ALGORITMOS E ESTRUTURAS DE DADOS I

Prof. Caio César de Freitas Dantas

Ponteiros

- Ponteiros são um dos recursos mais poderosos da linguagem C.
- A linguagem C é altamente dependente dos ponteiros.
- Ponteiros são tão importantes na linguagem C que você já os viu e nem percebeu, pois mesmo para se fazer um introdução básica à linguagem C precisa-se deles.

Ponteiros

Organizando as informações com variáveis.

- Na linguagem C, cada vez que é declarada uma variável (por exemplo, int x) é associado a esta variável um número hexadecimal (por exemplo, 0022FF77) que é denominado de endereço.

- Esse número realiza a associação entre o nome da variável com o espaço físico que ela ocupa na memória do computador.

- Sem o uso de endereços não seria possível distinguir ou recuperar informações armazenadas na memória do computador.

Ponteiros

Como Funcionam os Ponteiros

- Quando escrevemos: int a; declaramos uma variável com nome a que pode armazenar valores inteiros. Automaticamente, é reservado um espaço na memória suficiente para armazenar valores inteiros (geralmente 4 bytes).

- Da mesma forma que declaramos variáveis para armazenar inteiros, podemos declarar variáveis que, em vez de servirem para armazenar valores inteiros, servem para armazenar endereços de memória onde há variáveis inteiras armazenadas.

- Ponteiros são variáveis que guardam endereços de memória.

Ponteiros

- Um ponteiro nada mais é que uma variável capaz de armazenar um número hexadecimal que corresponde a um endereço de memória de outra variável.

- É importante lembrar que a declaração de uma variável do tipo caractere, por exemplo, implica em reservar um espaço de memória para armazenar a informação correspondente.

- Para atribuir e acessar endereços de memória, a linguagem utiliza dois operadores unários: o & e o *.

Ponteiros

Por que usar ponteiros?

- Ponteiros são muito úteis quando uma variável tem que ser acessada em diferentes partes de um programa.
- Neste caso, o código pode ter vários ponteiros espalhados por diversas partes do programa, “apontando” para a variável que contém o dado desejado.
- Caso este dado seja alterado, não há problema algum, pois todas as partes do programa têm um ponteiro que aponta para o endereço onde reside o dado atualizado.

Ponteiros

Alguns motivos para se usar ponteiros:

- Passagem de parâmetros para uma função por referência. Passamos o endereço (um ponteiro) da variável argumento;
- Forma elegante de passar matrizes e strings como argumentos para funções;
- A utilização sensata de ponteiros deixa o programa mais rápido;
- Ponteiros são a base para a criação de estruturas de dados mais avançadas, como listas, pilhas, filas, árvores..

Ponteiros - Funcionamento

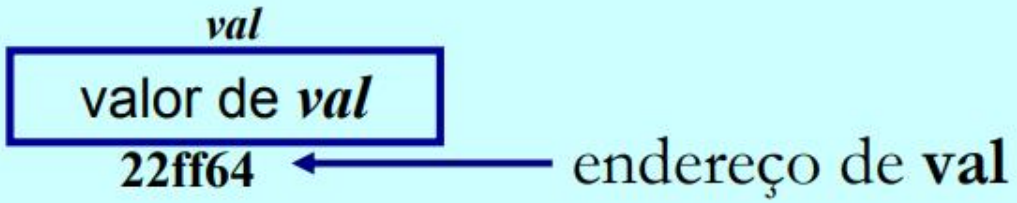
- O operador unário & (“endereço de”), aplicado a variáveis, resulta no endereço da posição da memória reservada para a variável.
- O operador unário * (“conteúdo de”), aplicado a variáveis do tipo ponteiro, acessa o conteúdo do endereço de memória armazenado pela variável ponteiro.
- Suponha por exemplo que a variável val, armazene dentro dela um valor inteiro.
- Quando ela é declarada, o compilador reservará automaticamente um espaço na memória (caixinha) para ela.

Ponteiros - Funcionamento

- O conteúdo desta variável pode ser acessado através do nome val , mas este conteúdo também pode ser acessado através do seu endereço na memória.

- O endereço da localização de memória da variável val pode ser determinado pela expressão &val.

