UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Definición y Conversión de datos

Agustín J. González

ELO-329

Calificador Const

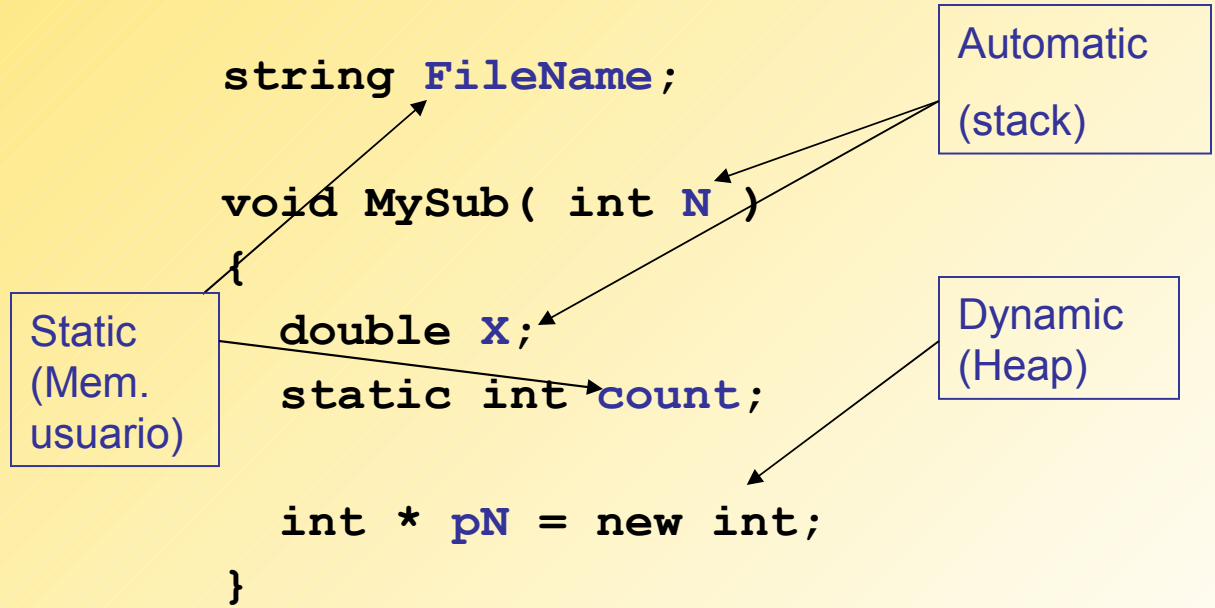
- El calificador const previene que un objeto sea modificado con posterioridad a su definición.
- El objeto calificado como constante debe tener un valor asignado en su definición.
- Ojo Definición versus declaración: en la definición la variable se solicita almacenamiento, en la declaración sólo se introduce un identificador y se indican sus atributos (tipos, parámetros etc).
- 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 calificador static
- Variables con almacenamiento automatic (automático) son almacenadas en el stack cuando son definidas dentro de un bloque.
  - Variables/Objetos locales de funciones
  - Parámetros de Funciones
- Variables con almacenamiento dynamic (Dinámico) son ubicadas 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

- Es posible definir variables con visibilidad sólo dentro de un bloque. un bloque queda descrito por los símbolos
  { ... }

```
:
{ int i =20;
   a+=i;
}
:
```

- Variables locales existen dentro del bloque de código.
  - Al interior de una función, por ejemplo.
- Atributos de una clase 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;  // referencia a N
    /* similar a puntero en semántica,
       pero con sintaxis normal*/
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 o variable escalar.

```
int N = 26;

int * pN = &N;     // get the address
/* la idea es similar al anterior,
pero debemos usar otra sintaxis para
acceder a los datos */
```

Variables Punteros

- double Z = 26;

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

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

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

Operador “des-referencia” (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 des-refenciados.

Notar:

- int *p1, *p2; // para definir dos punteros

Asignación de punteros

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

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

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

Punteros a Arreglos

- Un arreglo es un puntero constante. Esto permite hacer:
  int a [5];
  int *p;
  p=a;
- Luego es posible acceder a los elementos del arreglo:
  p[0], p[1] ..., p[4]
- Otras posibilidades son:
  int i=*(p+3); // lo cual es igual a
  int i=p[3];
- También es válido:
  int * p2=p+3;
  int b= p2[-1];
- Ver más en ejemplo pointerArray.cpp

Conversiones Implícitas datos

- 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
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.

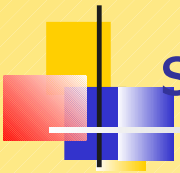
- Ejemplos...

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++.
- Es de responsabilidad del programador asegurar la compatibilidad de los datos involucrados.
- Éste reemplaza ambos el operador tradicional de C y el estilo función de C++.
- Se recomienda usar este estilo de cast.
- Existe también el dynamic_cast<>, éste asegura que el resultado de la conversión es un dato compatible. Sólo se aplica a punteros a objetos.
- dynamic_cast<> lanza error cuando el resultado de la conversión no es un objeto completo de la clase requerida.
- Ejemplos...

Ejemplos de static_cast

```
int w = static_cast<int>(3.5);
bool isOK = static_cast<bool>(1);
char ch = static_cast<char>(86);

class CBase { };
class CDerived: public CBase { };
CBase b; CBase* pb;
CDerived d; CDerived* pd;
pb = dynamic_cast<CBase*>(&d);
       // ok: derived-to-base
pd = dynamic_cast<CDerived*>(&b);
      // wrong: base-to-derived
```