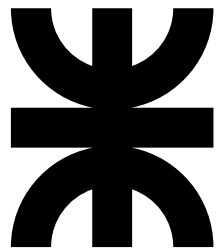
Informática II

Clases de almacenamiento, reglas de alcance, y calificadores de variables

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2018 –

Clases de almacenamiento

Introducción

- Se usan identificadores para los nombres de las variables

Clases de almacenamiento

Introducción

- Se usan identificadores para los nombres de las variables
- Los atributos de una variable son: tipo, nombre, y valor

Clases de almacenamiento

Introducción

- Se usan identificadores para los nombres de las variables
- Los atributos de una variable son: tipo, nombre, y valor

Cada identificador (también funciones) tiene otros atributos

1. Clase de almacenamiento: define a los siguientes atributos

Clases de almacenamiento

Introducción

- Se usan identificadores para los nombres de las variables
- Los atributos de una variable son: tipo, nombre, y valor

Cada identificador (también funciones) tiene otros atributos

1. Clase de almacenamiento: define a los siguientes atributos
2. Duración de almacenamiento: período durante el cual el identificador existe en memoria

Clases de almacenamiento

Introducción

- Se usan identificadores para los nombres de las variables
- Los atributos de una variable son: tipo, nombre, y valor

Cada identificador (también funciones) tiene otros atributos

1. Clase de almacenamiento: define a los siguientes atributos
2. Duración de almacenamiento: período durante el cual el identificador existe en memoria
3. Alcance: desde donde se puede referenciar al identificador (reglas de alcance)

Clases de almacenamiento

Introducción

- Se usan identificadores para los nombres de las variables
- Los atributos de una variable son: tipo, nombre, y valor

Cada identificador (también funciones) tiene otros atributos

1. Clase de almacenamiento: define a los siguientes atributos
2. Duración de almacenamiento: período durante el cual el identificador existe en memoria
3. Alcance: desde donde se puede referenciar al identificador (reglas de alcance)
4. Enlace: para programas de varios archivos fuentes

Clases de almacenamiento

Introducción

- Se usan identificadores para los nombres de las variables
- Los atributos de una variable son: tipo, nombre, y valor

Cada identificador (también funciones) tiene otros atributos

1. Clase de almacenamiento: define a los siguientes atributos
2. Duración de almacenamiento: período durante el cual el identificador existe en memoria
3. Alcance: desde donde se puede referenciar al identificador (reglas de alcance)
4. Enlace: para programas de varios archivos fuentes

C cuenta con 4 clases de almacenamiento

- `auto`
- `register`
- `static`
- `extern`

Clases de almacenamiento

Persistencia

Clases de almacenamiento

Persistencia

Persistencia automática: se crean cuando se entra en el ámbito de un bloque, existen solo en dicho bloque y se destruyen cuando se sale.
(solo para variables)

Clases de almacenamiento

Persistencia

Persistencia automática: se crean cuando se entra en el ámbito de un bloque, existen solo en dicho bloque y se destruyen cuando se sale.
(solo para variables)

Las palabras reservadas `auto` y `register` se usan en variables de persistencia automática.

Clases de almacenamiento

Persistencia

Persistencia automática: se crean cuando se entra en el ámbito de un bloque, existen solo en dicho bloque y se destruyen cuando se sale.
(solo para variables)

Las palabras reservadas `auto` y `register` se usan en variables de persistencia automática.

Persistencia estática: existen a partir de que el programa inicia su ejecución. Esto no significa que puedan ser utilizados (alcance).
(también para funciones)

Clases de almacenamiento

Persistencia

Persistencia automática: se crean cuando se entra en el ámbito de un bloque, existen solo en dicho bloque y se destruyen cuando se sale.
(solo para variables)

Las palabras reservadas `auto` y `register` se usan en variables de persistencia automática.

Persistencia estática: existen a partir de que el programa inicia su ejecución. Esto no significa que puedan ser utilizados (alcance).
(también para funciones)

Las palabras reservadas `static` y `extern` se usan en variables de persistencia estática.

