TEC | Tecnológico de Costa Rica

Estándar de programación C/C++

A continuación, se presentan reglas para la programación en C/C++ que se deben seguir para la entrega de los proyectos del curso. En cada caso se presenta de forma tabulada un conjunto de lineamientos para cada elemento seguido por ejemplos de aplicación de estas en algunos casos puntuales con el fin de aclarar el uso sin ser todos los casos posibles de aplicación de la regla.

Documentación

Archivos de código fuente

- Todo archivo fuente o de encabezado debe tener un encabezado estándar, siguiendo la plantilla que se presenta a continuación:

```
/*****Datos administrativos**********************
 * Nombre del archivo:
 * Tipo de archivo:
 * Proyecto:
 * Autor:
 * Empresa:
 *****Descripción*********************************
 *
 *
 *****Versión*************************************
 * ## | Fecha y hora | Autor
 *
 ************************************************/
```

Clases

- Al inicio de toda clase, debe describirse la misma siguiendo los lineamientos de la plantilla que se presenta a continuación:

```
/*****Nombre**************************************
 *
 *****Descripción*********************************
 *
 *****Atributos***********************************
 *
 *****Métodos*************************************
 *
 ************************************************/
```

Funciones, métodos y similares

- Cada uno de estos elementos debe tener una introducción con la siguiente plantilla:

```
/*****Nombre**************************************
 *
 *****Descripción*********************************
 *
 *****Retorno*************************************
 *
 *****Entradas************************************
 *
 ************************************************/
```

Nombres

Constantes globales

- Deben escribirse con todas sus letras en mayúscula.
- Las palabras deben separarse mediante un guion bajo (“_”).

```
const int A_GLOBAL_CONSTANT = 5;
const int SIZE = 254;
```

Variables de tipo puntero

- Todo puntero debe tener el prefijo “p”.
- El asterisco debe estar junto al nombre de la variable y no al tipo.

```
char *pNombre;
int *pNotas, promedio, media;
//Nótese que solo pNotas es puntero
```

Variables globales

- Deben tener el prefijo “g”.
- Si estas variables son también punteros, se deben usar ambos prefijos.

```
int gIdentificador;
lista *gpUsuarios;
```

Clases

- Usar letras mayúsculas al inicio de cada palabra que conforme el nombre, el resto deben ser minúsculas. El primer caracter siempre será mayúscula. No utilizar barras bajas (“_”).
- La clase debe ser nombrada con respecto a lo que representa. Si la clase no representa un solo elemento, probablemente obedezca a errores de diseño.
- Nombres que tengan más de tres palabras son una pista de que el diseño puede contener varias entidades en una sola clase.

```
class UnaClase
```

Atributos de clase

- Todo atributo de clase debe tener el prefijo “a” para identificarlo en cada lugar que se utilice y poder distinguirlo de parámetros y otras variables
- Después de este prefijo, se utilizan las mismas reglas que para las clases.
- Si una variable de clase es también de tipo puntero, se deben usar ambos prefijos.

```
class UnaClase {
private:
	int aVarA;
	int aVarB;
	char *apString;
}
```

Métodos y funciones

- Las reglas de nombre para las clases aplican.
- Los métodos y funciones usualmente realizan una acción, el nombre puede ser más claro al agregar esta acción como la primera parte que lo compone.
- Las clases usualmente se nombran usando pronombres, al agregar las acciones a los nombres de los métodos, estas se diferencian por empezar siempre con un verbo.

```
void MostrarEstado ();
```

```
        void ValidarAtributos ();
        int RetornarPeso ();
```

Parámetros

- La primera letra debe ser minúscula.
- Todas las palabras posteriores a la primera deben indicarse en mayúscula

```
        int suma (int varA, int varB);
```

Variables locales

- Todas las letras deben ser minúsculas.
- En caso de tener más de una palabra, se deben separar con un guion bajo (“_”).

```
int UnaClase::Suma (int varA, int varB) {
        int temporal = varA + varB;
        char *message[SIZE];
        std::cout << varA << ‘+’ << varB << ‘=’ << temporal << std::endl;
        return temporal;
}
```

