PONTEIROS E ALOCAÇÃO DINÂMICA DE MEMÓRIA

Professor: Francisco Dantas Nobre Neto
E-mail: dantas.nobre@ifpb.edu.br

Agenda

- Alocação dinâmica de memória:
  - Malloc e Free.
- Ponteiros para estrutura
- Ponteiros para matrizes
- Matrizes de ponteiros
- Ponteiros para ponteiros

Alocação Dinâmica de Memória

- Trabalharmos com array, em que o seu tamanho é definido de forma fixa, apresenta algum problema?
- Se definirmos o tamanho de um array antes de iniciarmos a execução da aplicação, corremos o risco de desperdiçar memória?

Alocação Dinâmica de Memória

- É possível definir o tamanho de um vetor ou de uma estrutura que se quer alocar em tempo de execução do programa:
  - Para isso, usa-se alocação dinâmica de memória.
- Alocar dinamicamente uma memória, em C, significa invocar a função *malloc(tamanho_espaco_alocado)*:
  - O espaço alocado em memória é em byte;
  - Deve-se usar a biblioteca **<*stdlib.h*>**.
- Geralmente, utiliza-se a função *sizeof(tipo)*, como parâmetro da função *malloc(espaco_alocado)*:
  - *Sizeof()* retorna o tamanho em bytes de um tipo.

Alocação de Dinâmica de Memória

- Utilização da função *sizeof()*.

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

int main(int argc, char *argv[]) {
  int *p;

 /* Estão sendo alocados 4 bytes */
  p = (int *) malloc(4);
  *p = 8;

  free(p);
  return 0;
}
```

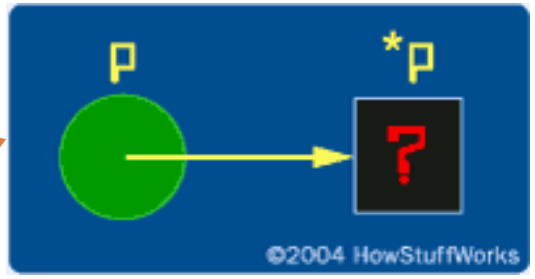
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int *p;

 /* Estão sendo alocados 4 bytes
    (tamanho do inteiro) */
  p = (int *) malloc(sizeof(int));
  *p = 8;

  free(p);
  return 0;
}
```

Mais elegante e mais correto!

Alocação Dinâmica de Memória

- O retorno da função *malloc()* é um ponteiro genérico para qualquer tipo:
  - Por isso, que adiciona-se o *(tipo *)* antes do *malloc()*;
  - Com isso, tem-se um **ponteiro para um número inteiro**.

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

int main(int argc, char *argv[]) {
  int *p;

/* Estão sendo alocados 4 bytes */
  p = (int *) malloc(sizeof(int));

