



Aula 08

Tecnologia em Análise e
Desenvolvimento de
Sistemas

SPOLOG1
Lógica de Programação I

1º Semestre

Profa. Me. Eurides Balbino
eubalbino@ifsp.edu.br

Introdução ao uso de ponteiros

Ponteiro e alocação de memória

Conjunto da obra

Exercícios

Ponteiros e alocação de memória

Na Linguagem C existe uma distinção bastante explícita entre um tipo (ou uma estrutura) e um endereço:

```
int x;      /* significa que x é uma variável do tipo inteiro */  
int * y;    /* significa que y é uma variável do tipo endereço que aponta para inteiros */
```

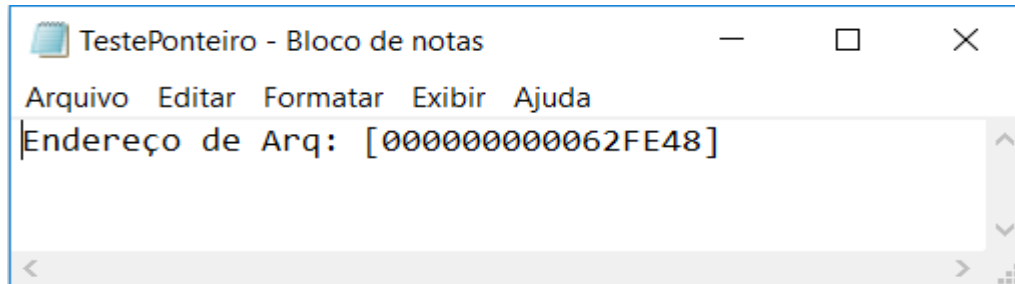
O asterisco (*) após a palavra *int* indica que estamos falando de um endereço para inteiro e não mais de um inteiro.

Ponteiros e alocação de memória

Vimos ponteiros quando manipulamos arquivos:

```
FILE * Arq; /* a variável Arq trata-se de um ponteiro (apontador) */  
        /* para endereços de uma estrutura FILE */
```

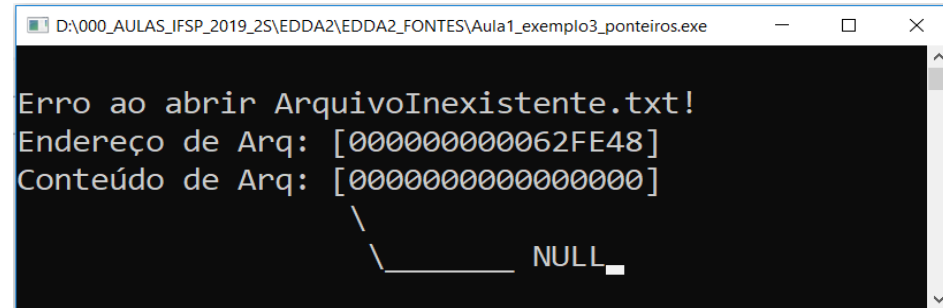
```
1  /* Bibliotecas */  
2  #include <stdio.h>  
3  #include <locale.h>  
4  #include <stdlib.h>  
5  /* corpo do programa */  
6  int main()  
7  {  
8      /* Variáveis locais ao main */  
9      FILE * Arq; /* a variável Arq trata-se de um ponteiro (apontador) */  
10     /* para endereços de uma estrutura FILE */  
11     setlocale (LC_ALL, "");  
12     system ("mode 100");  
13  
14     Arq = fopen ("TestePonteiro.txt", "w");  
15     if ( Arq == NULL)  
16     {  
17         printf ("Erro ao abrir TestePonteiro.txt!");  
18         getch();  
19         exit(0);  
20     }  
21     fprintf(Arq, "Endereço de Arq: [%p]", &Arq);  
22     fclose (Arq);  
23     return 0;  
24 }
```