Ex: **int** val;



- Quando à variável val é declarada, o que ocorre, na verdade, é que o espaço de memória, cujo endereço é 22ff64, fica disponível para ocupar um inteiro.

  PS: Observe que o endereço da variável é um números hexadecimal.

Ponteiros - Funcionamento

- Vamos agora atribuir o endereço da variável val a uma outra variável, pval. Então:

  pval = &val;

- Essa nova variável é chamada de ponteiro para val , pois ela aponta para a localização onde val está armazenada na memória.

- Lembre-se que pval armazena o endereço de val e não o seu valor

Ponteiros - Funcionamento

- Assim como as demais variáveis, um ponteiro também tem tipo.

- No C quando declaramos ponteiros nós informamos ao compilador para que tipo de variável vamos apontá-lo. Um ponteiro int aponta para um inteiro, isto é, guarda o endereço de um inteiro.

- Os ponteiros são utilizados para alocação dinâmica, e podem substituir matrizes com mais eficiência. Eles também fornecem um modo de se passar informações entre uma função e seu ponto de referência. Eles fornecem uma maneira de retornar múltiplos dados de uma função através dos parâmetros.

Ponteiros - Funcionamento

- O ponteiro deve ser inicializado (apontado para algum lugar conhecido) antes de ser usado! Isto é de suma importância!

- Para atribuir um valor a um ponteiro recém-criado poderíamos igualá-lo a um valor de memória.

Mas, como saber a posição na memória de uma variável do nosso programa?

Ponteiros - Funcionamento

- Seria muito difícil saber o endereço de cada variável que usamos, mesmo porque estes endereços são determinados pelo compilador na hora da compilação e realocados na execução.
- Podemos então deixar que o compilador faça este trabalho por nós.
- Para saber o endereço de uma variável basta usar o operador &.
- O ponteiro também pode ser inicializado com o valor nulo, para tanto basta fazermos: int *p=NULL;

Ponteiros – Declarando e Utilizando

- Para declarar um ponteiro temos a seguinte forma geral:

```
tipo_do_ponteiro *nome_da_variável;
```

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

Ponteiros – Declarando e Utilizando

- Para declarar um ponteiro temos a seguinte forma geral:

tipo_do_ponteiro *nome_da_variável;

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

Exemplos de declarações:
int *pt;
char *temp,*pt2;

O primeiro exemplo declara um ponteiro para um inteiro.
O segundo declara dois ponteiros para caracteres.
Eles ainda não foram inicializados (como toda variável do C que é apenas declarada).
Isto significa que eles apontam para um lugar indefinido.

Ponteiros – Declarando e Utilizando

Veja o exemplo:

```
int x=10;
int *pt1, *pt2=NULL;
pt1=&x;
```

- Criamos um inteiro x com o valor 10 e dois apontadores para um inteiro pt1 e pt2. A expressão &x nos dá o endereço de x, o qual armazenamos em pt1. pt2 foi inicializado com nulo.
- Repare que não alteramos o valor de x, que continua valendo 10.

- Como foi colocado um endereço em pt1 e pt2, ele está agora "liberado" para ser usado. Pode-se, por exemplo, alterar o valor de x usando pt1. Vamos usar o operador "inverso" do operador &. É o operador *.

- No exemplo acima, uma vez que fizemos pt1=&x a expressão *pt1 é equivalente ao próprio x. Isto significa que, para se mudar o valor de x para 12, basta fazer *pt1=12.

Ponteiros – Declarando e Utilizando

```
#include <stdlib.h>
#include <stdio.h>
main()
{ int *px1;
float *px2;
char *px4;
double *px3;
int i=1;
float f= 0.3;
double d= 0.005;
char c = '*';
px1=&i;
px2=&f;
px3=&d;
px4=&c;
```

Ponteiros – Declarando e Utilizando

```
printf("valores:\t\t i=%d \t f=%f \t d=%f \t c=%c \n\n", i, f, d, c);
printf("Enderecos:\t\t &i=%x &f=%x &d=%x &c=%x\n\n", &i, &f, &d, &c);
printf("Valores dos ponteiros: px1= %x px2=%x px3=%x\ px4=%x\n\n", px1, px2, px3, px4);
system("PAUSE");
}
```

```
#include <stdlib.h>
#include <stdio.h>
main()
{ int *px1;
float *px2;
char *px4;
double *px3;
int i=1;
float f= 0.3;
double d= 0.005;
char c = '*';
px1=&i;
px2=&f;
px3=&d;
px4=&c;
```

