Ponteiros e Alocação Dinâmica de Memória

INF01203 – Estruturas de Dados

Como funcionam os ponteiros

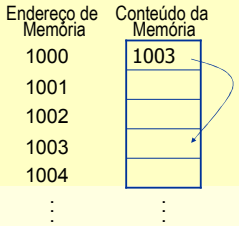
INT guarda inteiros
FLOAT guarda número de ponto flutuante
CHAR guarda caracter

PONTEIROS guardam endereços de memória

- O endereço é a posição de uma outra variável na memória
- Se uma variável *a* contém o endereço de uma variável *b*, dizemos que *a* aponta para *b*

O que são ponteiros?

- Ponteiros são variáveis que guardam endereços de memória.



Declaração de Ponteiros

tipo *nome;

- tipo: qualquer tipo válido em C
- *: operador especial para denotar que a variável armazena um endereço de memória.
- nome: nome da variável ponteiro

Exemplos:

```
float *f; // f é um ponteiro para variáveis do tipo float
int *p;  // p é um ponteiro para variáveis do tipo inteiro
```


Operadores de Ponteiros

& operador unário que devolve o endereço de memória de seu operando

Exemplo:

```
int count, q, *m;
m = &count; //coloca em m o endereço de memória da variável count
```

- Lê-se: "m recebe o endereço de count"
- Se a variável count está armazenada na posição de memória 2000, o valor de m será 2000.

Operadores de Ponteiros

***** operador unário que devolve o valor da variável localizada no endereço que o segue

Exemplo:

```
int count, q, *m;
q = *m; //coloca em q o valor contido no endereço de
        memória apontado por m
```

- Lê-se: "q recebe o valor que está no endereço m"
- Se a variável m aponta para um endereço que contém o valor 100, o valor de q será 100.

Exemplo 1

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

int main()
{
   int count, q, *m;

   count = 100;
   m = &count;
   q = *m;
   printf("count = %d\n", count);
   printf("q = %d\n",q);
   printf("m = %p\n",m); //%p para imprimir ponteiros
   printf("m aponta para = %d\n\n",*m); //conteúdo da memória apontada por m
   system("pause");
}
```

```
C:\Documents and Settings\Viviane\Meus documentos\Teachi
count = 100
q = 100
m = 0022FF74
m aponta para = 100

Pressione qualquer tecla para continuar. . .
```


Exemplo 2 – Onde está o problema?

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

int main()
{
   float x;
   int *p;

   x = 100;
   p = &x;
   printf("x=%f\p=%p\n\n", x,p);
   system("pause");
}
```

Erro de compilação: tipos incompatíveis
p é um ponteiro para **inteiros**, logo não pode apontar para uma variável do **tipo float**.

Atribuições com ponteiros

- Assim como qualquer variável, um ponteiro pode ser usado no lado direito de um comando de atribuição para passar seu valor para um outro ponteiro.

Exemplo:

```
int x, *p1, *p2;
p1 = &x; //p1 aponta para x
p2 = p1; //p2 recebe p1 e tbém passa a apontar para x
printf("%p", p2); //escreve o endereço de x
```


Exemplo 3 – O que será impresso na tela?

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

int main(){
      int a, *x,*y;
      a = 10;
      x = &a;
      y = x;
      printf("a = %d \n",a);
      printf("*x = %d, *y = %d \n",*x, *y);
      printf("x = %p, y = %p\n\n",x,y);
      *x = 20;
      printf("*x = %d, *y = %d \n",*x, *y);
      printf("x = %p, y = %p\n\n",x,y);
      printf("a = %d \n",a);
      system("PAUSE");
}
```

```
C:\Documents and Settings\renata\Desktop\C\ex03.exe
a = 10
*x = 10, *y = 10
x = 0023FF74, y = 0023FF74

*x = 20, *y = 20
x = 0023FF74, y = 0023FF74

a = 20
Pressione qualquer tecla para continuar. . .
```


Aritmética de Ponteiros

- Ao serem incrementados, os ponteiros passam a apontar para a posição de memória do próximo elemento do seu tipo base.
- Ao serem decrementados, os ponteiros passam a apontar para a posição de memória do elemento anterior do seu tipo base.
- O valor do ponteiro irá aumentar ou diminuir dependendo do número de bytes que o tipo base ocupa.

Aritmética de Ponteiros

- Podemos somar ou subtrair inteiros de ponteiros
  - p1 = p1 + 10;
  - Faz p1 apontar para o décimo elemento do tipo p1 (adiante do elemento atualmente apontado por p1)

Ponteiros e Vetores

- Há uma estreita relação entre ponteiros e vetores.
- O nome de um vetor é um ponteiro que aponta para a primeira posição do vetor e todas as operações já mencionadas para ponteiros podem ser executadas com um nome de vetor.
- Por exemplo, a declaração:

```
int v[100];
int *v;
```

declara um vetor de inteiros de 100 posições e a partir dela temos que v é um ponteiro equivalente a

- Logo, as seguintes declarações são idênticas e podem ser intercambiadas independemente do modo como v foi declarado:

```
v[i] == *(v+i)
&v[i] == v+i
```


Ponteiros e Vetores

- Existe uma diferença fundamental entre declarar um conjunto de dados como um vetor ou através de um ponteiro.
- Na declaração de vetor o compilador automaticamente reserva um bloco de memória para que o vetor seja armazenado.
- Quando apenas um ponteiro é declarado, a única coisa que o compilador faz é alocar um ponteiro para apontar para a memória, sem que espaço seja reservado.

Ponteiros e Strings

- Strings são vetores de caracteres

```
char str[80], *p1;
p1 = str; //atribui a p1 o end. do 1º. elemento de str

str[4] ou
*(p1+4) //devolvem o 5º. elemento de str
```


Inicialização de Ponteiros

- Após ser declarado, e antes de receber um valor, o valor de um ponteiro é desconhecido.
- Um ponteiro que não aponta para um local de memória válido recebe o valor nulo (NULL)
- Exemplo: p1 = NULL;

Problemas com Ponteiros

Exemplo04

- Ponteiro não inicializado

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

int main(){

  int x, *p;

  x = 10;
  *p = x;

  system("PAUSE");
}
```

Problema:
O ponteiro p nunca recebeu valor, por isso contém lixo. O programa atribui o valor 10 a uma posição de memória desconhecida.

Solução:
Certifique-se de que o ponteiro está apontando para algo válido antes de usá-lo.

Problemas com Ponteiros

Exemplo05

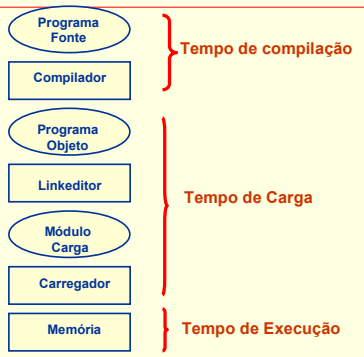
- Endereço vs. Conteúdo

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

int main(){

  int x, *p;

  x = 10;
  p = x;

  system("PAUSE");
}
```

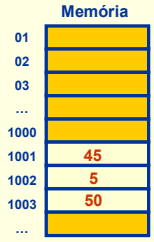
Problema:
O programa está atribuindo o valor de x (10) a um endereço de memória.

Solução:
p = &x; //p recebe o endereço de x

Processamento – Programa Usuário



Memória Principal: *visão simplificada*



- SO carrega o programa a partir de uma determinada posição da memória;
- Nesse espaço, variáveis declaradas são sinônimos de endereço;
- Comandos fazem referências a endereços, portanto, operações referenciam esses endereços.

X = K + N

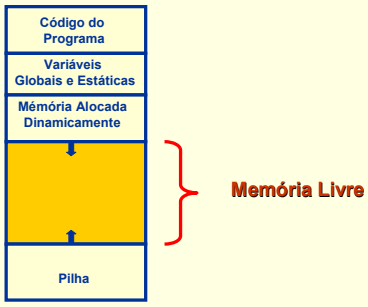
[1003] = [1001] + [1002]

50 = 45 + 5

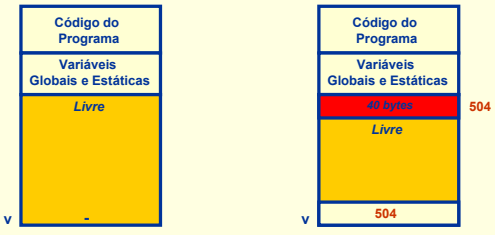
Objetivos

- Apresentar os conceitos de alocação dinâmica de memória
- Mostrar exemplos de alocação dinâmica de memória em C

Uso de Memória



Uso de Memória



Alocação estática de memória

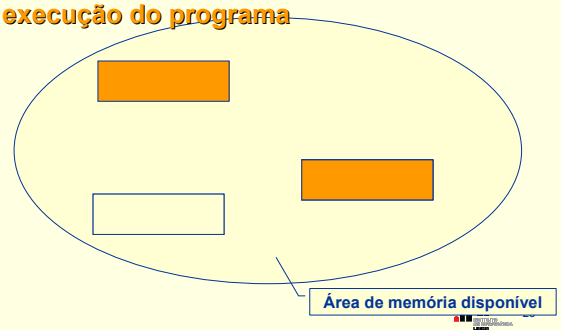
```
int main(){
  int vet[100];   <- declaração
...
  scanf("%d",&vet[8]);  <- referência
...
}
```

Permanece alocado durante toda a execução do programa

É preciso prever todo o espaço necessário antes da execução do programa !

Alocação dinâmica de memória

- espaço de memória alocado e liberado durante a execução do programa



Alocação dinâmica de memória

- espaço é **solicitado** somente quando necessário, e **liberado** quando não for mais necessário
- **erro de execução** caso não exista espaço suficiente
- tamanho e tipo de área de memória alocada informados no momento da solicitação
- Cuidar para não perder o endereço das variáveis alocadas dinamicamente

C

malloc (tamanho)

- aloca uma área de memória para um ponteiro
- Exemplo:

