Análisis de Algoritmos y Estructuras de Datos

Práctica 1: Programación en C++

Mª Teresa García Horcajadas José Fidel Argudo Argudo
Antonio García Domínguez Francisco Palomo Lozano

UCA
Universidad de Cádiz

Versión 1.0

Índice

1. Características
2. Un primer programa
3. Mejoras en el lenguaje
4. Entrada/salida
5. Cadenas
6. Clase cronómetro

Evolución

Historia

1. C++ fue diseñado por Bjarne Stroustrup
   - 1979 C con clases
   - 1983 C++
   - 1985 Primera versión comercial
2. Fue normalizado por ISO y continúa en evolución
   - 1998 ISO/IEC 14882:1998 (C++98)
   - 2003 ISO/IEC 14882:2003 (technical corrigendum)
   - 2011 ISO/IEC 14882:2011 (C++11)
3. Existen versiones libres y privativas

Generalidades

Datos

1. C++ es un lenguaje multiparadigma
   - Programación estructurada
   - Programación orientada a objetos
   - Programación genérica
2. Se trata de un lenguaje de nivel intermedio
   - Compromiso entre un lenguaje de bajo y de alto nivel
   - Extraordinariamente eficiente, si se usa correctamente
3. Se emplea intensivamente en programación de sistemas
   - Compiladores
   - Bases de datos
   - Sistemas operativos
   - Sistemas de defensa
   - Simuladores
   - Videojuegos

Hola a todos: el programa que saluda

Dos versiones

```
1 #include <iostream>

3 int main()
4 {
5   using namespace std;
6   cout << "¡Hola␣a␣todos!" << endl;
7 }
```

```
1 #include <iostream>

3 int main()
4 {
5   std::cout << "¡Hola␣a␣todos!" << std::endl;
6 }
```

Constantes

Características

1. Las constantes poseen enlace interno
   - Pueden aparecer en cabeceras
   - Se deben usar preferentemente a las macros

```
1 const int N = 100;
2 const int T = N * 1024 + 128;
3 const char* empleo = "Agítese␣antes␣de␣usar.";
```

2. Las definiciones de constantes deben inicializarse
   - Se inicializan mediante una expresión constante
   - Su valor puede ser desconocido durante la compilación

```
4 #include <cmath>

6 const double e = std::exp(1.0);
7 const double pi = 4.0 * std::atan(1.0);
```

Funciones en línea

Características

1. Pueden aparecer en cabeceras
2. Se deben usar preferentemente a las macros
3. El compilador podrá expandir su código en lugar de llamarlas
   - No deben contener llamadas recursivas
   - Deben consistir en fragmentos pequeños de código

```
1 #include <cmath>

3 inline double cuadrado(double x) { return x * x; }

5 inline double
6 distancia(double a, double b, double x, double y)
7 {
8   return std::sqrt(cuadrado(x - a) + cuadrado(y - b));
9 }
```

Declaraciones

Características

- Pueden aparecer casi en cualquier lugar

```
1 for (int k = 0; k < n; ++k)
2   for (int i = 0; i < n; ++i)
3     for (int j = 0; j < n; ++j) {
4       double d = p[i][k] + p[k][j];
5       if (d < p[i][j])
6         p[i][j] = d;
7     }
```

- Los nombres de enumeraciones, estructuras, ... definen un tipo

```
1 struct fecha { unsigned short d, m, a; };
2 fecha nacimiento, compra;
```

Referencias

Concepto

1. Una referencia es un identificador de otro objeto
   - Externamente se usa como una variable ordinaria
   - Internamente se comporta como un puntero
   - Su memoria coincide con la del objeto al que se refiere
2. Su uso principal es en el paso por referencia

```
1 void actualiza(double& e, double& v, double a, double t)
2 {
3   e += v * t + 0.5 * a * t * t;
4   v += a * t;
5 }

  ...

6 double e = 20.0, v = 0.0;
7 actualiza(e, v, 9.8, 1.0); // Modifica e y v.
```

Sobrecarga

Concepto

- Un mismo nombre de función puede significar cosas distintas

```
1 void actualiza(double& e, double v, double t)
2 {
3   e += v * t;
4 }
```