Ponteiros – Declarando e Utilizando

```
valores:                    i=1     f=0.300000      d=0.005000      c=*

Enderecos:                  &i=22ff64   &f=22ff60   &d=22ff58    &c=22ff57

Valores dos ponteiros: px1= 22ff64  px2=22ff60  px3=22ff58  px4=22ff57

Pressione qualquer tecla para continuar. . .
```

- Esse programa exibe valores e endereços associados a quatro tipos de variáveis: variável inteira i, variável real f, variável de dupla precisão d, e uma variável caractere c.
- O programa também utiliza px1, px2, px3 e px4, que representam respectivamente os endereços de i, f,d, c.
- A primeira linha exibe os valores das variáveis. A segunda linha , seus endereços conforme assinalado pelo compilador, e a terceira linha exibe o endereço representado pelas expressões ponteiros.

Ponteiros – Declarando e Utilizando

- Exemplo 2

```
#include <stdlib.h>
#include <stdio.h>
main()
{int *p, i =4;
  p = &i;
  printf("  i = %d \n",i);
  printf(" &i = %x \n",&i);
  printf("  p = %p \n",p);
  printf(" &p = %X \n",&p);
  printf("\n Conteudo do endereco apontado por p: ");
  printf("%d\n",*p; //conteúdo do endereço apontado por p
  printf("\n\n");
  system("PAUSE");
}
```

Ponteiros – Declarando e Utilizando

- Exemplo 2

```
 i = 4
&i = 22ff70
 p = 0022FF70
&p = 22FF74

Conteudo do endereco apontado por p:  4


Pressione qualquer tecla para continuar. . .
```

- Quando queremos acessar o conteúdo do endereço para o qual o ponteiro aponta fazemos *p. Isto é, usamos o operador "inverso" do operador & que é o operador *.

Ponteiros – Declarando e Utilizando

- Exemplo 3

```
#include <stdlib.h>
#include <stdio.h>
main ()
  { int num,valor;
   int *p;
   num=55;
   p=&num; // Pega o endereço de num
   valor=*p;//valor é igualado a num de maneira
            //indireta
   printf ("\n\n%d\n",valor);
   printf ("Endereco para onde o ponteiro aponta:
%d\n",p);
   printf ("Valor da variavel apontada: %d\n",*p);
   system("pause");
  }
```

Ponteiros – Declarando e Utilizando

- Exemplo 3

```
55
Endereco para onde o ponteiro aponta: 2293620
Valor da variavel apontada: 55
Pressione qualquer tecla para continuar. . .
```

- A variável valor recebeu o valor 55 através do ponteiro p (endereço da variável num) e do operador *. Isto é, através da linha de comando: valor=*p.

Ponteiros – Declarando e Utilizando

- Exemplo 4

```
#include <stdlib.h>
#include <stdio.h>
# include<conio.h>

int main ()
{
int num,*p;
num=55;
p=&num; /* Pega o endereco de num */
printf ("\nValor inicial: %d\n",num);
*p=100;// Muda o valor de num de uma maneira
//indireta
printf ("\nValor final: %d\n",num);
getch();//pára a tela de execução. É usado com
a biblioteca conio.h
}
```

```
Valor inicial: 55

Valor final: 100
```

Ponteiros – Declarando e Utilizando

Resumindo:

- Operador (*) é chamado de Operador de referência.
- Na linguagem C, o operador de referência (*) além de servir para declarar que um variável é do tipo ponteiro serve também para saber qual o valor contido no endereço armazenado (apontado) por uma variável do tipo ponteiro.

Ponteiros – Operações Aritméticas

- Podemos fazer algumas operações aritméticas com ponteiros.
- A operação mais simples, é igualar dois ponteiros. Se temos dois ponteiros p1 e p2 podemos igualá-los fazendo p1=p2.
- Observe que estamos fazendo p1 apontar para o mesmo lugar que p2.
- Se quisermos que a variável apontada por p1 tenha o mesmo conteúdo da variável apontada por p2 devemos fazer *p1=*p2.
- Depois que se aprende a usar os dois operadores (& e *) fica fácil entender operações com ponteiros.

