Definición y Conversión de datos

Agustín J. González

ELO-329

Calificador Const

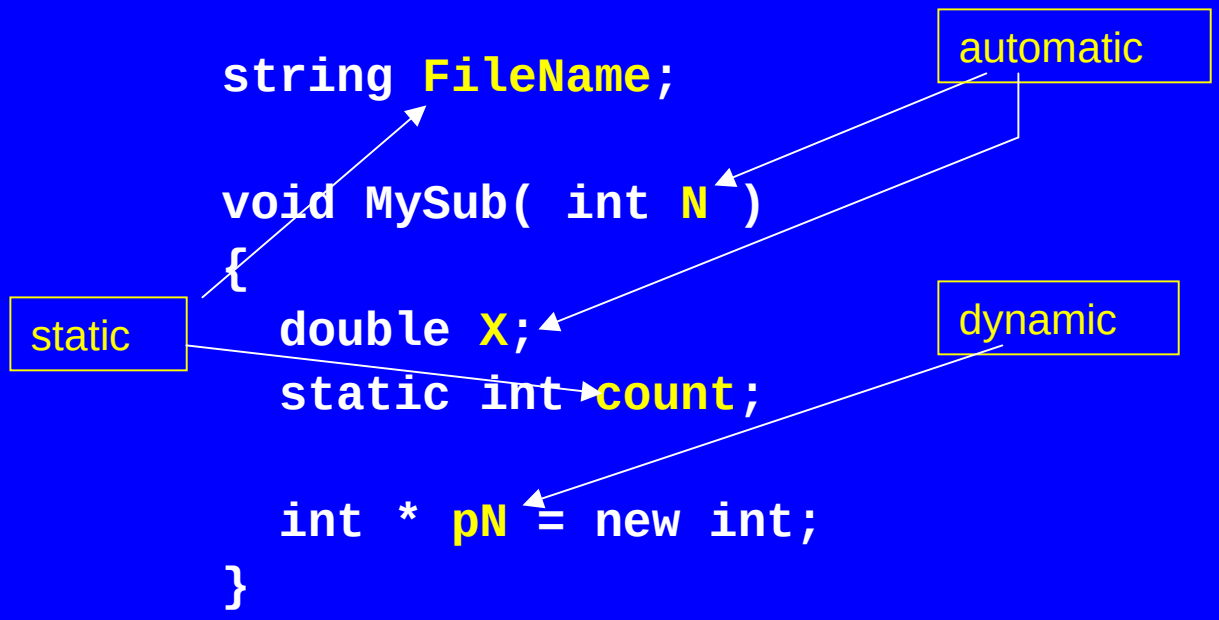
- El calificador const previene que un objeto sea modificado con posterioridad a su definición.
- El objeto calificado como constante debe tener un valor asignado en su definición.
- Ojo Definición versus declaración: en la definición la variable se solicita almacenamiento, en la declaración sólo se introduce un identificador y se indican sus atributos (tipos, parámetros etc).
- Un objeto calificado como constante puede ser asignado a uno no constante. Es mejor que usar #define ....

```
const int n = 25;
n = 36;                 // error
const double z;         // error
int m = n;
m = 36;
```

Asignación de almacenamiento

- Variables con almacenamiento static (estático) existen durante todo el programa.
  - Variables Global
  - Variables declaradas con el calicador static
- Variables con almacenamiento automatic (automático) son creadas en el stack cuando son definidas dentro de un bloque.
  - Variables/Objetos locales de funciones
  - Parámetros de Funciones
- Variables con almacenamiento dynamic (Dinámico) se crean en el heap, y pueden ser creadas y removidas en tiempo de ejecución (operadores *new* y *delete*).

Ejemplos de asignación de almacenamiento



Alcance de Variables

- Variables globales existen fuera de cualquier bloque de código.
  - un bloque queda descrito por los símbolos { ... }
- Variables locales existen dentro del bloque de código.
  - AL interior de una función, por ejemplo.
- Variables miembros existen dentro del bloque de definición de una clase
  - class *name* { ... }

Referencias

- Una referencia es un *alias* para algún objeto existente.
- Físicamente, la referencia almacena la dirección del objeto que referencia.
- EN el ejemplo, cuando asignamos un valor a rN, también estamos modificando N:

```
int N = 25;
int & rN = N;
rN = 36;
cout << N;          // "36" displayed
```