Ponteiros e alocação de memória

Também vimos que, quando o ponteiro aponta para um endereço inexistente, tal endereço é identificado como NULL (0000000000000000).

```
1  /* Bibliotecas */
2  #include <stdio.h>
3  #include <locale.h>
4  #include <stdlib.h>
5  /* corpo do programa */
6  int main()
7  {
8      /* Variáveis locais ao main */
9      FILE * Arq;
10     setlocale (LC_ALL, "");
11     system ("mode 80");
12     Arq = fopen ("ArquivoInexistente.txt", "r");
13     if ( Arq == NULL)
14     {
15         printf ("\nErro ao abrir ArquivoInexistente.txt!");
16         printf ("\nEndereço de Arq: [%p]", &Arq);
17         printf ("\nConteúdo de Arq: [%p]", Arq);
18         printf ("\n                \\");
19         printf ("\n                \\_____ NULL");
20         getch();
21         exit(0);
22     }
23     return 0;
24 }
```



```
D:\000_AULAS_JFSP_2019_2S\EDDA2\FONTES\Aula1_exemplo3_ponteiros.exe
Erro ao abrir ArquivoInexistente.txt!
Endereço de Arq: [000000000062FE48]
Conteúdo de Arq: [0000000000000000]
\_____ NULL
```

Ponteiros e alocação de memória

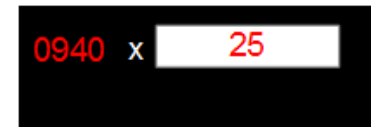
```
#include <stdio.h>

int main()
{
    int    x = 25;
    int  * y = &x;
    *y      = 30;
    printf("Valor atual de x: %i\n", x);
    return 0;
}
```

Ponteiros e alocação de memória

```
#include <stdio.h>

int main()
{
    int    x = 25;
    int * y = &x;
    *y     = 30;
    printf("Valor atual de x: %i\n", x);
    return 0;
}
```



**A variável x é inicializada com valor 25
(e está alocada no endereço de memória
0940).**

Ponteiros e alocação de memória

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

```
int main()
```

```
{
```

```
    int    x = 25;
```

```
    int *   y = &x;
```

```
    *y      = 30;
```

```
    printf("Valor atual de x: %i\n", x);
```

```
    return 0;
```

```
}
```

0940	x	25
0936	y	0940

A variável x é inicializada com valor 25.

A variável y (que está alocada no endereço de memória 0936) recebe o endereço de onde está a variável x.

Ponteiros e alocação de memória

```
#include <stdio.h>

int main()
{
    int    x = 25;
    int *   y = &x;
    *y      = 30;
    printf("Valor atual de x: %i\n", x);
    return 0;
}
```

0940	x	30
0936	y	0940

A variável x é inicializada com valor 25.

A variável y recebe o endereço onde está a variável x.

Coloca-se o valor 30 na posição de memória referenciada (apontada) pelo endereço armazenado em y (o conteúdo do endereço apontado por y (0940) recebe o valor 30).

Ponteiros e alocação de memória

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

```
int main()
```

```
{
```

```
    int    x = 25;
```

```
    int *   y = &x;
```

```
    *y      = 30;
```

```
    printf("Valor atual de x: %i\n", x);
```

```
    return 0;
```

```
}
```

0940	x	30
0936	y	0940

Valor atual de x: 30

A variável x é inicializada com valor 25.

A variável y recebe o endereço onde está a variável x.

Coloca-se o valor 30 na posição de memória referenciada (apontada) pelo endereço armazenado em y.

Ponteiros e alocação de memória

Declaração de ponteiro para int

```
#include <stdio.h>

int main()
{
    int x = 25;
    int *y = &x;
    *y = 30;
    printf("Valor atual de x: %i\n", x);
    return 0;
}
```

A variável x é inicializada com valor 25.

A variável y recebe o endereço onde está a variável x.

Coloca-se o valor 30 na posição de memória referenciada (apontada) pelo endereço armazenado em y.

Ponteiros e alocação de memória

Inicialização do ponteiro (y) com o endereço (&) de memória de um inteiro (x)

```
#include <stdio.h>

int main()
{
    int x = 25;
    int *y = &x;
    *y = 30;
    printf("Valor atual de x: %i\n", x);
    return 0;
}
```

A variável x é inicializada com valor 25.

A variável y recebe o endereço onde está a variável x.

Coloca-se o valor 30 na posição de memória referenciada (apontada) pelo endereço armazenado em y.