- Las funciones se diferencian por el n.º o tipo de sus parámetros

```
1 double e = 20.0, v = 0.0;
2 actualiza(e, v, 9.8, 1.0); // Con aceleración.
3 actualiza(e, v, 1.5); // Sin aceleración.
```

- A esto se le llama sobrecarga y es un tipo de polimorfismo

Clases

Concepto

- Una clase es una abstracción conjunta de datos y operaciones
- Aparecen en C++ como generalización de las estructuras
- Pueden contener miembros de datos y funciones miembro
- Los miembros pueden ser:

  Públicos Visibles desde el exterior

  Privados Ocultos al exterior

Ejemplo

- Clase que representa un objeto móvil

```
1  class movil {
2  public:
3    movil(double m);
4    double espacio() const;
5    double tiempo() const;
6    double velocidad() const;
7    double aceleracion() const;
8    void aplicaFuerza(double f, double dt);
9  private:
10   double m, e, t, v, a;
11 };
```

- Empleo

```
1 movil proyectil(m);
2 proyectil.aplicaFuerza(f, t);
```

Entrada/Salida

Stream o flujo de datos

- Es una abstracción de C++ para realizar operaciones de E/S, que aisla al programador de las complicaciones de controlar directamente los periféricos.
- Es una interfaz entre el programador y el S.O., proporcionada por la biblioteca estándar de E/S de C++.
- Se comporta como una corriente de bytes que actúa como fuente o destino de datos según sea de entrada o de salida.
- Es un objeto (asociado a un dispositivo como la pantalla, el teclado, un fichero en disco, etc.) donde un programa puede insertar, o desde el que puede extraer, datos.

Entrada/Salida

Conceptos

iostream Cabecera estándar en la que están definidos los objetos stream de entrada y salida.

cout Objeto definido en `iostream` para escribir caracteres en la salida estándar, que suele ser la pantalla.

cin Objeto definido en `iostream` para leer caracteres de la entrada estándar, que suele ser el teclado.

<< Operador de inserción en flujo de salida.

>> Operador de extracción de flujo de entrada.

endl Escribe un salto de línea y vacía el bufer de salida. En la pantalla equivale a un carácter `'\n'`.

Ejemplo

```
1  #include <iostream>
2  using namespace std;

4  int main()
5  {
6    int n = 0;
7    double s = 0.0;
8    cout << "Datos␣numéricos:\n" << endl;
9    while (cin) {
10     cout << 'x' << n << "␣=␣";
11     double x;
12     if (cin >> x) {
13       ++n; s += x;
14     }
15   }
16   cout << "\n\nLa␣media␣aritmética␣es␣" << (n ? s / n : 0) << '.'
17        << endl;
18   return 0;
19 }
```

Cadenas

Ejemplos básicos

```
1  #include <string>

3  using std::string;

5  string completo;
6  string nombre = "Pepito", apellido = "Pérez";
7  apellido += " García";
8  completo = apellido + ", " + nombre;

10 for (string::size_type i = 0; i < nombre.size(); ++i)
11   nombre[i] = std::toupper(nombre[i]);
```

Más cadenas

Ejemplo con `getline()`

```
1  #include <iostream>
2  #include <string>

4  int main()
5  {
6    using namespace std;
7    cout << "Nombre␣completo:␣";
8    string nombre;
9    getline(cin, nombre);
10   cout << "Hola,␣" << nombre << '.' << endl;
11 }
```

Clase cronómetro

Miembros públicos

- `cronometro();`
  Construye un cronómetro inactivo.
- `void activar();`
  Pone en marcha el cronómetro.
- `void parar();`
  Para el cronómetro.
- `double tiempo() const;`
  Si el cronómetro está activo, devuelve los segundos transcurridos desde que se puso en marcha; en caso contrario, devuelve los segundos entre su último inicio y su parada.

Ejemplo

```
1 cronometro c;
2 c.activar();
3 for (long i = 0; i < 1000000000L; ++i)
4   ;
5 c.parar();
6 cout << c.tiempo() << "␣s" << endl;
```