Principios de Programación

El lenguaje C

1. Variables locales y globales

1.1. Variables locales

Las funciones permiten al programador modularizar un programa. Todas las variables declaradas en las definiciones de función son *variables locales* - son conocidas sólo en la función en la cual están definidas (ninguna otra función tiene acceso a ellas). La mayor parte de las funciones tienen una lista de *parámetros*. Los parámetros de una función también son variables locales. Las variables locales comienzan su existencia cuando la función es llamada y desaparecen cuando la función termina su ejecución, por esta razón se conocen como *variables automáticas*. Las variables automáticas no retienen sus valores de una llamada a otra.

1.2. Variables globales

Es posible definir funciones que son *externas* a todas las funciones, esto es variables globales que pueden ser accedidas por cualquier función. Pueden ser utilizadas en el lugar de las listas de parámetros para comunicar información entre funciones. Las variables externas existen permanentemente y retienen su valores aun despues de que las funciones que las setean han terminado su ejecución.

Las variables externas deben ser definidas exactamente una vez, fuera de todas las funciones, esto les asigna almacenamiento. La variable debe también ser declarada en cada función que quiera acceder a ella. La declaración debe ser una sentencia *extern*. Veamos un ejemplo:

```
#include <stdio.h>

#define MAX_LINEA 1000 /* maximo tamanio de linea de entrada */

int max; /* maximo tamanio visto hasta el momento */
char linea[MAX_LINEA] /* linea actual */
char mas_larga[MAX_LINEA] /* linea mas larga hasta el momento */

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

/* imprimir linea de la entrada mas larga */
main ()
{
      int len;
      extern int max;
      extern char mas_larga[];

      max = 0;
```

```
        while ((len = getline()) ¿0)
                if (len ¿max)
                {
                  max=len;
                  copio();
                }
        if (max > 0) printf(" %s", mas_larga);
        system("PAUSE");
}

/* getline: lee una linea de la entrada y devuelve su largo */
int getline(void)
{
    int c,i;
    extern char linea[];

    for (i=0;i < MAX_LINEA - 1 && (c=getchar())!EOF && c != '\n';++i)
        linea[i]=c;
    if (c=='\n') {
      linea[i]=c;
      ++i; }
    linea[i]='\0';
    return i;
}

/* copio: copia una linea en otra */
void copio(void)
{
      int i;
      extern char linea[], mas_larga[];

      i=0;
      while ((mas_larga[i]=linea[i]) != '\0') i++;
}
```

En ciertas circunstancias las declaraciones *extern* pueden omitirse: si la definición de una variable externa aparece en el archivo antes de su uso en una función particular no hay necesidad de declaración *extern* en la función.

2. Reglas de alcance

El alcance de un nombre es la parte del programa en la cual el nombre está definido.

Para una variable automática declarada en una función el alcance es la función en la cual el nombre está declarado y variables con el mismo nombre en distintas funciones no están relacionadas. Lo mismo es cierto para los argumentos de la función.

El alcance de una variable externa es desde el punto en el cual es definida en un archivo hasta el final del archivo. Si necesitamos referenciar una variable

externa antes de ser definida o si es utilizada en un archivo distinto es necesario colocar la declaración extern.

3. Clases de almacenamiento

La clase de almacenamiento de un identificador ayuda a determinar su duración de almacenamiento y su alcance. La duración de almacenamiento de un identificador es el periodo durante el cual dicho identificador existe en memoria. El alcance de un identificador en un programa es donde puede ser referenciado.

3.1. Variables Estáticas

Existen dos tipos de identificadores con persistencia estática: los identificadores externos y las variables locales declaradas como **static**.

Pueden ser internas o externas. Variables estáticas internas son locales a la función en la que se definen pero se diferencian en que continuan en existencia entre una llamada y otra de la función. Estas variables conservan el valor con el que salen de la función. Luego, las variables internas estáticas tienen almacenamiento permanente en la función.

Una variable estática externa es conocida en el resto del archivo en el cual es declarada pero no en ningún otro archivo.

Variables estáticas se declaran prefijando a la declaración la palabra **static**.

