4. Classes Armazenamento de Variáveis

Um programa aloca memória para o armazenamento dos dados necessários à sua execução:

✔ A **forma** como essa alocação de memória é realizada, chama-se ***classe de armazenamento***.

✔ Até agora foram vistas duas classes de armazenamento:

- Armazenamento estático em:
  - Variáveis globais
  - Constantes definidas pelo programador ou codificadas dentro do programa.
- Armazenamento dinâmico em:
  - Variáveis locais e parâmetros de funções
  - Espaço de memória alocado dinamicamente de forma explicita através de `malloc()`.

✔ Além das formas acima, “C” possui 4 ***especificadores de classe de armazenamento***, que podem ser usados para *complementar a declaração* de uma variável:

- `extern` var. global externa ao arquivo fonte
- `static` var. explicitamente estática
- `register` var. de acesso rápido
- `auto` var. explicitamente local

✔ Esses especificadores serão usados para informar ao compilador de que forma a variável cuja declaração o sucede, deve ser armazenada.

Forma geral:

```
especificador_armazenamento tipo   nome_da_variável;
```

4.1. extern

- Usado para variáveis globais.
- Indica que a variável em questão é a mesma que já está sendo declarada em um outro módulo ou arquivo do programa.

✔ Razão:

- Em “C” uma variável global só é global dentro de um arquivo de programa. Em outros arquivos ou módulos do mesmo program, ela é desconhecida.
- Redeclarar uma variável global em todos os arquivos do programa, leva, na verdade à criação de várias variáveis.
- Como “C” permite que módulos de um programa grande sejam compilados separadamente para então serem linkeditados juntos, é necessário que se possa referenciar variáveis globais declaradas em outros arquivos.
- `extern` antes de uma declaração de variável indica que não deve ser gerado código para alocação de memória para esta variável , mas sim um marcador que o linkeditor substitui por uma referência ao endereço da variável em tempo de linkagem.

✔ Solução: Declarar todas as variáveis globais utilizadas em muitos módulos em um arquivo único (tipicamente o módulo contendo a função `main()` ou `meu_programa.h`) :

Arquivo: sistema.c

```
int x, y;
char car;

main ()
{
   ...
   ...
}

int func1()
{
   ...
   ...
}
```

Arquivo: cálculo.c

```
extern int x, z;
extern char car;

void funcAA()
{
      ...
      ...
}

char *funcBB()
{
      ...
      ...
}
```

Compilação:

```
cc -o sistema sistema.c cálculo.c
```

4.2. `static`

- Variáveis com alocação explicitamente estática, **mesmo que sejam variáveis locais em uma função**.
- O especificador `static` indica ao compilador que para esta variável dever ser gerado código para alocação global de memória ao início do programa.
- Variáveis do tipo `static`, apesar de terem um armazenamento "global" são conhecidas exclusivamente dentro de seu escopo (função ou módulo).

4.2.1. Variáveis com armazenamento estático em funções

- **Utilidade**: quando for necessário que variáveis de uma função preservem o seu valor entre chamadas dessa função.
- Se variáveis estáticas não fossem permitidas, variáveis globais teriam de ser admitidas, abrindo brechas para efeitos colaterais.
- Uma variável estática declarada em uma função **não pode** ser acessada de fora da função onde foi declarada.
- Declaração:

```
int serie ()
{
   static int numero_de_serie = 3; // Inicial. só na 1ª
                                   // chamada à função.
   numero_de_serie = numero_de_serie + 3;
   return (numero_de_serie);
}
```

4.2.2. Variáveis globais com armazenamento estático

- Uma variável global é por definição, uma variável estática.
- Por que então declarar uma variável global explicitamente como estática ?
  - Variáveis globais com a especificação static são limitadas ao escopo do arquivo (módulo) onde estão declaradas.
  - É uma forma de se ter uma variável global para, por exemplo, um conjunto de funções, mas que não pode ser acessada por outras funções externas ao arquivo.

<u>Arquivo</u>: sistema.c

```
int x, y;
static char car;

main ()
{
   ...
   ...
}

int func1()
{
   ...
   ...
}
```

<u>Arquivo</u>: cálculo.c

```
extern int x, z;
extern char car;

void funcAA()
{
      ...
      ...
}

char *funcBB()
{
      ...
      ...
}
```

Erro !

4.3. `register`

✔ Criada par casos onde é necessário um **armazenamento e acesso muito rápido** a uma variável.

✔ Originalmente instruia o compilador para gerar código para armazenar estas variáveis em um registrador da CPU ao invés da RAM.

- Dessa forma não era necessário acesso à memória para determinar ou modificar seu valor.
- Assim, operações repetidas utilizando estas variáveis podiam ocorrer muito mais rápido.
- Utilidade: computação gráfica, cálculo numérico.

✔ Padrão ANSI define atualmente que o compilador deve apenas procurar a ***forma de armazenamento mais rápida possível***.

Exemplo: calcular um número elevado a uma potência inteira.

```
int potência( int valor, register int expoente )
{
   register int temp;
   temp = 1;
   for ( 0; e, e-- )
      temp = temp * valor;
   return (temp);
}
```

4.4. auto

✔ Declaração explícita de uma variável local. Desnecessário, pois o contexto é suficiente.

- Determina a utilização do stack para o armazenamento de seu valor.
- Variáveis `auto` só podem ser referenciadas de dentro do bloco onde foram declaradas.
- Devem ser declaradas logo no início do bloco.

✔ Exemplos:

```
void f(int valor, register int expoente)
{
   auto int temp;       // auto aqui é desnecessário
   temp = 1;
   for ( 0; e, e-- )
    {
      auto float temp2; // auto aqui é desnecessário
      temp = temp * valor;
      temp2 = temp / 2;
      printf ( ”Metade do valor atual: %f”, temp2 ),
    }
   return (temp);
}
```