Clase #22 de 29
Declaraciones

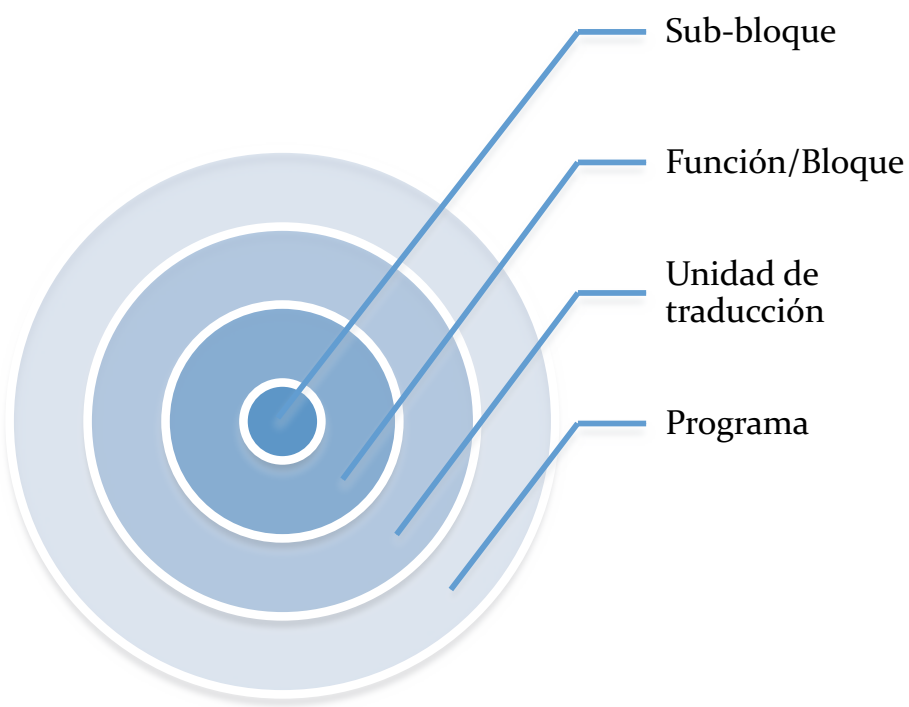
Septiembre 30, Martes
Octubre 1ro, Miércoles

Variables Externas y Alcance

K&R 1.10 Variables Externas y Alcance

Última parte de Introducción a ANSI C

Alcance y Vinculación



- Programa
  - ¿Qué es un programa?
    - Conjunto de Unidades de Traducción (UT, archivos fuente .c)
  - Vinculación entre todas las UT
  - o entre solo algunas UT
  - o entre algunas funciones de algunas UT
- UT
  - Alcance entre todas las funciones
  - o vinculación entre solo algunas funciones
- Función
  - Alcance en el bloque del cuerpo que la define
- Sub Bloque
  - Alcance en el subloque.

Variables Automáticas versus Variables Externas

Variables Automáticas

- Variable Automática: privada o local
  - Solo la función tiene acceso directo
  - ¿Parámetros?
    - Son variables automáticas
- ¿Por qué automáticas?
  - Reserva y liberación de forma automática
    - Push de argumentos
    - Pop en parámetros
  - Comienzan al invocar la función
  - Desaparecen al terminar
  - No retienen valor entre invocaciones
  - Sin inicialización ⇒ Basura
- Mismo Nombre ⇏ Mismo Objeto
  - Variable i de getline y de copy

Variables Externas

- Definidas externas a toda función: Públicas o Globales
- Acceso directo, por nombre, desde "todas" las funciones de la unidad de traducción
  - Antes de usar un nombre de variable, debe estar declarado
  - Comparar con el bloque más externo de Pascal
- Alternativa a lista de argumentos para comunicación entre funciones
- Definición y Declaración
  - Definida una sola vez, de forma externa ⇒ Reserva de memoria
  - Declaradas (tipo de dato) en cada función que la usa
    - De forma Explícita con keyword extern
    - De forma Implícita por contexto y orden
- Vinculadas con otras
  - El Linker conecta las variables
  - ¿Entre otras unidades de traducción o en una misma unidad de traducción?
- Son variables estáticas
  - Permanentes ⇒ Retienen el valor
  - Inicialización en cero.

