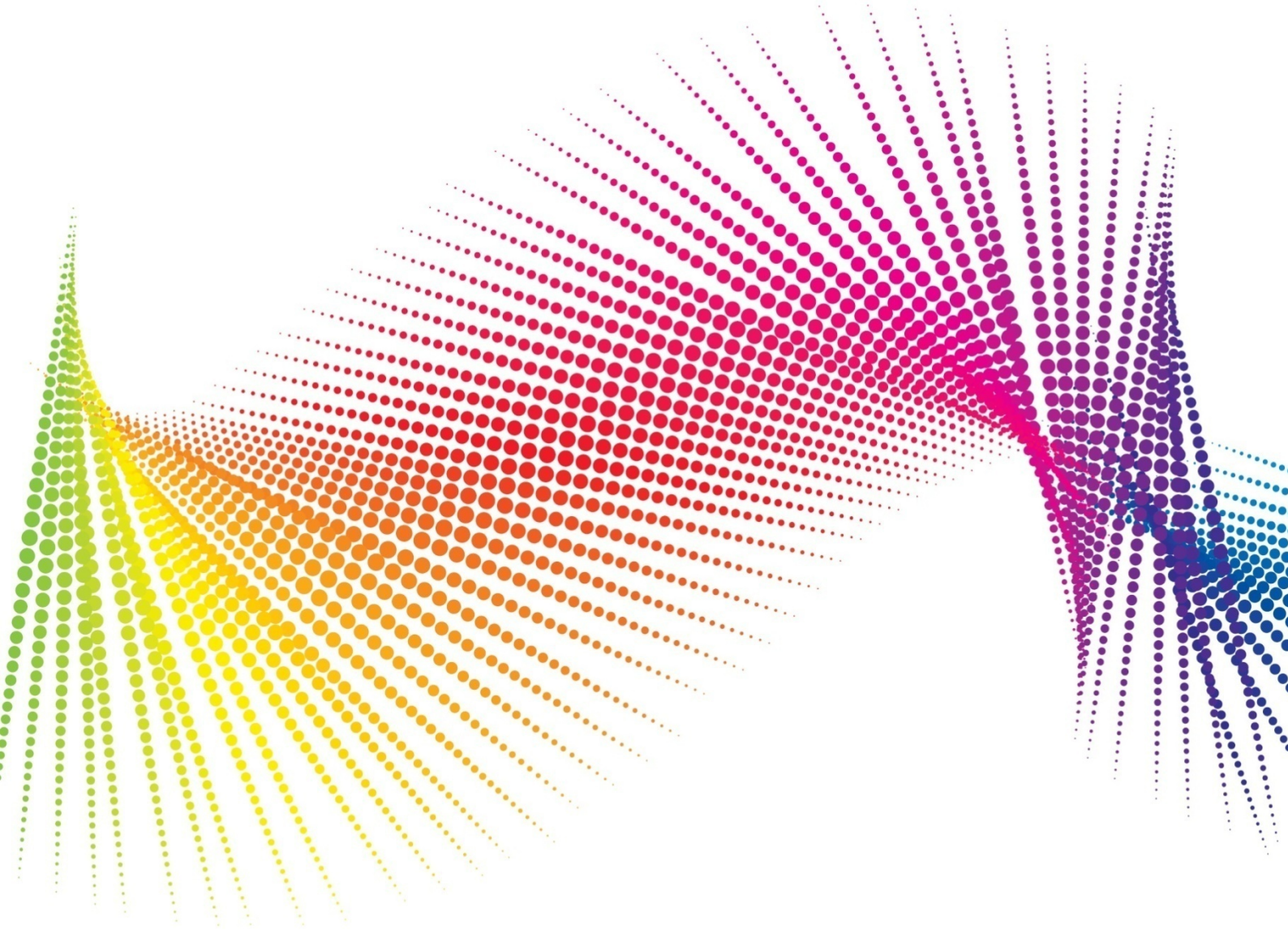
UNINOVE
Universidade Nove de Julho

Estrutura de Dados

Aula 05



Este material é parte integrante da disciplina oferecida pela UNINOVE.
O acesso às atividades, conteúdos multimídia e interativo, encontros virtuais, fóruns de discussão e a comunicação com o professor devem ser feitos diretamente no ambiente virtual de aprendizagem UNINOVE.

Uso consciente do papel.
Cause boa impressão, imprima menos.

Aula 5: Ponteiros

Objetivo: Estudar os **ponteiros**, variáveis especiais capazes de armazenar endereços de memória.

Ponteiros

Caro aluno, como você já sabe, as variáveis do tipo **int** podem armazenar números inteiros; as do tipo **float** podem armazenar números reais; as do tipo **char** armazenam caracteres. E os ponteiros?