Ponteiros – Operações Aritméticas

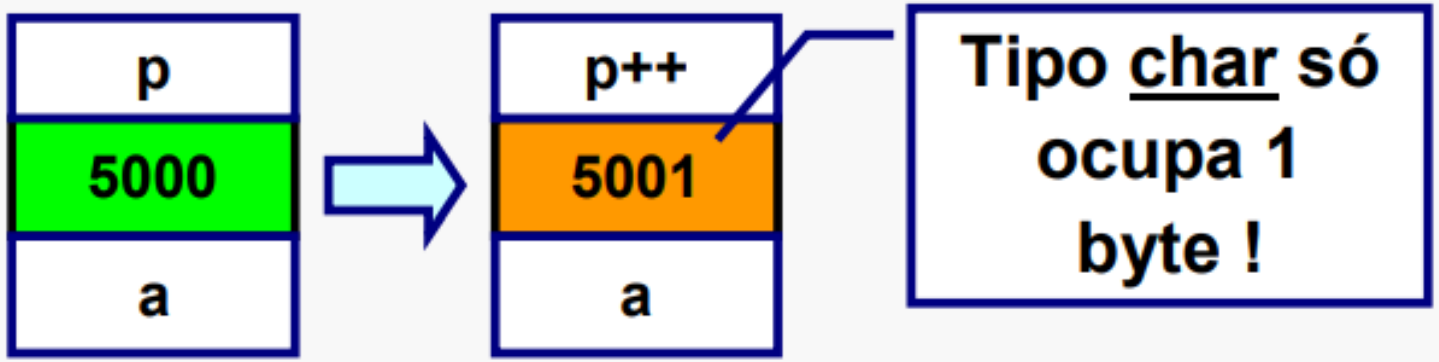
- Um tipo de operação muito usada, é o incremento e o decremento (por exemplo: p+1 e p-1).
- Quando incrementamos um ponteiro p ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta. Isto é, se temos um ponteiro para um inteiro e o incrementamos ele passa a apontar para o próximo inteiro.
- Esta é mais uma razão pela qual o compilador precisa saber o tipo de um ponteiro: se você incrementa um ponteiro *char ele anda 1 byte na memória e se você incrementa um ponteiro *double ele anda 8 bytes na memória.
- O decremento funciona semelhantemente. Supondo que p é um ponteiro, as operações podem ser escritas como:

```
p++;    p--;
```

Ponteiros – Operações Aritméticas

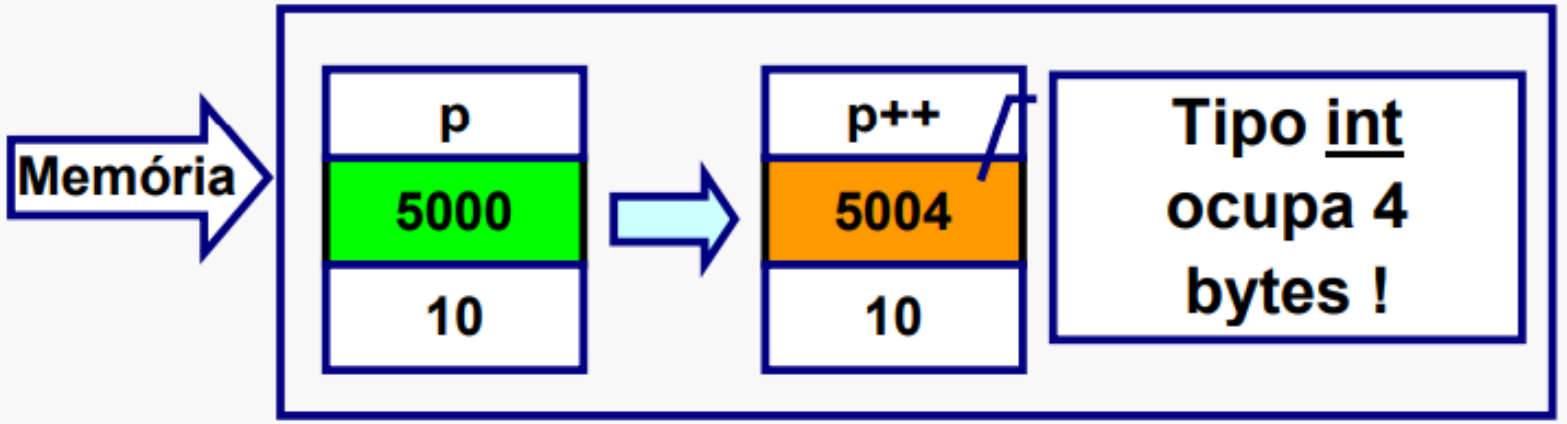
- Uma variável do tipo ponteiro está sempre associada a um tipo. Ou seja, um ponteiro para um dado tipo t endereça o número de bytes que esse tipo t ocupa em memória

```
char ch, *p;
ch = 'a';
p = &ch;
p++;
```