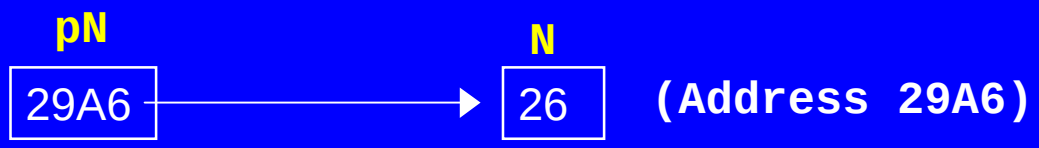
Punteros

- Un puntero almacena la dirección de algún objeto.
- El operador *dirección-de* (&) obtiene la dirección de un objeto.

```
int N = 26;

int * pN = &N;    // get the address
```

Implementación de un puntero



pN apunta a N porque pN contiene la dirección de N

Variables Punteros

Un puntero debe ser definido con el mismo tipo al que éste apunta.

pN no puede apuntar a Z porque Z es un double:

```
double Z = 26;

int * pN;
pN = &Z;                    // error!
```

El formato interno y tamaño de un puntero no es el mismo que el de un entero!

Operador “desreferencia” (Dereference Operator)

El operador * obtiene el contenido de la variable que es referenciada por un puntero. Obtiene el contenido de.

```
int N = 26;
int * pN = &N;

cout << *pN << endl;        // "26"
```

Sólo punteros pueden ser desrefenciados.

El operador * (cont)

El operador * también puede ser usado para modificar el contenido de la variable (u objeto) referenciada.

```
int N = 26;
int * pN = &N;

*pN = 35;
cout << *pN << endl;   // "35"
cout << N << endl;     // "35"
```

AL asignación de un valor a *pN cambia el valor de N.

Asignación de punteros

- Un puntero puede ser asignado a otro siempre y cuando apunten al mismo tipo de variable.
- Ejemplo, cuando pZ es desreferenciado, nos permite cambiar el valor de N:

```
int N = 26;
int * pN = &N;
int * pZ;

pZ = pN;
*pZ = 35;              // now N = 35
```

Asignación de punteros

La asignación de un objeto a otro puede ser hecha desreferenciando punteros.

```
int N = 26;
int * pN = &N;
int Z = 0;
int * pZ = &Z;

*pZ = *pN;              // Z = 26
```

Conversiones de datos

Conversiones Implícitas

- *Conversiones Implícitas* pueden tener lugar cuando un objeto de un tipo es asignado a un objeto de otro tipo.
- C++ maneja conversiones automáticamente en el caso de tipos numéricos intrínsecos (int, double, float)
- Mensajes de advertencia (warning) pueden aparecer cuando hay riesgo de perdida de información (precisión).
  - Hay variaciones de un compilador a otro

Ejemplos...

Ejemplos de Conversión

```
int n = 26;
double x = n;
double x = 36;
int b = x;          // possible warning
float f = 36.5;     // possible warning
bool isOK = 1;      // possible warning
int n = true;
char ch = 26;       // possible warning
int w = 'A';
```

Operación Cast

- Una operación de “casteo” *cast* explícitamente convierte datos de un tipo a otro.
- Es usado en conversiones “seguras” que podrían ser hechas por el compilador.
- Son usadas para abolir mensajes de advertencia (warning messages).
- El operador tradicional del C pone el nuevo tipo de dato entre paréntesis. C++ mejora esto con una operador cast tipo función.

Ejemplos...

Ejemplos de Cast

```
int n = (int)3.5; // traditional C
int w = int(3.5); // estilo de función
bool isOK = bool(15);
char ch = char(86);  // símbolo ascii
string st = string("123");

// errors: no conversions available:
int x = int("123");
string s = string(3.5);

double x=3.1415
char *p = (char*)&x;  // para acceder a
                      //x byte por byte
```

static_cast<>

- El operador static_cast<> es la forma preferida para hacer conversiones “seguras” en C++.
- Éste reemplaza ambos el operador tradicional de C y el estilo función de C++.
- Fue introducido recientemente.
- Se recomienda usar este estilo de cast.

Ejemplos...

Ejemplos de static_cast

```
int w = static_cast<int>(3.5);

bool isOK = static_cast<bool>(1);

char ch = static_cast<char>(86);
```

Fin