) {
                cerr << "ERROR: can't read file name." << endl;
```

```
                exit(EXIT_FAILURE);
            }
            file_name = argv[i];

        } else if( arg == "-h" ) {
            show_usage();
            exit(EXIT_SUCCESS);
         } else {
             cerr << "ERROR: unknown option '" << arg << "'." << endl;
             show_usage();
             exit(EXIT_FAILURE);
         }

    }

// Processing parameters

    if( length == SENTINEL ) {
        cout <<  "ERROR: mandatory car length not specified" << endl;
        exit(EXIT_FAILURE);
    }

    if( file_name.empty() ) {
        cerr << "ERROR: missing file name." << endl;
        exit(EXIT_FAILURE);
    }

    ifstream is(file_name);

    if( !is ) {
            cerr << "ERROR: can't open file: " << file_name << endl;
            exit(EXIT_FAILURE);
    }

//  Searching cars ...

    cout << "Searching in data file: " << file_name << endl;
    if( is_sport_car )
        cout << "Searching for sport cars" << endl;

    cout << "Searching for cars older than "
         << age << " years." << endl;
    cout << "Searching for cars longer than "
         << length << " meters." << endl;

    return 0;
```

```
}
```

1.11. Leyendo datos de un fichero de texto

Ejemplo 1.2. *Leer una matriz de enteros de un fichero de texto. La primera línea del fichero contiene dos enteros, separados por espacios en blanco, que indican las filas y columnas de la matriz. En líneas sucesivas vienen los elementos de las filas de la matriz, también separados por espacios en blanco.*

```
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

int main() {
    const string file_name = "matrix.dat";

    ifstream is(file_name);

    if( !is ) {
        cerr << "ERROR: can't open file '" << file_name << "'." <<endl;
        exit(EXIT_FAILURE);
    }

    int rows, cols;
    is >> rows >> cols;
    vector< vector<int>> mat( rows, vector<int>( cols ));
    for( int i = 0; i < rows; i++ )
        for( int j = 0; j < cols; j++ )
            is >> mat[i][j];


    for( int i = 0; i < rows; i++ ) {
        for( int j = 0; j < cols; j++ )
            cout << mat[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```