Table of Contents

Estándares de Desarrollo del Proyecto

Motor de entrada y salida de datos y simulación

Sobre los archivos

Escribir los nombres de los archivos con letras minúsculas separando cada palabra mediante el carácter "■underscore".

Los nombres de los archivos de cabecera deben terminar con la extensión ".H" mientras que los archivos de fuentes deben terminar con la extensión ".C".

Si un par de archivos (cabecera y fuente) son contenedores de una clase, éstos deben tener el mismo nombre de la clase que contienen con la extensión respectiva.

Por cada clase deben existir un par de archivos para su definición (archivo de cabecera) y su implementación (archivo fuente) con excepción de las clases de tipo plantilla que deben ser definidas solamente en un archivo de cabecera.

Cada archivo que se cree debe comenzar con un bloque de comentario multilínea que contenga el texto de la licencia, por ejemplo:

Code highlighting:

```
/*
  Texto de la licencia
*/
```

Luego se coloca otro bloque de comentario multilínea que contenga una breve descripción del archivo, autor o autores y la fecha de creación, por ejemplo:

Code highlighting:

```
/*
  Este archivo contiene la implementación de ...

  Autor(es): Alejandro J. Mujica, José Ruiz, Julie Vera

  Fecha de Creación: 27/05/2014
*/
```

Normas de Codificación

Las Variables

Asignarles nombres significativos a lo que almacenan y plantearlos como sustantivos.

No todo el tiempo una variable tiene un nombre significativo por sí mismo, en esos casos agregar un comentario a la derecha de la declaración que explique brevemente su significado, si el comentario es multilínea, agregarlo antes de la declaración de la variable.

Escribir los nombres en letras minúsculas y separar las palabras que lo conformen mediante el caracter ■underscore.

Inicializarlas al momento de declararse y si se coloca una explicación a su lado.

Usar una línea diferente para cada declaración.

Por ejemplo:

Code highlighting:

```
int age = 18;
int x = 0; // Componente x de un punto
/*
 Componente y de un punto
 Ejemplo multilínea
*/
int y = 0;
```

Las Funciones

Asignarles nombres significativos a la operación que efectúen y plantearlos como verbos.

Escribir un bloque de comentario multilínea que contenga una breve explicación de lo que hacen antes de su definición. Nota: Si es muy evidente la operación que ésta realice, se puede ignorar este paso.

Escribir los nombres en letras minúsculas y separar las palabras que lo conformen mediante el caracter ■underscore.

Para los parámetros de las funciones usar las mismas especificaciones de las variables.

Por ejemplo:

Code highlighting:

```
/*
 Efectúa la división de dos números reales (a / b), si b es igual a 0 arroja
 excepción de error de dominio.
*/
float divide_float(const float & a, const float & b)
{
 if (b == 0)
   throw std::domain_error("Division by 0");

 return a / b;
}
```

Clases

Escribir mediante un comentario multilínea una breve explicación de los que representa la clase así como el autor o autores de ésta.

Asignarles nombres significativos de lo que éstas representen y plantearlos como sustantivos.

Escribir los nombres con el estilo de nombres propios, es decir, la primera letra de cada palabra que lo conforme en mayúscula y el resto en minúscula; separar cada palabra que compone el nombre mediante el caracter ■underscore.

Ejemplo de definición de una clase:

Code highlighting:

```
/* Clase que representa una lista mediante arreglos

  Autor: Alejandro J. Mujica
*/
template <typename T>
class Array_List
{

};
```

A los atributos de una clase aplicarle las mismas especificaciones de una variable.

A los metodos de una clase aplicarle las mismas especificaciones que a las funciones.

Consideraciones especiales sobre métodos

Constructores

Asignarle valor inicial a todos los atributos de la clase, usar el espacio de inicialización del constructor para el valor inicial y el espacio de implementación para cualquier operación adicional sobre los atributos.

Siguiendo con el ejemplo de la clase definida anteriormente:

Code highlighting:

```
/* Plantilla que representa una lista mediante arreglos

  Recibe como parámetro plantilla el tipo de elemento que va a almacenar.

  Autor: Alejandro J. Mujica
*/
template <typename T>
class Array_List
{
 size_t size;  // Almacena el tamaño del arreglo
 T *    array; // Arreglo de elementos de tipo T
```

```
public:
 Array_List(const size_t & _size)
   : size(_size), array(nullptr)
 {
   array = new T[size];
 }
};
```

Modificadores y Observadores

A los observadores colocarles como nombre el mismo nombre del atributo al cual observa con el prefijo 'get_'.

A los modificadores colocarles como nombre el mismo nombre del atributo al cual observa con el prefijo 'set_'.

Por ejemplo para consultar el tamaño de la lista:

Code highlighting:

```
/* Clase que representa una lista mediante arreglos

  Autor: Alejandro J. Mujica
*/
template <typename T>
class Array_List
{

 // ... Otras implementaciones ...

 const size_t & get_size() const
 {
   return size;
 }
};
```

Interfaz de Usuario