Duración del Almacenamiento, Tiempo de Vida, Alcance y Vinculación

Storage Duration, Lifetime, Scope, Linkage

- Duración del Almacenamiento, Tiempo de Vida (Storage Duration, Lifetime)
  - Estática
    - Externa
    - Keyword static
  - Automática
    - Parámetros
    - Bloque
    - Keyword auto
  - "Alocada"
    - Funciones estándar
      - malloc
      - calloc
      - realloc
      - free
  - Thread
    - C99
- Alcance (Scope)
  - Desde la declaración
    - En UT
    - En función
    - En bloque
- Retención de valor entre invocaciones
  - Variable declarada dentro de la función, con la keyword static
  - Alcance: bloque
  - Duración: estática
- Vinculación (Linkage)
  - Funciones y variables
    - Entre varias UT
      - Vinculación Externa
    - Dentro de UT
      - Vinculación Interna
  - Sin vinculación
    - struct, enum, typedef, label

Ejemplo – Versión especializada de Imprimir la línea más larga (1 de 5)

```
#include <stdio.h>
#define MAXLINE 1000  /* maximum input line size */

/*Analizar semántica*/
int max;                /* maximum length seen so far */
char line[MAXLINE];     /* current input line */
char longest[MAXLINE]; /* longest line saved here */

int getline(void);
void copy(void);
```

- Declaraciones externas de variables
- Prototipos, ya se usaron como declaraciones externas, pueden hacerse privados a las UT con palabra reservada static
- Definición de variables y de funciones.

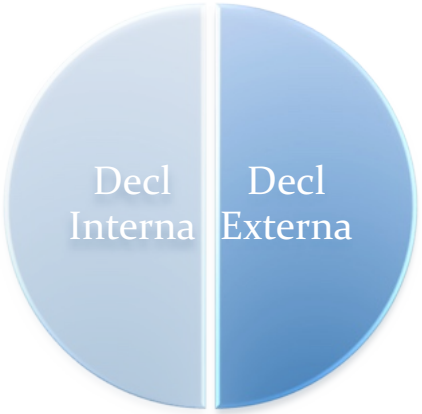
Ejemplo – Versión especializada de Imprimir la línea más larga (2 de 5)

```
int main(void){
  int len;

  /* Analizar semántica */
  extern int max;
  extern char longest[];

  …
```

- Declaración externa
  - Fuera de cualquier función
- Declaración con extern
  - usa extern delante
  - Es opcional si existe definición previa
  - Declaraciones con extern en header (.h)
    - Para vincular entre UT
    - Convención y razón del nombre
  - Puede ser una declaración externa o interna
  - Puede ser definición.



Ejemplo – Versión especializada de Imprimir la línea más larga (3 de 5)

```
    /* Uso de externas y de automáticas */
    max = 0;
    while ((len = getline()) > 0)
      if (len > max) {
        max = len;
        copy();
      }

    if (max > 0) /* there was a line */
      printf("%s", longest);

    return 0;
}
```

Ejemplo – Versión especializada de Imprimir la línea más larga (4 de 5)

```
int getline(void){
  int c, i;
  extern char line[]; /*Analizar semántica*/

  for(
    i=0;
    i < MAXLINE - 1 && (c=getchar())!=EOF && c!='\n';
    ++i
  )
    line[i] = c;

  if (c == '\n') {
    line[i] = c;
    ++i;
  }

  line[i] = '\0';

  return i;
}
```

Ejemplo – Versión especializada de Imprimir la línea más larga (5 de 5)

```
void copy(void){
    int i;

    /*Analizar semántica*/
    extern char line[], longest[];

    i = 0;
    while ((longest[i] = line[i]) != '\0')
        ++i;
}
```

Identificar declaraciones, definiciones, variables automáticas y variables externas

```
#include <stdio.h>
#define MAXLINE 1000

int max;
char line[MAXLINE];
char longest[MAXLINE];

int getline(void);
void copy(void);

int main(void){
  int len;

  extern int max;
  extern char longest[];

  max = 0;
  while ((len = getline()) > 0)
    if (len > max) {
      max = len;
      copy();
    }

  if (max > 0)
    printf("%s", longest);

  return 0;
}
```

```
int getline(void){
     int c, i;
     extern char line[];

     for(
          i=0;
          i < MAXLINE - 1 &&
          (c=getchar())!=EOF && c!='\n';
          ++i
     )
          line[i] = c;

     if (c == '\n') {
          line[i] = c;
          ++i;
     }

     line[i] = '\0';

     return i;
}

void copy(void){
     int i;
     extern char line[], longest[];

     i = 0;
     while((longest[i]=line[i])!='\0')
          ++i;
}
```

