Clases de Almacenamiento

Clases de almacenamiento

- La clase de almacenamiento de un identificador permite determinar su duración de almacenamiento, su alcance y su enlace.
  - **Duración**: un identificador puede existir durante todo el programa, sólo en algunos entornos o ser creado reiteradamente,
  - El **alcance** de un identificador indica donde puede ser referenciado.
  - El **enlace** de un identificador determina, cuando el programa está formado por varios archivos, si el identificador es conocido en un único archivo fuente o en cualquiera de ellos.

Clases de almacenamiento

- Los cuatro especificadores se dividen en
  - Persistencia automática (**auto** y **register**)
    - Estos identificadores se aplican a variables. Se crean al comenzar el bloque donde están definidas y se destruyen al salir del bloque.
    - Es una forma de ahorrar memoria.
  - Persistencia estática (**static** y **extern**)
    - Estos identificadores se aplican a variables y nombres de función. Existen desde el momento en que se inicia la ejecución del programa.
    - Por ahora sólo aplicaremos el especificador **static** a variables.

```c
#include <stdio.h>
int main()
{   auto int suma = 0;
    register int indice;

    for (indice=1; indice<10000; indice++)
    {
        int MuyDinamica = 0;

        MuyDinamica++;
        suma += MuyDinamica;
        if (indice % 1000==0) {
            printf("i=%6d  ", indice);
            printf("suma = %6d  ", suma);
            printf("MuyDinamica = %d\n",MuyDinamica);
        }
    }
    return 0;
}
```

```c
#include <stdio.h>
int main()
{   auto int suma = 0;
}
```

- Sólo las variables tienen persistencia automática.
- La palabra **auto** declara de forma explícita las variables de persistencia automática.
- Por omisión, las variables locales tienen persistencia automática por lo que la palabra **auto** rara vez se utiliza.

```c
#include <stdio.h>
int main()
{   auto int suma = 0;
    register int indice;
}
```

- La palabra **register** puede ser utilizada sólo con variables automáticas.
- Esta declaración sugiere que se coloque la variable entera *indice* en uno de los registros de la computadora.
- El compilador puede ignorar declaraciones **register** (por ejemplo, quizás no exista un número suficiente de registros).
- El tener variables directamente almacenadas en los registros elimina la sobrecarga de su traslado de memoria a los registros y el posterior almacenamiento de los resultados en memoria.

```c
#include <stdio.h>
int main()
{   auto int suma = 0;

    register int indice;

    for (indice=1; indice<10000; indice++)
    {
        int MuyDinamica = 0;
    }
    re
}
```

- Esta variable es local al **for** es decir que sólo existe cuando el for se está ejecutando.
- La variable automática *MuyDinamica* se crea al comienzo de la iteración y se destruye cuando el control llega a la llave que cierra el bloque.
- Cada vez que se crea, esta declaración dice que se inicializa en cero.

```c
#include <stdio.h>
int main()
{   auto int suma = 0;
    register int indice;

    for (indice=1; indice<10000; indice++)
    {
        int MuyDinamica = 0;

        MuyDinamica++;
        suma += MuyDinamica;
        if (indice % 1000==0) {
            printf("i=%6d  ", indice);
            printf("suma = %6d  ", suma);
            printf("MuyDinamica = %d\n",MuyDinamica);
        }
    }
    return 0;
}
```

- Qué imprime?

Variables.c

Persistencia estática

- Existen dos tipos de identificadores con persistencia estática
  - Los identificadores externos (variables globales y nombres de función). Estos identificadores, por omisión, pertenecen a la clase de almacenamiento **extern**.
    - Las variables globales y las funciones pueden ser referenciadas por cualquier función luego de su declaración.
  - Las variables locales declaradas con el especificador de clase de almacenamiento **static**.
    - Son aun conocidas sólo dentro del bloque donde fueron definidas pero conservan su valor cuando éste termina.
    - La próxima vez que se ejecute el bloque, la variable local **static** contendrá el valor que tenía cuando el bloque terminó por última vez.

```c
#include <stdio.h>
int contador;
int main()
{   int indice;
    for (indice=1; indice<10; indice++)
            auxiliar();

    MuestraLlamados();
    printf("contador = %d\n", contador);
    return 0;
}
int CantLlamados;

void auxiliar(void)
{ static int suma =0;
  suma = suma + 1;
  printf("Suma = %d\n", suma);
  CantLlamados ++;
  contador++;
}
void MuestraLlamados(void)
{    CantLlamados ++;
     printf("Llamados = %d\n", CantLlamados);
}
```

Variable global conocida en todo el programa. Se inicializa en cero automáticamente (por ser global)

Variable global sólo conocida por las funciones que están definidas debajo. Se inicializa en cero automáticamente (por ser global)

```c
#include <stdio.h>
int contador;
int main()
{   int indice;
    for (indice=1; indice<10; indice++)
            auxiliar();

    MuestraLlamados();
    printf("contador = %d\n", contador);
    return 0;
}
int CantLlamados;

void auxiliar(void)
{ static int suma =0;
  suma = suma + 1;
  printf("Suma = %d\n", suma);
  CantLlamados ++;
  contador++;
}
void MuestraLlamados(void)
{    CantLlamados ++;
     printf("Llamados = %d\n", CantLlamados);
}
```

Variable automática. Son las variables habituales. También se pudo haber declarado como:

auto int indice

Variable estática. Se inicializa sólo la primera vez y luego estará disponible con su valor anterior en las sucesivas llamadas de la función.

```c
#include <stdio.h>
int contador;
int main()
{   int indice;
    for (indice=1; indice<10; indice++)
            auxiliar();

    MuestraLlamados();
    printf("contador = %d\n", contador);
    return 0;
}
int CantLlamados;

void auxiliar(void)
{ static int suma =0;
  suma = suma + 1;
  printf("Suma = %d\n", suma);
  CantLlamados ++;
  contador++;
}
void MuestraLlamados(void)
{    CantLlamados ++;
     printf("Llamados = %d\n", CantLlamados);
}
```

- Qué imprime?

Variables1.c

```c
#include <stdio.h>
int main()
{   auto int suma = 0;

    register int indice;

    for (indice=1; indice<10000; indice++)
    {
        static int MuyDinamica = 0;

        MuyDinamica++;
        suma += MuyDinamica;
        if (indice % 1000==0) {
              printf("i=%6d  ", indice);
              printf("suma = %6d  ", suma);
              printf("MuyDinamica = %d\n",MuyDinamica);
        }
    }
    return 0;
}
```

- Qué imprime?

Variables2.c