Sistemas Operacionais I

Revisão Linguagem C
Ponteiros

1

Agenda

- Considerações sobre a memória
- Variáveis versus memória
- Ponteiros
- Aritmética de ponteiros
- Utilização de Ponteiros: Alocação Dinâmica e Passagem por referência
- Exercícios

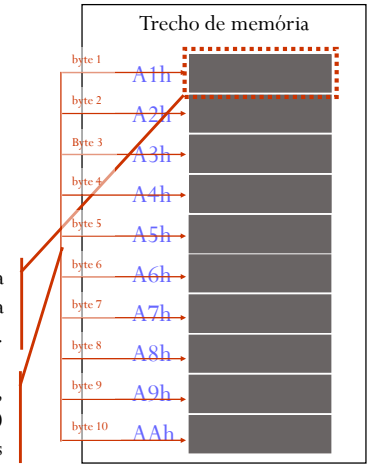
2

Considerações sobre a memória

- O computador possui uma grande capacidade de memória RAM, que o programador pode utilizar em seus programas.
- Por ser muito grande, vamos utilizar um pedaço, um trecho, de memória como exemplo.

A memória do computador é dividida em bytes. Cada pedaço laranja representa 1 byte.

Portanto, neste trecho de memória, pego como exemplo, nós temos 10 bytes

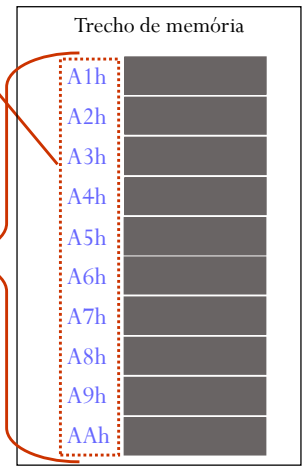


3

Considerações sobre a memória

Esses números que aparecem ao lado de cada byte representam seu endereço (sua posição) de memória *(note que eles estão em hexadecimal)*

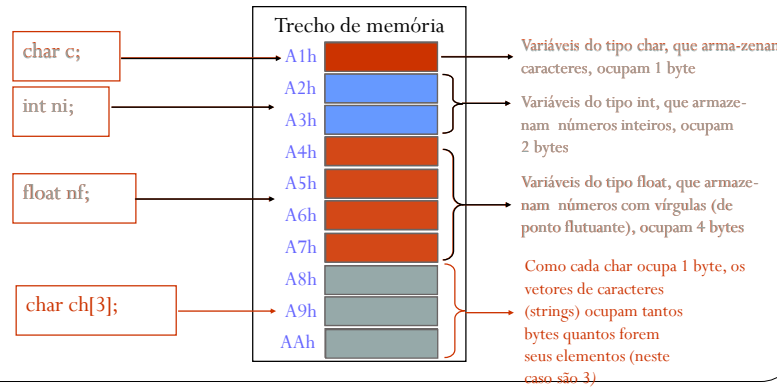
É nestes locais (bytes) que o computador vai armazenar o código e as variáveis dos programas que você fizer



4

Variáveis versus memória

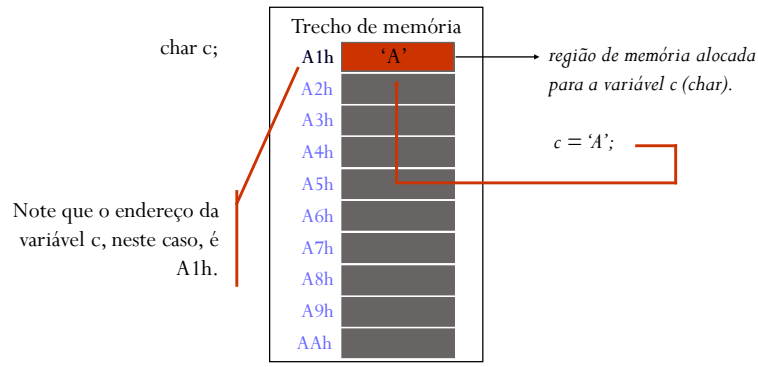
- Quando você declara uma variável, o compilador reserva uma região de memória para ela.
- Essa região de memória é reservada de acordo com o tamanho do tipo de dado para o qual a variável foi declarada.