Utilización de variables externas

- Pueden ser modificadas accidentalmente
  - Definir al final y declarar solo en las funciones que las deben usar
  - Un programa C puede estar compuesto por varios ó UT: escribir módulos chicos
- Similares a los atributos de instancia o de clase
  - Accesibles por todos los métodos de esa clase
  - En este caso la clase es análoga a la UT
- Las variables externas siempre están disponibles
  - Ventajas
    - Disponibilidad
    - Lista de argumentos más cortas
    - Simplifica comunicación
    - Objetos únicos: stdin, CPU
  - Desventajas
    - Acoplamiento no obvio
    - Bugs
    - Mantenimiento
- En este caso perdemos la oportunidad de crear dos funciones genéricas: getline y copy
  - ANSI provee: fgets y strcpy.

Blibliografía

- [K&R1988] 1.10 Variables Externas y Alcance
- [K&R1988] 4.3 Variables Externas
- [K&R1988] 4.4 Reglas de Alcance
- [K&R1988] 4.6 Variables Estáticas

Ejercicios

- De ejemplos de variables
  - Con alcance de bloque (i.e., local), con duración estática
  - Con alcance de archivo, con vinculación interna
- De ejemplos de identificadores
  - Sin vinculación
  - Con vinculación interna
  - Con vinculación externa
  - Con alcance función
- Explique que es la vinculación de identificadores
- Explique la diferencia entre duración y alcance
- Explique la diferencia entre espacio de nombre y alcance
- De ejemplo de nombre de identificadores léxicamente iguales pero que pertenecen a diferentes espacios de de nombre
- Comparación de Lenguajes: elija un lenguaje diferente a C y a C++ y ejemplifique los siguientes conceptos:
  - Duración
  - Alcance
  - Espacio de nombre

Declaraciones Complicadas

Declaraciones complicadas:

Traducir de C a Lenguaje Natural [*K&R1988 5.12*]

```
int i;
i: int

int a[10];
a: array[10] of int

int *p;
p: pointer to int

int f();
f: function returning int

int *fp();
fp: function returning pointer
to int

int (*pf)();
pf: pointer to function
returning int

void *comp()
comp: function returning
pointer to void

void (*comp)()
comp: pointer to function returning
void

char **argv
argv: pointer to pointer to char

int daytab[13]
daytab: array[13] of int

int (*daytab)[13]
daytab: pointer to array[13] of int

int *daytab[13]
daytab: array[13] of pointer to int

char (*(*x())[])()
x: function returning pointer to
array[] of pointer to function
returning char

char (*(*x[3])())[5]
x: array[3] of pointer to function
returning pointer to array[5] of char.
```

Sintaxis de C: Declaraciones

Declaraciones

Sintaxis y un poco de semántica
Conceptos principales
- *Forma general*
- *Especificadores de declaración*
- *struct*
- *union*
- *enum*
- *typedef*
- *Declaradores*
- *Inicializadores*

Bibliografía
- *MUCH*
- *K&R*

Declaraciones (1/11) – Axioma

declaración
    especificadores-de-declaración *lista-de-declaradores-inic?* **;**

especificadores-de-declaración
    calificador-de-tipo *especificadores-de-declaración?*
    especificador-de-clase-de-almacenamiento *especificadores-de-declaración?*
    especificador-de-tipo *especificadores-de-declaración?*

lista-de-declaradores-inic
    declarador-inic
    lista-de-declaradores-inic **,** *declarador-inic*

declarador-inic
    declarador
    declarador **=** *inicializador*

Declaraciones (2/11) – Almacenamiento y Tipo

especificador-de-clase-de almacenamiento

- **`static`**
- **`auto`**
- **`register`**
- **`extern`**
- **`typedef`**

especificador-de-tipo

- **`void`**
- **`char`**
- **`short`**
- **`int`**
- **`long`**
- **`float`**
- **`double`**
- **`signed`**
- **`unsigned`**
- *especificador-de-struct-o-union*
- *especificador-de-enum*
- *nombre-de-typedef*