Clases de almacenamiento

Persistencia automática

- `auto`: Variables locales de una función, ya sean declaradas en la lista de parámetros o en el cuerpo de la función.

```
auto float x, y;
```

  Las variables locales tienen persistencia automática por defecto.

Clases de almacenamiento

Persistencia automática

- `auto`: Variables locales de una función, ya sean declaradas en la lista de parámetros o en el cuerpo de la función.

```
auto float x, y;
```

  Las variables locales tienen persistencia automática por defecto.

- `register`: Le sugiere al compilador que la variable se guarde en los registros del microprocesador.

```
register int contador = 0;
```

  La palabra reservada **`register`** se puede usar solo en variables automáticas. (el compilador puede ignorar la sugerencia)

Clases de almacenamiento

Persistencia automática

- `auto`: Variables locales de una función, ya sean declaradas en la lista de parámetros o en el cuerpo de la función.

```
auto float x, y;
```

Las variables locales tienen persistencia automática por defecto.

- `register`: Le sugiere al compilador que la variable se guarde en los registros del microprocesador.

```
register int contador = 0;
```

La palabra reservada **`register`** se puede usar solo en variables automáticas. (el compilador puede ignorar la sugerencia)

Solo las variables pueden tener persistencia automática.

Clases de almacenamiento

Persistencia estática

- `static`: Variables locales declaradas con el especificador de clase de almacenamiento `static`. Conocidas solo en la función donde son definidas, pero a diferencia de las variables automáticas, estas conservan su valor cuando se sale de la función.

```
static int contador = 1;
```

  Se inicializan a cero si no son inicializadas de forma explícita (o `NULL` para punteros).

Clases de almacenamiento

Persistencia estática

- `static`: Variables locales declaradas con el especificador de clase de almacenamiento `static`. Conocidas solo en la función donde son definidas, pero a diferencia de las variables automáticas, estas conservan su valor cuando se sale de la función.

```
static int contador = 1;
```

  Se inicializan a cero si no son inicializadas de forma explícita (o `NULL` para punteros).

- `extern`: Se utiliza para programas de varios archivos fuentes. Por defecto, las variables globales y los nombres de funciones son de la clase de almacenamiento `extern`.

Clases de almacenamiento

Más sobre `extern`

- Una variable es *externa* si se encuentra fuera de cualquier función

Clases de almacenamiento

Más sobre `extern`

- Una variable es *externa* si se encuentra fuera de cualquier función
- *externo* en contraste a *interno* –en funciones–

Clases de almacenamiento

Más sobre `extern`

- Una variable es *externa* si se encuentra fuera de cualquier función
- *externo* en contraste a *interno* –en funciones–

Para el caso de variables `extern`:

Declaración: expone las propiedades de una variable (principalmente su tipo)

Definición: provoca que se reserve espacio para el almacenamiento

Alcance de variables

Introducción

Alcance de un identificador: porción del programa donde puede ser referenciado.

Alcance de variables

Introducción

Alcance de un identificador: porción del programa donde puede ser referenciado.

Ejemplo: variable local declarada en un bloque –puede ser referenciada en el bloque o bloques anidados–

Alcance de variables

Introducción

Alcance de un identificador: porción del programa donde puede ser referenciado.

Ejemplo: variable local declarada en un bloque –puede ser referenciada en el bloque o bloques anidados–

Tipos de alcances

1. Alcance de archivo: identificador declarado fuera de cualquier función

Alcance de variables

Introducción

Alcance de un identificador: porción del programa donde puede ser referenciado.

Ejemplo: variable local declarada en un bloque –puede ser referenciada en el bloque o bloques anidados–

Tipos de alcances

1. Alcance de archivo: identificador declarado fuera de cualquier función
2. Alcance de función: etiquetas (identificador seguido de `:`). p.e.: `start:`
   Etiquetas `case` en estructura `switch`

Alcance de variables

Introducción

Alcance de un identificador: porción del programa donde puede ser referenciado.