Vá até o AVA e assista à animação para conhecer o que são ponteiros. Esta animação faz parte da sequência desta aula e, portanto, é essencial para a sua aprendizagem. Agora que você assistiu à animação, vamos ver isso com mais detalhes no que se segue.

Declaração de Ponteiros

Para se declarar um ponteiro a forma geral é:

```
tipo *nome_do_ponteiro;
```

É o **asterisco (*)** que faz o compilador saber que aquela variável não vai guardar um valor, mas, sim, um endereço de memória capaz de armazenar aquele tipo especificado.

Exemplos de declarações de ponteiros

```
int *pt1;
char *apchar,*pt2;
```

O primeiro exemplo declara um ponteiro do tipo inteiro chamado `pt1`. O segundo declara dois ponteiros para caracteres, `apchar` e `pt2`.

Inicialização de ponteiros com endereços de variáveis já declaradas

Nos exemplos, os ponteiros foram apenas declarados, mas ainda não foram inicializados (como toda variável da linguagem C que é apenas declarada). Isto significa que eles apontam para um lugar indefinido, já que não têm, ainda, um endereço válido. Usar o ponteiro nestas circunstâncias, sem um endereço válido, pode levar a um travamento do computador ou a algo pior.

Um ponteiro deve ser inicializado: apontado para algum lugar conhecido, ou seja, para um endereço de memória válido, antes de ser usado. Isto é de suma importância!

Para atribuir um valor a um ponteiro recém-criado pode-se igualá-lo a um valor de memória conhecido. Mas, como saber a posição na memória de uma variável do programa? Seria muito difícil para o programador escolher um endereço para cada variável que se usa num programa, mesmo porque estes endereços são determinados pelo compilador na hora da compilação e realocados na execução. Assim, o compilador, em parceria com o sistema operacional, faz este trabalho de alocar espaço na memória para cada variável declarada pelo programador.

Mas, para sabermos o endereço de memória de uma variável, basta usarmos o operador **& (endereço de memória de)**. Vejamos o exemplo:

```
int cont=10;
  int *pt1;
pt1=&cont;
```

Neste trecho criamos um inteiro **cont** com o valor 10 e um apontador para um inteiro **pt1**. A expressão **&cont** fornece o endereço de **cont**, que foi armazenado em **pt1**. Fácil, não é? Note que o valor de **cont** não foi alterado, continua valendo 10.

Como foi colocado um endereço em **pt1**, ele está agora "pronto" para ser usado. Podemos, por exemplo, alterar o valor de **cont** usando **pt1**. Para isso, é necessário usar o operador "inverso" do operador **& (endereço de memória de)**, que é o operador ***(conteúdo do endereço de memória)**.

No exemplo, uma vez que foi feito **pt1=&cont;** a expressão ***pt1** é equivalente ao próprio **cont**. Isto significa que, caso você queira mudar o valor de **cont** para 15, basta fazer ***pt1=15**.

Uma observação importante: apesar do símbolo ser o mesmo, o operador ***** (multiplicação) não é o mesmo operador que o ***** (referência de ponteiros). Para começar o primeiro é binário e o segundo é unário pré-fixado.

Outra observação igualmente importante: **o * é usado tanto para declarar um ponteiro, como para mostrar o conteúdo do endereço de memória**. Temos que ficar atentos ao contexto em que é usado para sabermos o que ele indica.

Vamos fazer um pequeno resumo. Leia tudo novamente até estar certo que entendeu, ok?

& ..... operador que mostra o endereço de memória.

* ..... operador que mostra o conteúdo de um endereço de memória.

***** ..... operador que indica referência a um ponteiro; usado para declarar um ponteiro.

Reforçando o conceito de variáveis e ponteiros

Caro aluno, cada espaço da memória do computador tem um endereço associado a ele. Quando declaramos uma variável, o compilador reserva um espaço de memória para poder armazenar o valor desta variável, e quando se deseja saber este valor, o programa procura no endereço de memória da variável.

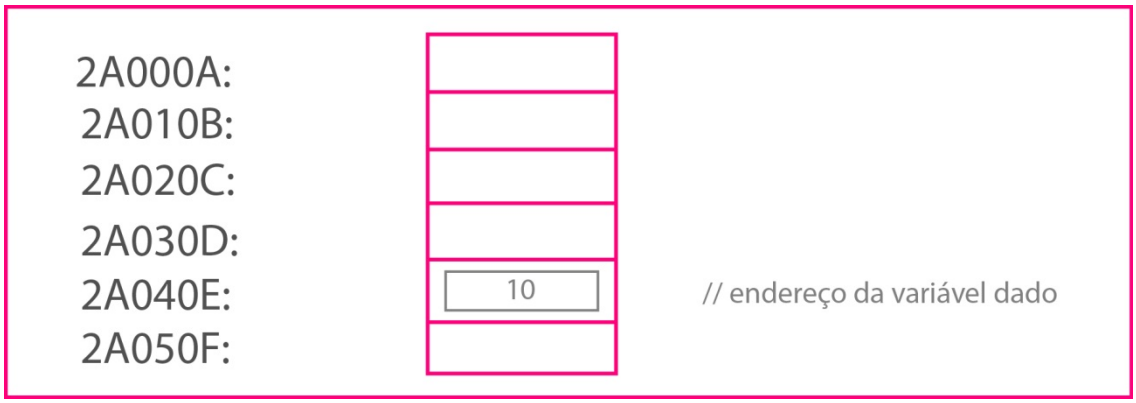
Exemplo:

```
int dado; /* O compilador associa “dado” com um endereço de
             memória (2A040E, por exemplo)*/

dado = 10;   // O valor 10 é armazenado no endereço de dado (2A040E)

printf(“\nDado=%i”,dado); /* O valor armazenado no endereço de
                             dado é impresso na tela */
```