5

Variáveis versus memória

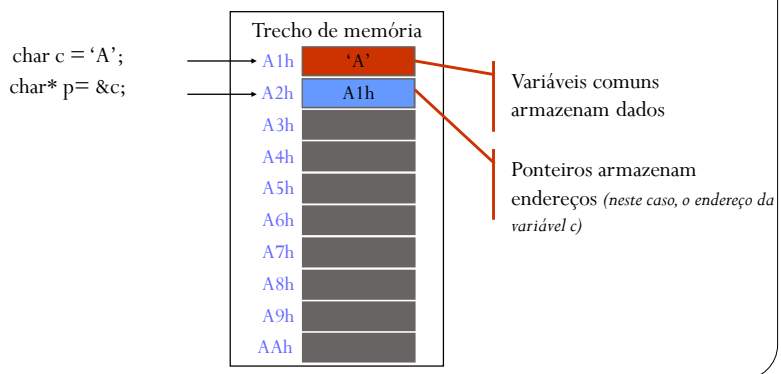
- Portanto, cada variável fica em um endereço de memória diferente;
- Quando fazemos uma atribuição a uma variável, o valor atribuído é colocado na região de memória que foi reservada para ela.



6

Ponteiros

- Sendo assim, nós podemos manipular o conteúdo de uma variável utilizando ela mesmo ou utilizando o seu endereço.
- A linguagem C oferece um tipo especial de variável chamado *ponteiro*. Os ponteiros são variáveis que armazenam endereços de outras variáveis.



7

Declarando ponteiros

- Os ponteiros são iguais às variáveis comuns. A única diferença está no fato dele armazenar um endereço, e não um dado ou valor.
- Para informar que você quer declarar um ponteiro, e não uma variável comum, coloque o símbolo de asterisco ao lado do tipo da variável:

char* ponteiro;

- O símbolo de asterisco é que vai indicar ao c que você quer um ponteiro e não uma variável comum. Fora esse detalhe, a declaração é idêntica a de uma variável comum.

8

Declarando ponteiros

- Para cada tipo de variável há um ponteiro específico.
- Isso significa que se você vai armazenar o endereço de uma variável do tipo int, deve criar um ponteiro para int (que é int*). Se for char, um ponteiro para char (char*).
  - char* pc;
- No exemplo acima, o C vai saber que você está criando uma variável chamada pc que vai armazenar endereços de variáveis do tipo char.
  - int* pn;
- No exemplo acima, o C vai saber que você está criando uma variável chamada pn que vai armazenar endereços de variáveis do tipo int.

9

Declarando ponteiros

- Alguns exemplos:

Tipo	Variável comum	Ponteiro p/o tipo
char	char letra;	char* pontLetra;
int	int numero;	int* pontNumero;
float	float num;	float* pontNum;

10

Utilizando ponteiros

- Utilizar um ponteiro é simples. Como qualquer variável, para colocar um valor dentro dela, basta atribuí-lo:

```
char* ponteiro = 10030;   // coloca o endereço 10030 no ponteiro
```

- Porém, os ponteiros são mais utilizados para armazenar o endereço de outras variáveis:

```
char c = 'A';              // coloca o valor 'A' na variável c
char* ponteiro = &c;       // coloca o endereço da variável c no ponteiro
```

- Note que o operador & (que retorna o endereço de uma variável) foi utilizado!

OBS: A princípio, você não pode atribuir endereços de variáveis que não sejam do mesmo tipo do ponteiro:

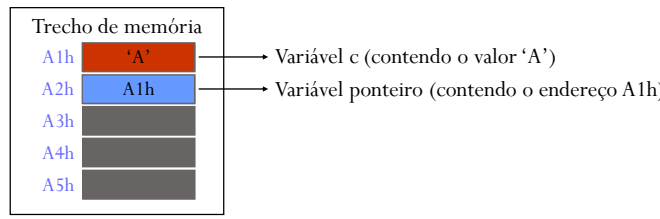
```
int a = 20;
char* pont = &a; // Errado, pois a não é do tipo char!
```

11

Utilizando ponteiros

- Se você mandar imprimir uma variável do tipo ponteiro, ela vai mostrar o endereço armazenado nela:

```
char c = 'A';              // coloca o valor 'A' na variável c
char* ponteiro = &c;       // coloca o endereço da variável c no ponteiro
printf("%d", ponteiro);    // imprime o conteúdo do ponteiro, que é o
                           // endereço da variável c
```

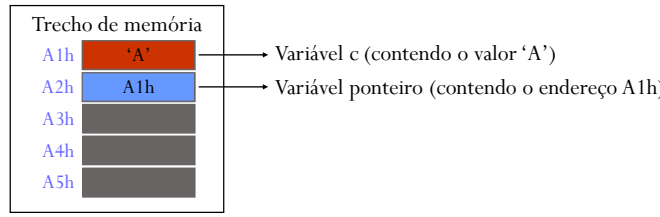


12

Utilizando ponteiros

- Você pode utilizar o ponteiro para mostrar ou manipular o conteúdo do endereço (da variável) que ele aponta.
- Para fazer isso, você deve utilizar o operador * :

```
char c = 'A';              // coloca o valor 'A' na variável c
char* ponteiro = &c;       // coloca o endereço da variável c no ponteiro
printf("%c", *ponteiro);   // imprime o conteúdo do endereço para o qual
                           // o ponteiro aponta ( que é 'A' )
```