Ejemplo: variable local declarada en un bloque –puede ser referenciada en el bloque o bloques anidados–

Tipos de alcances

1. Alcance de archivo: identificador declarado fuera de cualquier función
2. Alcance de función: etiquetas (identificador seguido de `:`). p.e.: `start:`
   Etiquetas `case` en estructura `switch`
3. Alcance de bloque: identificadores declarados dentro de un bloque
   Las variables locales `static` declaradas dentro de funciones tienen alcance de bloques aún cuando existen al momento de ejecutarse el programa

Alcance de variables

Introducción

Alcance de un identificador: porción del programa donde puede ser referenciado.

Ejemplo: variable local declarada en un bloque –puede ser referenciada en el bloque o bloques anidados–

Tipos de alcances

1. Alcance de archivo: identificador declarado fuera de cualquier función
2. Alcance de función: etiquetas (identificador seguido de `:`). p.e.: `start:`
   Etiquetas `case` en estructura `switch`
3. Alcance de bloque: identificadores declarados dentro de un bloque
   Las variables locales `static` declaradas dentro de funciones tienen alcance de bloques aún cuando existen al momento de ejecutarse el programa
4. Alcance de prototipo de función: lista de parámetros en los prototipos de funciones

Alcance de variables

Ejemplos

El alcance de una variable o función *externa* va desde el punto en el que se declara hasta el fin del archivo.

Alcance de variables

Ejemplos

El alcance de una variable o función *externa* va desde el punto en el que se declara hasta el fin del archivo.

```
int main(void) { . . . }

int sp = 0;
double val[MAXVAL];

void push(double f) { . . . }
double pop(void) { . . .}
```

Alcance de variables
Ejemplos

El alcance de una variable o función *externa* va desde el punto en el que se declara hasta el fin del archivo.

```
int main(void) { . . . }

int sp = 0;
double val[MAXVAL];

void push(double f) { . . . }
double pop(void) { . . .}
```

- Las variables **sp** y **val** se pueden utilizar en **push** y **pop**, pero no en **main**.
- Si se hace referencia a una variable *externa* antes de su definición, o si está definida en un archivo fuente diferente, es obligatorio una declaración **extern**.

Alcance de variables

Ejemplos

Si las líneas

```
int sp;
double val[MAXVAL];
```

aparecen fuera de cualquier función, *definen* las variables externas `sp` y `val`, reservan un espacio para almacenamiento.

Alcance de variables
Ejemplos

Si las líneas

```
int sp;
double val[MAXVAL];
```

aparecen fuera de cualquier función, *definen* las variables externas **sp** y **val**, reservan un espacio para almacenamiento.

Las líneas

```
extern int sp;
extern double val[];
```

declaran para el resto del archivo que **sp** es un **int** y que **val** es un arreglo **double** (cuyo tamaño se determina en algún otro lugar), pero no crea las variables ni les reserva espacio.

Alcance de variables
Ejemplos

Si las líneas

```
int sp;
double val[MAXVAL];
```

aparecen fuera de cualquier función, *definen* las variables externas **sp** y **val**, reservan un espacio para almacenamiento.

Las líneas

```
extern int sp;
extern double val[];
```

declaran para el resto del archivo que **sp** es un **int** y que **val** es un arreglo **double** (cuyo tamaño se determina en algún otro lugar), pero no crea las variables ni les reserva espacio.

Debe existir una única *definición* de una variable externa entre todos los archivos fuentes que forman un programa. Los demás archivos pueden hacer *declaraciones* **extern** de estas variables para tener acceso a ellas.

Alcance de variables

Ejemplos

Archivo 1

```
int sp = 0;
double val[MAXVAL];
```

Archivo 2

```
extern int sp;
extern double val;

void push(double f) { . . . }

double pop(void) { . . . }
```

Alcance de variables

Ejemplos

Archivo 1

```
int sp = 0;
double val[MAXVAL];
```

Archivo 2

```
extern int sp;
extern double val;

void push(double f) { . . . }

double pop(void) { . . . }
```

Ver ejemplo de *alcance* D&D 5.12.