Es posible declarar funciones como estáticas, esto hace que su nombre sea desconocido fuera del archivo en el cual es declarada.

3.2. Variables registro

Los datos de un programa en la versión en lenguaje de máquina para cálculos y otros procesos normalmente se cargan en registros.

Una declaración **register** le avisa al compilador que la variable en cuestión va a ser utilizada ampliamente. Cuando es posible, variables registro se colocan en los registros de la máquina lo cual va a resultar es programas más rapidos.

Las declaraciones son de la forma:

```
register int x;
register char c;
```

las variables registro se pueden aplicar solo a variables automáticas y a los parámetros formales de una función.

El compilador puede ignorar declaraciones register. Puede que no exista un número suficiente de registros disponibles.

4. Más sobre reglas de alcance

Hay cuatro alcances posibles:

1. alcance de función
2. alcance de archivo

3. alcance de bloque

4. alcance de prototipo de función

4.1. Alcance de función

Las etiquetas (identificador seguido por dos puntos) son los únicos identificadores con *alcance de función*. Pueden ser utilizadas en cualquier parte dentro de la función en la cual aparecen, pero no pueden ser utilizadas fuera del cuerpo de la función. Se utilizan en estructuras switch (como etiquetas case) y en **goto**.

4.2. Alcance de archivo

Un identificador declarado por fuera de cualquier función tiene alcance de archivo. Es conocido en todas las funciones desde el punto donde el identificador se declara hasta el final del archivo. Las variables globales, las definiciones de función y los prototipos de función colocados fuera de una función tienen alcance de archivo.

4.3. Alcance de bloque

Los identificadores dentro de un bloque tienen alcance de bloque. Este termina en la llave derecha de terminación del bloque. Las variables locales declaradas al principio de una función tienen alcance de bloque al igual que los parámetros de la función. Cuando los bloques están anidados y un identificador del bloque externo tiene el mismo nombre que un identificador del bloque interno, el identificador del bloque externo estará "oculto"hasta que el bloque interno termine.

Las variables locales declaradas static tendrán alcance de bloque aunque existan a partir del momento en que empieza a ejecutarse el programa.

4.4. Alcance del prototipo de función

Los únicos identificadores con este alcance son los que se declaran en la lista de parámetros del prototipo de una función. El compilador ignora estos nombres.

4.5. Ejemplo

```
/* Un ejemplo de alcance */
#include <stdio.h>

void a(void);
void b(void);
void c(void);

int x = 1; /* variable global */

main ()
{
      int x=5;
```

```
        printf("x local en alcance exterior de main es %d\n",x);
        { /* nuevo alcance */
          int x=7;
          printf("x local en alcance interior de main es %d\n",x);
        }
        printf("x local en alcance exterior de main es %d\n",x);
        a();
        b();
        c();
        a();
        b();
        c();
        printf("x local en main es %d\n",x);
        system("PAUSE");
}

void a(void)
{
      int x = 25;
      printf("\n x local en a es %d luego de entrar a a\n",x);
      ++x;
      printf("x local en a es %d antes de salir de a\n",x);
}

void b(void)
{
      static int x=50;
      printf("\n x local static en b es %d luego de entrar a b\n",x);
      ++x;
      printf("x local static en b es %d antes de salir de b\n",x);
}

void c(void)
      printf("\n x global es %d al entrar a c\n",x);
      x*=10;
      printf("x global es %d antes de salir de c\n",x);
}
```

La salida del programa es:

```
x local en alcance exterior de main es 5
x local en alcance interior de main es 7
x local en alcance exterior de main es 5

x local en a es 25 luego de entrar a a
x local en a es 26 antes de salir de a

x local static en b es 50 luego de entrar a b
x local static en b es 51 antes de salir de b
```

x global es 1 al entrar a c
x global es 10 antes de salir de c

x local en a es 25 luego de entrar a a
x local en a es 26 antes de salir de a

x local static en b es 51 luego de entrar a b
x local static en b es 52 antes de salir de b

x global es 10 al entrar a c
x global es 100 antes de salir de c
x local en main es 5