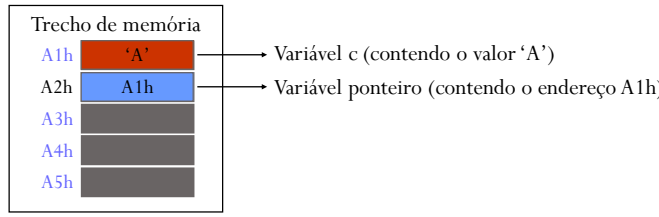


13

Utilizando ponteiros

- Ponteiros também têm endereço. Logo, você também pode imprimir o seu endereço ou armazená-lo ou utilizá-lo em um outro ponteiro:

```
char c = 'A';              // coloca o valor 'A' na variável c
char* ponteiro = &c;       // coloca o endereço da variável c no ponteiro
printf("%d", &ponteiro);   // imprime o endereço do ponteiro (A2h)
char** p2 = &ponteiro;     // cria um ponteiro de ponteiro, e coloca o
                           // endereço do ponteiro lá
```

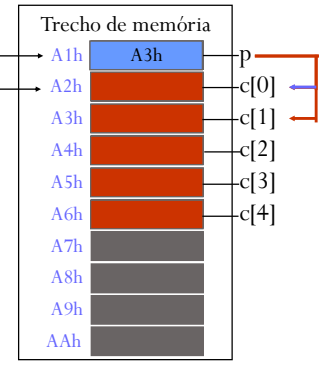


14

Aritmética de ponteiros

- Assim como nas variáveis comuns, você pode realizar operações matemáticas com os endereços armazenados em ponteiros:

```
char* p;
char c[5];
p = &c[0]; // p recebe o endereço do 1º c
```



- Ao incrementar o p, a linguagem C identifica primeiramente qual é o tipo do ponteiro. Daí, incrementa o endereço que ele contém, de acordo com o tamanho do seu tipo.

```
p++;
```

15

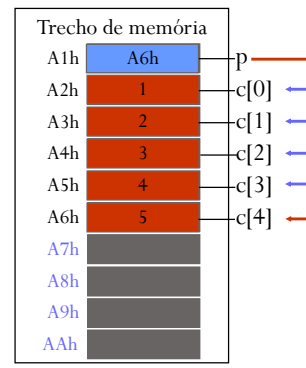
Aritmética de ponteiros

- Deste modo, você pode fazer um programa que preenche o vetor sem ter que utilizar a variável do vetor (variável c):

```
void main(void) {
 char* p;
 char c[5];
 p = c;  // o mesmo que p = &c[0]
 for(int cont=0;cont<5;cont++) {
   *p = cont+1; // coloca cont+1 no local apontado por p
    p++;  // passa para o endereço do próximo elemento
 }
}
```

OBS: o nome do vetor (sem o índice) contém o endereço do primeiro elemento dele!

por isso que no scanf("%s", ...) não vai o &

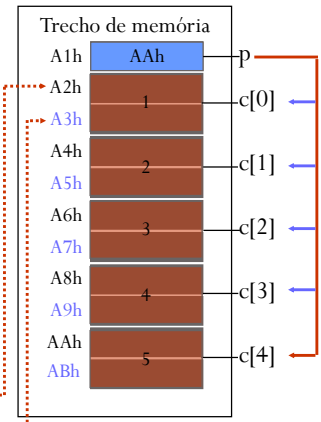


16

Aritmética de ponteiros

- Se a variável c fosse um vetor de inteiros, o p seria incrementado de dois em dois bytes:

```
void main(void) {
 int* p;
 int c[5];
 p = c;  // o mesmo que p = &c[0]
 for(int cont=0;cont<5;cont++) {
   *p = cont+1; // coloca cont+1 no local apontado por p
    p++;  // passa para o endereço do próximo elemento
 }
}
```



- É por esse motivo que os ponteiros devem apontar para variáveis que sejam de um tipo igual ao seu. Caso contrário, ao serem incrementados ou decrementados, podem acabar apontando para o local errado:

```
char* p2 = &c[0];
p2++;
```

17

Resumo

- Operadores de ponteiros:

Operador	Uso	Exemplo
*	Declaração de ponteiro	char* p;
&	Retorna o endereço	p = &variavel;
*	Utilização de conteúdo	char c = *p;
++	Incrementar o endereço *(aponta para o próximo elemento)*	p++;
--	Decrementar o endereço *(aponta para o elemento anterior)*	p--;