Calificadores

`volatile` y `const`

`volatile`

- Le indica al compilador no optimizar lo relacionado a dichas variables
- Utilizado principalmente para el acceso al hardware
- Le indica que la variable no se guarde en cache

Calificadores

`volatile` y `const`

`volatile`

- Le indica al compilador no optimizar lo relacionado a dichas variables
- Utilizado principalmente para el acceso al hardware
- Le indica que la variable no se guarde en cache

Ejemplo:

```
quit = 0;
while(!quit)
{
  /* bucle corto completamente
   * visible al compilador */
}
```

Calificadores

`volatile` y `const`

`const`

- Le informa al compilador que el valor de una variable no debe modificarse

Calificadores

`volatile` y `const`

`const`

- Le informa al compilador que el valor de una variable no debe modificarse
- No existía en la primeras versiones de C. Fue agregado en el ANSI C

Calificadores

`volatile` y `const`

`const`

- Le informa al compilador que el valor de una variable no debe modificarse
- No existía en la primeras versiones de C. Fue agregado en el ANSI C
- Seis posibilidades del uso y no uso de **`const`** con parámetros de función

Calificadores

`volatile` y `const`

`const`

- Le informa al compilador que el valor de una variable no debe modificarse
- No existía en la primeras versiones de C. Fue agregado en el ANSI C
- Seis posibilidades del uso y no uso de **`const`** con parámetros de función
  - dos al pasar parámetros en llamada por valor, y

Calificadores

`volatile` y `const`

`const`

- Le informa al compilador que el valor de una variable no debe modificarse
- No existía en la primeras versiones de C. Fue agregado en el ANSI C
- Seis posibilidades del uso y no uso de `const` con parámetros de función
  - dos al pasar parámetros en llamada por valor, y
  - cuatro al pasar parámetros en llamada por referencia (punteros).

Calificadores

`volatile` y `const`

`const`

- Le informa al compilador que el valor de una variable no debe modificarse
- No existía en la primeras versiones de C. Fue agregado en el ANSI C
- Seis posibilidades del uso y no uso de **`const`** con parámetros de función
  - dos al pasar parámetros en llamada por valor, y
  - cuatro al pasar parámetros en llamada por referencia (punteros).

Ejemplo:

```
void imprimir_arreglo(const int datos[], const int tam)
{
. . .
}
```

Calificadores

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

Calificadores

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

1. Puntero no constante a datos no constantes, p.e.: `int *pi;` (notación) (no incluye `const`)

Calificadores

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

1. Puntero no constante a datos no constantes, p.e.: `int *pi;` (notación) (no incluye `const`)

2. Puntero no constante a datos constantes

Calificadores

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

1. Puntero no constante a datos no constantes, p.e.: `int *pi;` (notación) (no incluye `const`)

2. Puntero no constante a datos constantes

3. Puntero constante a datos no constantes. Inicializados al declaralos (nombre de arreglo)

Calificadores

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

1. Puntero no constante a datos no constantes, p.e.: `int *pi;` (notación) (no incluye `const`)

2. Puntero no constante a datos constantes

3. Puntero constante a datos no constantes. Inicializados al declaralos (nombre de arreglo)

4. Puntero constante a datos constantes

Calificadores

Calificador `const` con punteros –Ejemplos

Puntero a entero constante

```
int * const foo;
```

Calificadores

Calificador `const` con punteros –Ejemplos

Puntero a entero constante

```
int * const foo;
```

Puntero constante a entero

```
int const * foo;
const int * foo;
```

Calificadores

Calificador `const` con punteros –Ejemplos

Puntero a entero constante

```
int * const foo;
```

Puntero constante a entero

```
int const * foo;
const int * foo;
```

Puntero constante a un entero constante

```
int const * const foo;
const int * const foo;
```