Ponteiros – Operações Aritméticas

```
int x, *p;
x = 10;
p = &x;
p++;
```



Espaço de 4 bytes para armazenar a variável x tipo int.

A operação p++ percorre sizeof(tipo p) bytes !

p				p++			
x	x	x	x	?	?	?	?
5000	5001	5002	5003	5004	5005	5006	5007

Ponteiros – Declarando e Utilizando

- Exemplo 4

```
main()
{ int *px1;
 float *px2;
 char *px4;
 double *px3;
 int i=1;
 float f= 0.3;
 double d= 0.005;
 char c = '*';
 px1=&i;
 px2=&f;
 px3=&d;
 px4=&c;
```

```
printf("Valores ponteiros antes ++:px1=%d px2=%d
     px3=%d px4=%d  \n\n",px1, px2,px3,px4);
```

Ponteiros – Declarando e Utilizando

- Exemplo 4

```
Valores ponteiros antes ++:  px1=2293604  px2=2293600  px3=2293592  px4=2293591
```

```
px1++;
px2++;
px3++;
px4++;
```

```
printf("Valores dos ponteiros depois ++:px1=%d
        px2=%d px3=%d  px4=%d\n\n",px1,px2,px3,px4);
```

```
Valores ponteiros depois ++:px1=2293608  px2=2293604  px3=2293600  px4=2293592
```

Ponteiros – Operações Aritméticas

- Estamos falando **de operações com ponteiros e não de operações com o conteúdo das variáveis** para as quais eles apontam. Por exemplo, para incrementar o conteúdo da variável apontada pelo ponteiro p, faz-se: (*p)++;

- Outras operações aritméticas úteis são a soma e subtração de inteiros com ponteiros. Suponha que você queira incrementar um ponteiro de 15. Basta fazer: p=p+15; ou p+=15;

- Se você quiser usar o conteúdo do ponteiro 15 posições adiante basta fazer *(p+15);

- A subtração funciona da mesma maneira que a adição.

Ponteiros – Operações Aritméticas

- Um programa para zerar uma matriz:

```
#include <stdio.h>
int main (){
    float matriz [50][50];
    int i,j;
    for (i=0;i<50;i++)
        for (j=0;j<50;j++)
            matriz[i][j]=0.0;
    return(0);
}
```

Ponteiros – Operações Aritméticas

- Usando ponteiros:

```c
#include <stdio.h>
int main (){
    float matriz [50][50];
    float *p;
    int count;
    p=matriz[0];
    for (count=0;count<2500;count++){
        *p=0.0;
        p++;
    }
    return(0);
}
```

Ponteiros – Operações Aritméticas

- Usando ponteiros:

```c
#include <stdio.h>
int main (){
    float matriz [50][50];
    int i,j;
    for (i=0;i<50;i++)
        for (j=0;j<50;j++)
            matriz[i][j]=0.0;
    return(0);
}
```

```c
#include <stdio.h>
int main (){
    float matriz [50][50];
    float *p;
    int count;
    p=matriz[0];
    for (count=0;count<2500;count++){
        *p=0.0;
        p++;
    }
    return(0);
}
```

FIM!