18

Ponteiros – Principais Usos

- Implementação de estruturas de dados dinâmicas (usando alocação dinâmico e ponteiros)
- Em funções para implementar passagem de parâmetros por referência

19

Alocação Dinâmica

- Existem 2 métodos através dos quais um programa em C pode armazenar informações na memória principal
  - 1 - através de variáveis locais e globais
  - 2 - através de funções de alocação dinâmica de memória: malloc() e free() - nesse método o programa aloca armazenamento para informações da área de memória livre (heap)

20

Alocação Dinâmica

- Função malloc() - aloca uma parte do restante de memória livre, serve para alocação de memória com finalidades gerais. Sintaxe:
- void *malloc (int número_de_bytes)
- retorna um ponteiro do tipo void que significa que deve ser usado um tipo cast quando atribuir o ponteiro devolvido por malloc() a um ponteiro desejado
- malloc() devolve um ponteiro para o primeiro byte da região de memória que foi alocada na área de alocação dinâmica
- se não houver memória suficiente, malloc devolve um tipo nulo
- pode usar sizeof

21

Alocação Dinâmica

- Função free() - devolve ao sistema memória previamente alocada. Sintaxe:
- free (void *p);
- free nunca deve ser chamado com um argumento inválido, pois será destruída a lista livre

22

Alocação Dinâmica

- Exemplo uso de alocação dinâmica

```
main()
{
    int *p, t;

    p = (int *)malloc(40*sizeof(int));
    if (!p)
            printf ("memória insuficientes\n");
    else {
            for (t=0;t<40;++t) *(p+t)=t;
            for (t=0;t<40;++t) printf("%d ",*(p+t));
    }
}
```

- Lembre-se: antes de usar o ponteiro que malloc devolve, assegura-se que o pedido foi bem sucedido testando o valor devolvido!!!!!

23

Ponteiros para estruturas

- Como outros tipos de dados, ponteiros para estruturas são declarados colocando-se o operador * na frente do nome da variável estrutura:

```
struct data *ap_ontem;

tipoData *ap_amanha;
```

24

Ponteiros para estruturas

- Para acessar uma estrutura com ponteiros pode-se usar duas sintaxes:
- Operador ponto:

```
*<ponteiro_estrutura>.<campo>
```

- Operador seta:

```
<ponteiro_estrutura>-><campo>
```

25

Passagem de Parâmetros em C (Função)

- Na linguagem C não existe passagem por referência
- Usamos ponteiros para implementar passagem por referência
- Vetores e matrizes são passados usando ponteiros (por referência)
- POR VALOR
  - Uma cópia do valor do argumento é passado para a função chamada
  - Modificações de valor feitas dentro da função chamada não modificam a variável original na função que chamou
- POR REFERÊNCIA
  - É passado para a função chamada o endereço do parâmetro
  - A função chamada pode através do endereço de memória modificar o valor do argumento da função que chamou
  - Passagem por referência utiliza PONTEIROS

26

Passagem de Parâmetros em C (Função)

- Exemplo: função que realiza troca de valores entre duas variáveis passadas por parâmetro

```
void troca(int *x, int *y)
{
 int aux;
 aux=*x;
 *x=*y;
 *y=aux;
}
```

```
void main(void)
{
 int a=10,b=20;
 troca(&a,&b);
 printf("%d %d",a,b);
}
```

27

Exercícios

- Seja o seguinte trecho de programa:

```
int i=3,j=5;
int *p, *q;
p = &i;
q = &j;
```

- Qual é o valor das seguintes expressões ?

a) p == &i; b) *p - *q c) **&p d) 3* - *p/(*q)+7

- Diga qual é a saída do seguinte programa:

```
int *p;
p = (int *) malloc (sizeof(int)); ou (int *) malloc (1);
scanf ("%d", p);
*p = *p + 142;
printf ("%d", *p);
free (p);
```

28

Exercícios

- Implementar um programa em C que faça a alocação de espaço em memória para 10 inteiros. O programa deverá imprimir os seus respectivos endereço de memórias e o seu conteúdo.
- Utilizando estrutura, fazer um programa em C que permita a entrada de nome, endereço e telefone de 5 pessoas e os imprima em ordem alfabética. Utilizar na solução alocação dinâmica e ponteiros.
- Implementar um programa em C que cria dois vetores de inteiros e passa para um função que realiza a troca dos elementos dos dois vetores.
- Faça uma função que receba um valor inteiro como referência e retorne o resto da divisão deste número por 10. Altere também o valor da variável passada por referência, dividindo-a por 10.

29