Directiva define del preprocesador

- Antes de crear cualquier clase en un archivo de encabezado, se debe utilizar la directiva define para asegurarse que existe una única definición de la clase en el programa.
- Para utilizarla debe incluirse la directiva ifndef correspondiente.
- Al definir la variable del preprocesador, esta debe nombrarse usando el nombre del archivo en mayúsculas, con los puntos sustituidos con guiones bajos, incluyendo la extensión.

```
// Contenidos del archivo NodoLista.h
#ifndef NODOLISTA_H
#define NODOLISTA_H
class NodoLista {
        ...
}
#endif
```

Archivos

- Todo archivo de encabezado debe contener la declaración e implementación de una sola clase.
- El archivo debe llamarse igual que la clase que contiene y utilizar la extensión h
- Inmediatamente después del encabezado de documentación del archivo debe hacerse la validación detallada en el punto anterior.
- Los archivos de implementación de los programas deben tener la extensión C o CPP según corresponda a la utilización de C o C++ como lenguaje de programación.
- Si la compilación del programa requiere el enlace de 2 o más archivos de implementación, debe entregarse un archivo *make* con el que se pueda hacer la compilación en un solo paso respetando los pasos de construcción establecidos por el programador
- El archivo que contiene el método de ejecución principal debe tener Main como parte de su nombre

```
// Contenidos del archivo ProgramaMain.cpp
#include <iostream>
#include “NodoLista.h”

int main (int argc, char **argv) {
        ...
} // Fin de función main
```

Formato

Llaves e identación

- La apertura de llaves de bloque debe hacerse siempre en la misma línea que la sentencia que la utiliza.
- Las líneas de código que se encuentran entre cada par de llaves tendrán una tabulación adicional que la línea donde está la apertura del bloque.
- Al cerrar el bloque, la llave debe ubicarse con la misma identación que tenía la línea de apertura.
- La llave de cierre debe tener un comentario que indique qué es lo que está cerrando

```
if (condicion1) {
	while (condicion2) {
		...                                ...
	} // while (condicion2)
} // if (condicion1)
```

Sentencias de control sin llaves

- Toda sentencia de control que afecte una única línea de código debe escribirse en una sola línea que contenga la instrucción de control y la sentencia que afecta.

```
if (1 == unvalor) unvalor = 2;

while (esperando) LlamarFuncion ();

for (int i = 0; i < SIZE; i++) FuncionArreglo (i);
```

Condicionales multiples

- Los condicionales múltiples deben escribirse siempre con llaves y el comentario debe ir al final de las condiciones, no es necesario en cada cierre de bloque.

```
if (condicion1) {
} else if (condicion2) {
} else {
}
```

Una instrucción o variable por línea

- Cada línea debe contener únicamente una instrucción ejecutable o la declaración de una variable.

```
int a = 2;
int b = 3;
int c = 4;
aSuma = a + b + c;
aMultiplicacion = a * b * c;
```

Control de acceso

- Los declaradores de acceso de una clase van en línea con la sentencia de apertura de la clase.
- El orden de los declaradores debe ser específicamente private, protected, public.

```
class UnaClase {
private:
	int aVarA;
	int aVarB;
public:
	int RetornarVarA ();
	int RetornarVarB ();
```

```
                void AsignarVarB (int);
        }
```

Mejores prácticas de programación

Inicializar todas las variables

- Ya sea en el momento de declararla o en un constructor de una clase, una variable siempre debe recibir un valor inicial válido o que represente la invalidez de la no asignación de un valor válido.

Inicialización en los constructores

- Los constructores deben usarse únicamente para dar valores iniciales a los atributos de una clase.
- La ejecución de procesos o funciones que dependen de la preexistencia de variables y valores en memoria pueden fallar al ejecutarse en el constructor y con esto podría fallar la creación del objeto y el programa por completo.

Utilice constantes

- Cuando necesite valores literales en el código, utilice constantes en lugar de los números.
- Al usar constantes se le da significado a las expresiones, es muy sencillo olvidar por qué se utilizaba un valor específico en el código, contrario a lo que puede agregar un nombre sencillo de una constante.

```
#define   SIZE (22)
const int MAX = 19;
```