Declaraciones (3/11) – Estructuras y Uniones

especificador-de-struct-o-union
    struct-o-union *identificador?* **{** *lista-de-declaraciones-struct* **}**
    struct-o-union *identificador*

struct-o-union
    `struct`
    `union`

lista-de-declaraciones-struct
    declaración-struct
    lista-de-declaraciones-struct *declaración-struct*

Declaraciones (4/11) – Estructuras y Uniones (cont.)

declaración-struct
    lista-de-especificadores-calificadores *lista-de-declaradores-struct* ;

lista-de-especificadores-calificadores
    especificador-de-tipo *lista-de-especificadores-calificadores?*
    calificador-de-tipo *lista-de-especificadores-calificadores?*

lista-de-declaradores-struct
    declarador-struct
    lista-de-declaradores-struct , *declarador-struct*

declarador-struct
    declarador
    declarador? : *expresión-constante*

Declaraciones (5/11) – Enumeraciones

especificador-de-enum

enum *identificador?* **{** *lista-de-enumeradores* **}**

enum *identificador*

lista-de-enumeradores

enumerador

lista-de-enumeradores **,** *enumerador*

enumerador

constante-de-enumeración

constante-de-enumeración **=** *expresión-constante*

Declaraciones (6/11) – Calificadores de Tipo

calificador-de-tipo

const

volatile

```
const struct s { int mem; } cs = { 1 };
struct s ncs; /* the object ncs is modifiable */
typedef int A[2][3];
const A a = {{4,5,6},{7,8,9}}; /* array of array of const int
*/
int *pi;
const int *pci;

ncs = cs; /* valid */
cs = ncs; /* violates modifiable lvalue constraint for = */
pi = &ncs.mem; /* valid */
pi = &cs.mem; /* violates type constraints for = */
pci = &cs.mem; /* valid */
```

Declaraciones (7/11) – Declaradores

declarador
    puntero? declarador-directo

declarador-directo
    identificador
    (*declarador* **)**
    declarador-directo **[** *expresión-constante?* **]**
    declarador-directo **(** *lista-tipos-parámetros* **)**
    declarador-directo **(** *lista-de-identificadores?* **)**

puntero
    ***** *lista-calificadores-tipos?*
    ***** *lista-calificadores-tipos? puntero*

lista-calificadores-tipos
    calificador-de-tipo
    lista-calificadores-tipos calificador-de-tipo

Declaraciones (8/11) – Declaradores: Parámetros Funciones

lista-tipos-parámetros
    lista-de-parámetros
    lista-de-parámetros , . . .

lista-de-parámetros
    declaración-de-parámetro
    lista-de-parámetros , *declaración-de-parámetro*

declaración-de-parámetro
    especificadores-de-declaración declarador
    especificadores-de-declaración declarador-abstracto?

lista-de-identificadores
    identificador
    lista-de-identificadores , *identificador*

Declaraciones (9/11) – Nombre de tipo y declarador abstracto

nombre-de-tipo
    lista-de-calificadores *declarador-abstracto?*

declarador-abstracto
    puntero
    puntero? *declarador-abstracto-directo*

declarador-abstracto-directo
    (*declarador-abstracto* **)**
    declarador-abstracto-directo? **[** *expresión-constante?* **]**
    declarador-abstracto-directo? **(** *lista-tipos-parámetros?* **)**

Declaraciones (10/11) – Typedef

nombre-de-typedef
identificador

- Es diferente a *nombre-de-tipo*
- Ejemplos de nombre-de-tipo
  - `struct Punto {double x, y};`
  - `struct Punto p;`
  - `int *a[20];`
  - `sizeof (int *[20])`

- Ejemplos typedef
  - `typedef int *Avpi[20];`
  - `Avpi a;`
  - `sizeof ( Avpi )`

  - `typedef struct Punto{double x, y} Punto;`
  - ó
  - `typedef struct {double x, y} Punto;`
  - `struct Punto p;`
  - `.`

Declaraciones (11/11) –Inicialización

inicializador
    expresión-de-asignación
    { *lista-de-inicializadores* **}**
    { *lista-de-inicializadores* **,** **}**

lista-de-inicializadores
    inicializador
    lista-de-inicializadores **,** *inicializador*

constante-de-enumeración
    identificador

Glosario

- Especificadores de declaración
- Especificadores de clase de almacenamiento
- Especificadores de tipo
- Calificadores de tipo
- Declarador
- Inicializador
- **`struct`**
- **`union`**
- **`enum`**
- **`typedef`**

¿Consultas?

Fin de la clase