Ponteiros e alocação de memória

Atribuição do valor 30 ao conteúdo (*) do ponteiro y
(conteúdo do conteúdo de y)

```
#include <stdio.h>

int main()
{
    int x = 25;
    int *y = &x;
    *y = 30;
    printf("Valor atual de x: %i\n", x);
    return 0;
}
```

A variável x é inicializada com valor 25.

A variável y recebe o endereço onde está a variável x.

Coloca-se o valor 30 na posição de memória referenciada (apontada) pelo endereço armazenado em y.

Conjunto da obra

D:\000_AULAS_IFSP_2019_2S\EDDA2\EDDA2_FONTES\Aula1_exemplo1_ponteiros.c - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

IDM-GCC 4.9.2 64-bit Release

(globals)

[*] Aula1_exemplo1_ponteiros.c Aula1_exemplo2_ponteiros.c Aula1_exemplo3_ponteiros.c

```
1  /* Bibliotecas */
2  #include <stdio.h>
3  #include <locale.h>
4  #include <stdlib.h>
5  /* corpo do programa */
6  int main()
7  {
8      /* Variáveis locais ao main */
9      int x = 25;
10     int *y = &x;
11
12     setlocale (LC_ALL, "");
13     system ("mode 100");
14
15     printf("\nEndereço de x: [%p]\tEndereço de y: [%p]", &x, &y);
16     printf("\nConteúdo de x: [%i]\t\t\tConteúdo de y: [%p]", x, y);
17     *y = 30;
18     printf("\nConteúdo de x: [%i]\t\t\tConteúdo de y: [%p]", x, y);
19     printf("\nConteúdo do conteúdo de y: [%i]", *y);
20     return 0;
21 }
```

Line: 22 Col: 1 Sel: 0

D:\000_AULAS_IFSP_2019_2S\EDDA2\EDDA2_FONTES\Aula1_exemplo1_ponteiros.exe

Endereço de x: [000000000062FE4C]

Conteúdo de x: [25]

Conteúdo de x: [30]

Conteúdo do conteúdo de y: [30]

Endereço de y: [000000000062FE40]

Conteúdo de y: [000000000062FE4C]

Conteúdo de y: [000000000062FE4C]

Conjunto da obra

```
/* Bibliotecas */  
#include <stdio.h>  
#include <conio.h>  
#include <locale.h>  
#include <stdlib.h>
```

```
/* Constantes pré-definidas */  
#define alturaMaxima 2.25
```

```
/* Tipos de dados definidos pelo programador*/  
typedef struct  
{  
    float    peso;    /* peso em quilogramas    */  
    float    altura; /* altura em metros    */  
}  
PesoAltura;
```

Conjunto da obra

```
/* Corpo do programa */
int main()
{
    setlocale (LC_ALL, "");
    system ("mode 80, 6"); /* define tela com 80 colunas e 6 linhas */
    /* Variáveis locais ao main */
    int                x;
    PesoAltura         pessoal;

    /* Define os valores do peso e da altura da pessoal */
    pessoal.peso        = 80;
    pessoal.altura       = 1.85;


    /* Monta o cabeçalho do que será exibido em tela */
    printf("& x\t\t\t& do peso da pessoal\t\t& da altura da pessoal\n");
    /* Exibe os endereços de memória de: x, pessoal e da altura da pessoal*/
    printf("[%p]\t[%p]\t\t[%p]\n", &x, &(pessoal.peso), &(pessoal.altura));
    /* Exibe os conteúdos das variáveis: peso de pessoal e altura de pessoal */
    printf("x: %d\t\t\tPeso: %.2f\t\t\tAltura %.2f. ", x, pessoal.peso, pessoal.altura);

    /* Se a altura da pessoal for maior que a altura máxima pré-definida...*/
    if (pessoal.altura>alturaMaxima)
        /* ... informa que a altura da pessoal está acima da máxima.*/
        printf("\n\t\t\t\t\t\t\t\tAltura acima da máxima.\n");
    else
        /* ... se não... informa que a altura da pessoal está abaixo da máxima.*/
        printf("\n\t\t\t\t\t\t\t\tAltura abaixo da máxima.\n");

    getch();

    return 0;
}
```