```
char *p;
p = malloc(1000); /*aloca 1000 bytes para o
                    armazenamento de caracteres*/
```

Problema:
A memória ocupada por um tipo pode variar de máquina para máquina.

Solução:
Usar o operador **sizeof.**

```
p = malloc(1000 * sizeof(char));
(char *)malloc(sizeof(char));
```


C

free (< variável apontador >)

- libera a área que está sendo apontada pelo apontador
- apontador passa a conter endereço inválido

```
free(p); /*libera o espaço ocupado por ptr*/
```

< variável apontador > = NULL

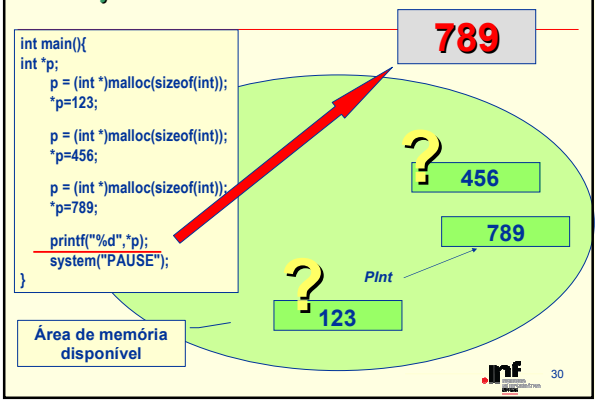
- endereço *NULL* em um apontador significa que este não está associado a nenhuma variável

Exemplo 06

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

int main(){
int *p;
      p = (int *)malloc(sizeof(int));
     *p=10;
      printf("valor P = %d \n",*p);
      printf("endereco P = %d \n",p);
      free(p);
      system("pause");
}
```


Exemplo 07

```
int main(){
int *p;
      p = (int *)malloc(sizeof(int));
      *p=123;

      p = (int *)malloc(sizeof(int));
      *p=456;

      p = (int *)malloc(sizeof(int));
      *p=789;

      printf("%d",*p);
      system("PAUSE");
}
```



Exemplo 08

```
int main(){

int *p,*x,*y;
      p = (int *)malloc(sizeof(int));
      x=p;
      y=p;
      printf("%d\n",*p);
      *p=10;
      printf("p: %d\nx: %d\ny:%d ",*p,*x,*y);
      system("PAUSE");
}
```


Exemplo 09

```
int main(){

int *p,*x;
      x = (int *)malloc(sizeof(int));
      *x=10;
      p = x;
      printf("%d\n%d\n",*x,*p);
      printf("%d\n%d\n",(int)x,(int)p);

system("PAUSE");
}
```


Exemplo 10

```
int main(){
int *p;
int i;
      p = (int *)malloc(sizeof(int));
      i=100;
      *p= i;
      printf("i: %d\n*p: %d\n %d\n",i, *p,(int)p);
      i=40;
      printf("i: %d\n*p: %d\n %d\n",i, *p,(int)p);
      p= &i;
      printf("%d\n",p);
      i=55;
      printf("i: %d\n*p: %d\n %d\n",i, *p,(int)p);
      system("PAUSE");
      free(p);
}
```


Exemplo 11

```
typedef struct {
      char nome[30];
      int idade;
      int altura;
}Pessoa;
int main(){
Pessoa *p;
  p = (Pessoa*) malloc(sizeof(Pessoa));
  printf("endereço de p %d\n",p);
  printf("nome:"); scanf("%s",&p->nome);
  printf("idade:"); scanf("%d",&p->idade);
  printf("altura:"); scanf("%d",&p->altura);
  printf("nome: %s\n",p->nome);
  printf("idade: %d\n",p->idade);
  printf("altura: %d\n",p->altura);
  free(p);
  system("PAUSE");
}
```

Exemplo 12

```
typedef struct Tpessoa{
      char nome[30];
      struct Tpessoa *elo;
}pessoa;

int main(){
pessoa *p1;
pessoa *p2;
  p1 = (pessoa*) malloc(sizeof(pessoa));
  p2 = (pessoa*) malloc(sizeof(pessoa));
  scanf("%s",p1->nome);
  scanf("%s",p2->nome);
  p1->elo=p2;
  printf("%s\n",p1->elo->nome);
  printf("%s\n",p2->nome);
  system("PAUSE");
  free(p1);
  free(p2);
  system("PAUSE");
 return 0;
}
```