Agora com ponteiros.

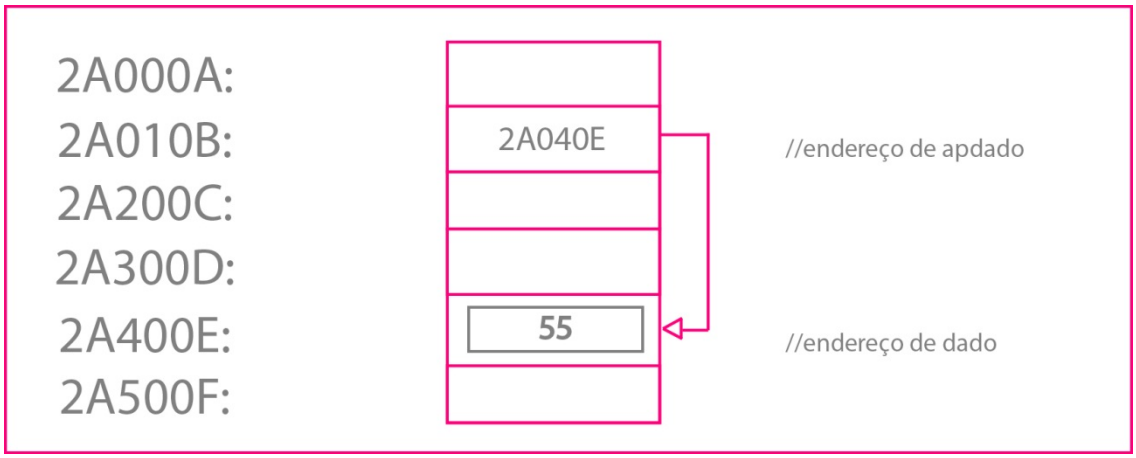


Exemplo

```
int dado = 55, *apdado;

apdado=&dado;  /* Atribui a apdado o endereço de memória que
      contém a variável dado: diz-se que apdado aponta para dado */

printf("\nDado=%i Endereco de dado = %p",dado, apdado);
        /* O valor armazenado no endereço de dado é impresso e seu
endereço de memória, que está em apdado também é impresso */
```



Como igualamos o endereço de dado ao ponteiro apdado, podemos alterar o valor de dado através de apdado. Veja como podemos fazer isso:

```
*apdado = 100;
```

Entender isso é muito simples. O operador *** indica o conteúdo de memória de**. Assim, o comando `*apdado=100;` pode ser entendido como: coloque o número 100 no conteúdo de memória de `apdado`. Como o conteúdo de memória de `apdado` é o endereço de `dado`, o valor 100 é colocado na variável `dado`!

Agora acesse o AVA e assista a animação para consolidar estas informações. Esta animação faz parte da sequência desta aula e, portanto, é essencial para a sua aprendizagem.

Outras informações importantes sobre ponteiros

- Os ponteiros apontam para o byte de posição mais baixa de uma variável. Esta informação é importante, porque conforme o tipo, a variável pode ter 1, 2, 4, 6, 8 ou mais bytes.
- O operador **&** só se aplica a variáveis e a elementos de vetor.
- O operador **&** não pode ser aplicado a expressões, constantes ou variáveis do tipo **register**.
- Para apresentarmos um endereço de memória, utilizamos o formato **%p**

Agora, caro aluno, vamos praticar resolvendo os exercícios propostos. **Leia a lista, resolva os exercícios e verifique seu conhecimento**. Caso fique alguma dúvida, leve a questão ao fórum e divida com seus colegas e professor.

REFERÊNCIAS

UNIVERSIDADE FEDERAL DE MINAS GERAIS. *Curso de C: como funcionam os ponteiros*. 2005. Disponível em: <http://www.ead.cpdee.ufmg.br/cursos/C/aulas/c610.html>. Acesso em: 01 mai. 2012.

SCHILDT, H. *C Completo e total*. 3. ed. São Paulo: Makron Books, 1997.