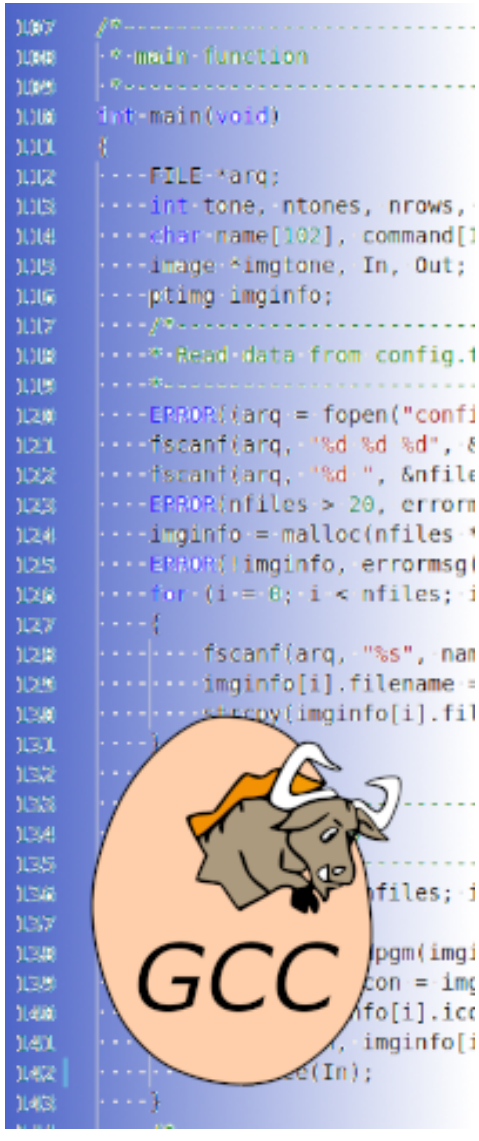

Programação

Ponteiro

Luiz Eduardo da Silva

Universidade Federal de Alfenas

Agenda

Agenda

Definição

Definição

Ponteiros são variáveis que apontam para algum endereço de memória. São usadas para passagem por referência e alocação dinâmica de espaço em memória, ou seja, alocação de espaço de memória que ocorre em tempo de execução dos programas.
Declaração:

```
tipo *variavelponteiro;
```

Exemplo:

```
int *p;
// o conteúdo da variável apontada por p é inteiro
```

Operadores

Os operadores para variáveis ponteiro são:

- & - Devolve o endereço de uma variável. Exemplo:

```
m = &x;
/* m recebe o endereço da variável x */
```

- * - Devolve o conteúdo de memória apontado por um ponteiro. Exemplo:

```
c = *p;
/* c recebe o conteúdo de memória apontado por p */
```

Observação

O símbolo asterisco (*) tem dois usos com ponteiro. Na declaração para definir que a variável é ponteiro e numa expressão para retornar o conteúdo(valor) apontado pela variável ponteiro.

Exemplo

```
1  #include <stdio.h>
2  int main()
3  {
4     /* uma variável inteira        */
5     int x = 10;
6     /* dois ponteiros de inteiros */
7     int *p1, *p2;
8     /* p1 recebe o endereço de x  */
9     p1 = &x;
10    /* p2 recebe o endereço apontado por p1*/
11    p2 = p1;
12    /* escreve o endereço e o valor de x    */
13    printf("%p %d\n", p2, *p2);
14    return 0;
15 }
```

Aritmética

Aritmética de Ponteiros

Duas operações válidas com ponteiros são a adição e a subtração. Quando incrementamos um ponteiro estamos na verdade saltando o tamanho do tipo base na memória do computador.

```
char *ch = 3000;
ch = ch + 1;
int  *i  = 3000;
i = i + 1;
```

Comparação

Comparação de Ponteiros

Pode-se comparar ponteiros normalmente em C. Todas os operadores relacionais são válidos (==, ¡, !=, etc...).
Exemplo:

```
int *p, *q;
...
if (p < q)
  printf ("o endereco apontado por p1 e < que o de q");
```

Ponteiros e matrizes

Na declaração :

```
char str [80], *p1;
```

Tanto a variável str com p1 apontam para um caracter (str aponta para o primeiro caracter do string e p1 para um caracter qualquer). A atribuição:

```
p1 = str;
```

Faz com que p1 e str apontem para a primeira posição do string. Então para acessar o quinto elemento do string poderemos proceder de duas formas:

```
str [4] ou *(p + 4)
```

Alocação de memória

Alocação dinâmica de memória

Pode-se alocar (reservar) um espaço de memória durante a execução de um programa em C usando a função:

```
ponteiro = malloc (tamanhoEmBytes);
```

Isto é muito útil quando não sabemos de antemão a quantidade de memória que necessitaremos para um vetor/variável do programa. Para se desfazer do espaço anteriormente alocado para uma variável usamos a função:

```
free(ponteiro);
```

Exemplo

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int i, qtd, *valores;
   puts("Digite a quantidade e os valores:");
   scanf("%d", &qtd);
   // sizeof = tamanho de um tipo em bytes
   valores = malloc(qtd * sizeof(int));
   for (i = 0; i < qtd; i++)
      scanf("%d", valores + i); // ou &valores[i]
   for (i = qtd - 1; i >= 0; i--)
      printf("%d\n", valores[i]);
   free(valores);
   return 0;
}
```