Conjunto da obra



```
D:\000_AULAS_IFSP_2019_2S\EDDA2\EDDA2_FONTES\EstruturaSimples.exe
& x                & do peso da pessoa1          & da altura da pessoa1
[000000000062FE4C]  [000000000062FE40]          [000000000062FE44]
x: 0                Peso: 80,00              Altura 1,85.
                   Altura abaixo da máxima.
```


Ponteiros e alocação de memória

Para alocarmos memória dinamicamente na Linguagem C utilizamos a função malloc (*memory allocation*).

Para usarmos a função malloc precisamos saber que:

- Devemos incluir a biblioteca `stdlib.h` em nosso programa;
- Devemos passar para malloc, como parâmetro, o **número de bytes** que se deseja alocar;
- A função malloc retorna o endereço inicial do bloco de bytes que foi alocado, porém esse retorno tem o tipo `void*` (ponteiro para void);
- Existe a função chamada `sizeof` que recebe como parâmetro um tipo (simples ou composto) e retorna a quantidade de bytes ocupada por esse tipo.

Ponteiros e alocação de memória

Exemplificando o uso de malloc:

```
1  /* Bibliotecas */
2  #include <stdio.h>
3  #include <conio.h>
4  #include <locale.h>
5  #include <stdlib.h>
6  /* Tipos de dados definidos pelo programador */
7  typedef struct
8  {
9      int dia, mes, ano;
10 }
11 data;
12 /* Corpo do programa */
13 int main()
14 { /* Variáveis locais ao main */
15     data *d;
16     int qtd_memoria;
17     setlocale (LC_ALL, "");
18     system ("mode 115,6");
19     /* sizeof retornará a quantidade de bytes da estrutura data */
20     qtd_memoria = sizeof (data);
21     /* o ponteiro d apontará para o endereço de onde começa a memória alocada por malloc */
22     d = (data *) malloc (qtd_memoria);
23     /* os conteúdos da memória reservada (d) são preenchidos...*/
24     d->dia = 1;    /*...com dia... */
25     d->mes = 8;    /*...com mês... */
26     d->ano = 2019; /*... e com ano...*/
27     /* Monta cabeçalho para exibição dos dados */
28     printf("& qtd_memoria\t\t& do ponteiro d\t\t& de d->dia\t\t& de d->mes\t\t& de d->ano\n");
29     /* Exibe os endereços de memória de: qtd_memoria, d, d->dia, d->mes e d->ano */
30     printf("[%p]\t[%p]\t[%p]\t[%p]\t[%p]\n", &qtd_memoria, &d, &d->dia, &d->mes, &d->ano);
31     /* Exibe os conteúdos das variáveis: qtd_memoria, d, d->dia, d->mes e d->ano */
32     printf("qtd_memoria=%d\t\t&= %p\t&dia=%d\t\t&mes=%d\t\t&ano=%d. ", qtd_memoria, d, d->dia, d->mes, d->ano);
33     getch();
34     return 0;
35 }
```

Ponteiros e alocação de memória

Exemplificando o uso de malloc:

```
D:\000_AULAS_IFSP_2019_2S\EDDA2\EDDA2_FONTES\Aula1_exemplo5_ponteiros.exe
& qtd_memoria      & do ponteiro d      & de d->dia      & de d->mes      & de d->ano
[000000000062FE44]  [000000000062FE48]  [0000000000AB7100]  [0000000000AB7104]  [0000000000AB7108]
qtd_memoria=12      d=0000000000AB7100  dia=1              mês=8              ano=2019. █
```

Exercícios

1. Discuta, passo a passo, o efeito do seguinte fragmento de código:

```
int *v;  
v = malloc (10 * sizeof (int));
```

2. Elabore um programa em Linguagem C para testar esse fragmento de código identificando os endereços e conteúdos envolvidos.
3. Escreva um programa em Linguagem C que defina uma função que receba um byte c (que pode representar um caractere ASCII, por exemplo) e transforme-o em uma string, ou seja, devolva uma string de comprimento 1 tendo c como único elemento.