Tópicos sobre variáveis

Conteúdo em preparação

Escopo

Por “escopo” de uma variável entende-se em que trechos do código ela pode ser acessada. Há basicamente dois escopos possíveis em C: **local** e **global**:

- **Variáveis globais** podem ser acessadas a partir de qualquer ponto do código-fonte.
- **Variáveis locais** são definidas em um bloco e só podem ser acessadas dentro daquele bloco. É o caso das variáveis definidas dentro de uma função, dos parâmetros da função e de variáveis locais usadas em laços `for`, por exemplo.

O trecho de código a seguir ilustra a declaração e uso de variáveis globais e locais:

achar código mais curto ou função mais simples

```c
#include <stdio.h>

int teto ;  // variável global, pode ser acessada de qualquer lugar

int somaN (int n)  // variável local "n", só pode ser acessada dentro da função
{
  int soma = 0 ;    // idem, "soma"

  for (int i = 0; i < n; i++) // "i" só pode ser acessada dentro deste laço
  {
    int aux = soma + i ; // idem, "aux"
    if (aux < teto)
      soma = aux ;
  }
  return (soma) ;
}

int main ()
{
  int val = 10 ;  // variável local
  printf ("%d\n", somaN (val)) ;
}
```

Uma variável local pode ter o mesmo nome de uma variável global ou de uma variável local declarada em um bloco mais externo. Ao acessar uma variável, será buscada a definição feita no mesmo bloco, ou no bloco externo mais próximo, como mostra o exemplo a seguir:

```c
#include <stdio.h>

int valor = 100 ;  // variável global

void func ()
```

```
{
  int valor = 5 ; // variável local

  printf ("2: %d\n", valor) ;
  {
    int valor = 17 ; // nova variável local, dentro do bloco

    printf ("3: %d\n", valor) ;
  }
  printf ("4: %d\n", valor) ;
}

int main ()
{
  printf ("1: %d\n", valor) ;
  func () ;
  printf ("5: %d\n", valor) ;
}
```

O resultado dessa execução é o seguinte (tente explicá-lo!):

```
1: 100
2: 5
3: 17
4: 5
5: 100
```

Por default, o escopo de uma variável global é limitado ao arquivo onde ela está definida. Todavia, em um projeto com vários arquivos, pode-se referenciar uma variável global definida em outro arquivo usando o modificador `extern`.

Por exemplo, supondo que `arq1.c` e `arq2.c` sejam arquivos de código-fonte do mesmo projeto, e que uma variável global seja definida no primeiro arquivo:

arq1.c

```
// aqui a variável é definida, alocada e inicializada
int default = 100 ;

...
```

Caso deseje-se acessar essa variável global em outro arquivo do mesmo projeto, basta declará-la como `extern`:

arq2.c

```
// aqui a variável é somente declarada
extern int default ;

...
```

A declaração `extern` acima pode ser entendida como "*use a variável global inteira chamada `default`, que já foi declarada e alocada em outro arquivo deste projeto*.

Alocação

Fix Me! Explicar `auto`, `static`, `extern` e `register` com exemplos

- `auto` : informa que a variável deve ser alocada automaticamente no início da função ou bloco e liberada ao final do mesmo. É a forma de alocação default das variáveis locais em funções.
- `static` : informa que a variável deve ser alocada do início ao final da execução; é a forma de alocação default das variáveis globais.
- `register` : informa ao compilador que a variável deve ser alocada em um registrador da CPU, se houver registrador livre, para maior desempenho.
- `extern` : informa que uma variável global já está definida e alocada em outro arquivo fonte e que não precisa ser alocada novamente. Vide Esta página para uma boa explicação sobre o uso correto da declaração `extern`.

Constantes

Fix Me! Falar sobre `const` e `#define`

- `const` : indica que o valor da variável é constante/fixo, ou seja, que a variável pode ser lida mas não escrita. Fix Me! conferir

Outros modificadores

- `volatile` : informa ao compilador que o valor da variável pode mudar mesmo que nenhuma atribuição a ela ocorra naquele trecho do código. Em consequência, o compilador evita otimizar os trechos de códigos que envolvam essa variável. Esse tipo de comportamento pode ocorrer em programas que envolvam várias *threads* ou que façam entrada/saída mapeada em memória.
- `restrict` : aplicada a ponteiros, informa que aquele ponteiro é o único caminho para acessar os dados apontados por ele (não existe outro ponteiro ou outra forma de acessar aqueles dados); essa informação pode ser usada pelo compilador para otimizações.