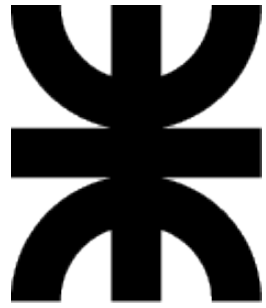
Informática II

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

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

1. ¿Qué valor se imprime en `main`?

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

1. ¿Qué valor se imprime en `main`?
2. ¿Qué valor se imprime en `foo`?

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

1. ¿Qué valor se imprime en `main`?
2. ¿Qué valor se imprime en `foo`?
3. ¿Qué representan las llaves dónde está definida la variable `x=5`?

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

1. ¿Qué valor se imprime en `main`?
2. ¿Qué valor se imprime en `foo`?
3. ¿Qué representan las llaves dónde está definida la variable `x=5`?
4. ¿Qué tipo de variable es la `x` definida fuera de `main` y `foo`?

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

1. ¿Qué valor se imprime en `main`?
2. ¿Qué valor se imprime en `foo`?
3. ¿Qué representan las llaves dónde está definida la variable `x=5`?
4. ¿Qué tipo de variable es la `x` definida fuera de `main` y `foo`?
5. ¿Qué tipo de variable es la `x` definida en `main`?

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

1. ¿Qué valor se imprime en `main`?
2. ¿Qué valor se imprime en `foo`?
3. ¿Qué representan las llaves dónde está definida la variable `x=5`?
4. ¿Qué tipo de variable es la `x` definida fuera de `main` y `foo`?
5. ¿Qué tipo de variable es la `x` definida en `main`? Y en `foo`?

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

1. ¿Qué valor se imprime en `main`?
2. ¿Qué valor se imprime en `foo`?
3. ¿Qué representan las llaves dónde está definida la variable `x=5`?
4. ¿Qué tipo de variable es la `x` definida fuera de `main` y `foo`?
5. ¿Qué tipo de variable es la `x` definida en `main`? Y en `foo`?
6. ¿Durante cuanto tiempo de la ejecución del programa existe cada variable `x`?

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

Alcance vs. percistencia

1. ¿Qué valor se imprime en `main`?
2. ¿Qué valor se imprime en `foo`?
3. ¿Qué representan las llaves dónde está definida la variable `x=5`?
4. ¿Qué tipo de variable es la `x` definida fuera de `main` y `foo`?
5. ¿Qué tipo de variable es la `x` definida en `main`? Y en `foo`?
6. ¿Durante cuanto tiempo de la ejecución del programa existe cada variable `x`?

Introducción

```c
#include <stdio.h>

int x = 0;

int main(void) {
  int x = 2;
  {
    int x = 5;
  }
  printf("x = %d\n", x);
  foo();
  return x;
}

void foo(void) {
  int x = 10;
  printf("x = %d\n", x);
}
```

1. ¿Qué valor se imprime en `main`?
2. ¿Qué valor se imprime en `foo`?
3. ¿Qué representan las llaves dónde está definida la variable `x=5`?
4. ¿Qué tipo de variable es la `x` definida fuera de `main` y `foo`?
5. ¿Qué tipo de variable es la `x` definida en `main`? Y en `foo`?
6. ¿Durante cuanto tiempo de la ejecución del programa existe cada variable `x`?

Alcance vs. percistencia

Ver ejemplo D&D 5.12
(fuente `fig05_12a.c` sin comentarios)

Clases de almacenamiento

Cada identificador tiene otro atributo llamados clase de almacenamiento, el cual define:

Clases de almacenamiento

Cada identificador tiene otro atributo llamados clase de almacenamiento, el cual define:

1. Persistencia o duración de almacenamiento: período durante el cual el identificador existe en memoria

Clases de almacenamiento

Cada identificador tiene otro atributo llamados clase de almacenamiento, el cual define:

1. Persistencia o duración de almacenamiento: período durante el cual el identificador existe en memoria
2. Alcance: dónde se puede referenciar al identificador (reglas de alcance)

Clases de almacenamiento

Cada identificador tiene otro atributo llamados clase de almacenamiento, el cual define:

1. Persistencia o duración de almacenamiento: período durante el cual el identificador existe en memoria
2. Alcance: dónde se puede referenciar al identificador (reglas de alcance)
3. Enlace/vinculación: para programas de varios archivos fuentes

Clases de almacenamiento

Cada identificador tiene otro atributo llamados clase de almacenamiento, el cual define:

1. Persistencia o duración de almacenamiento: período durante el cual el identificador existe en memoria
2. Alcance: dónde se puede referenciar al identificador (reglas de alcance)
3. Enlace/vinculación: para programas de varios archivos fuentes

C cuenta con 4 clases de almacenamiento

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

Repaso de identificadores

- Identificadores: para nombres de variables (y funciones)

Repaso de identificadores

- Identificadores: para nombres de variables (y funciones)
- Atributos de una variable: nombre, tipo (tamaño) y valor

Repaso de identificadores

- Identificadores: para nombres de variables (y funciones)
- Atributos de una variable: nombre, tipo (tamaño) y valor

Reglas para identificadores:

- Serie de caracteres que constan letras, dígitos y guiones bajos (_).
- No pueden comenzar con un dígito, sí con guíon bajo.
- Puede tener cualquier longitud (recomendado no más de 31 caracteres).
- Diferencia entre minúsculas y mayúsculas (case sensitive).
- Se utiliza también para definir constantes simbólicas y macros.
- No se pueden utilizar las palabras reservadas como identificador.

Persistencia (duración)

Persistencia automática vs. persistencia estática

Persistencia (duración)

Persistencia automática vs. persistencia estática

1. Persistencia automática
   - se crean cuando se entra en el ámbito de un bloque,

Persistencia (duración)

Persistencia automática vs. persistencia estática

1. Persistencia automática
   - se crean cuando se entra en el ámbito de un bloque,
   - existen solo en dicho bloque y

Persistencia (duración)

Persistencia automática vs. persistencia estática

1. 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)

Persistencia (duración)

Persistencia automática vs. persistencia estática

1. 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 (duración)

Persistencia automática vs. persistencia estática

1. 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.

2. Persistencia estática
   - existen a partir de que el programa inicia su ejecución.

Persistencia (duración)

Persistencia automática vs. persistencia estática

1. 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.

2. Persistencia estática
   - existen a partir de que el programa inicia su ejecución.
   - Esto no significa que puedan ser utilizados (alcance).

   (también para funciones)

Persistencia (duración)

Persistencia automática vs. persistencia estática

1. 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.

2. 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.

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.

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)

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).

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`.

Alcance de variables

Alcance de un identificador

Porción del programa donde puede ser referenciado.

Alcance de variables

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

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

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 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

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 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

---

3. Alcance de función: etiquetas (identificador seguido de `:`). p.e.: `start:`
   Etiquetas `case` en estructura `switch`

Alcance de variables

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 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

---

3. Alcance de función: etiquetas (identificador seguido de `:`). p.e.: `start:`
   Etiquetas `case` en estructura `switch`
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 indice = 0;
double vector[MAXVAL];

void f1(double f) { . . . }
double f2(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 indice = 0;
double vector[MAXVAL];

void f1(double f) { . . . }
double f2(void) { . . .}
```

- Las variables `indice` y `vector` se pueden utilizar en `f1()` y `f2()`, 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 indice;
double vector[MAXVAL];
```

aparecen fuera de cualquier función, *definen* las variables externas `indice` y `vector`, y reserva espacio para su almacenamiento.

Alcance de variables – Ejemplos

Si las líneas:

```
int indice;
double vector[MAXVAL];
```

aparecen fuera de cualquier función, *definen* las variables externas `indice` y `vector`, y reserva espacio para su almacenamiento.

Las líneas:

```
extern int indice;
extern double vector[];
```

declaran que `indice` es un `int` y que `vector` 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 indice;
double vector[MAXVAL];
```

aparecen fuera de cualquier función, *definen* las variables externas `indice` y `vector`, y reserva espacio para su almacenamiento.

Las líneas:

```
extern int indice;
extern double vector[];
```

declaran que `indice` es un `int` y que `vector` es un arreglo `double` (cuyo tamaño se determina en algún otro lugar), pero no crea las variables ni les reserva espacio.

Archivo 1

```
int indice = 0;
double vector[MAXVAL];
```

Archivo 2

```
extern int indice;
extern double vector;

void f1(double f) { . . . }
double f2(void) { . . . }
```

Repaso: Definición vs. declaración

Definición: provoca que se reserve espacio para el almacenamiento

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

Repaso: Definición vs. declaración

Definición: provoca que se reserve espacio para el almacenamiento

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

- 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.

(*externo* en contraste a *interno*)