…
}
```



Estado da memória
após a execução do *malloc()*.

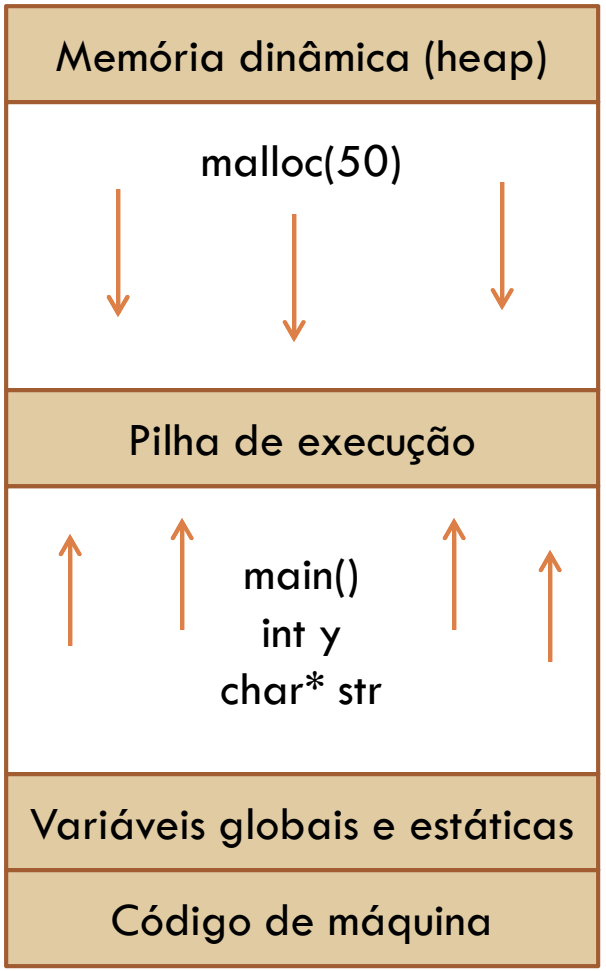
Alocação Dinâmica de Memória

- A função *malloc()* solicitará um bloco de memória na **memória dinâmica (*heap*)**:
  - O sistema operacional reservará um bloco de memória para o programa em que ocorreu a função.
- Ao término da utilização do bloco de memória alocado dinamicamente (com *malloc()*), é importante liberar a memória utilizada:
  - Por meio da função *free(ponteiro)*.

Alocação Dinâmica de Memória

- Funcionamento prático.

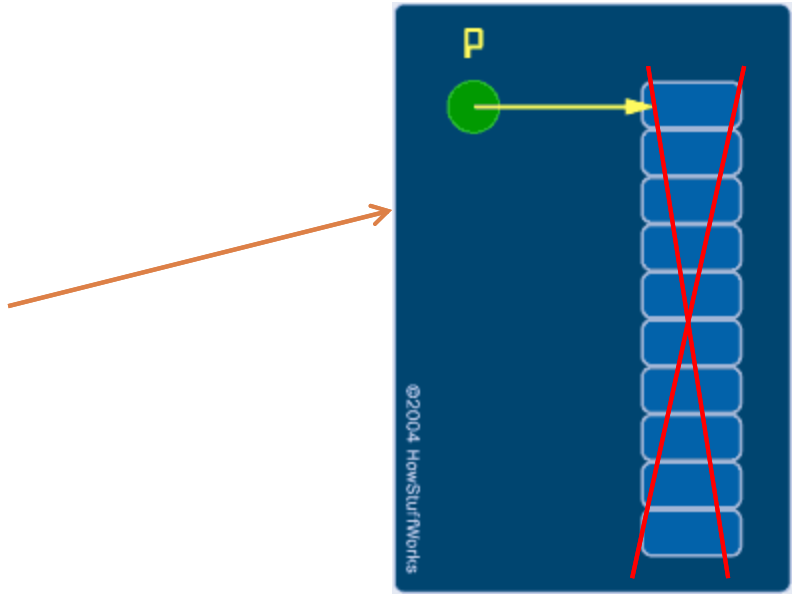
```
int x;                          /* pilha (stack)*/
void main() {
   int y;                       /* pilha (stack) */
   char* str;                    /* pilha (stack) */
   str = malloc(50);             /* aloca 50 bytes
                                  dinamicamente na heap */
}
```



Alocação Dinâmica de Memória

- O comando *free(ponteiro)* desaloca a memória da memória dinâmica (*heap*):
  - Aceita um ponteiro como parâmetro;
  - Após a execução do comando *free(ponteiro)*, a área reservada de memória é liberada.

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

int main(int argc, char *argv[]) {
  int *p;

 /* Estão sendo alocados 40 bytes */
  p = (int *) malloc(10*sizeof(int));
  …
  free(p);
}
```



Alocação Dinâmica de Memória

- É sempre **necessário** verificar se o bloco de memória **solicitado** pelo comando *malloc()* foi realmente criado:
  - Deve-se fazer isso pelo código *if (p == 0)*:
    - Se o *ponteiro* contiver o valor 0 (ou *Nulo*), o bloco de memória não foi alocado;
    - Se o *ponteiro* for diferente de 0, o bloco de memória foi alocado contiguamente.

Alocação Dinâmica de Memória

- Questionamentos:
  - É importante verificar se o ponteiro é zero após cada alocação?
    - Sim, não há garantia de que o *malloc* será bem sucedido.
  - O que acontece se não houver liberação da memória com o *free()*?
    - Quando o programa encerra, o sistema operacional faz a liberação da memória;
    - No entanto, a não liberação de memória durante a execução do programa, pode incorrer num crescimento descontrolado do programa.

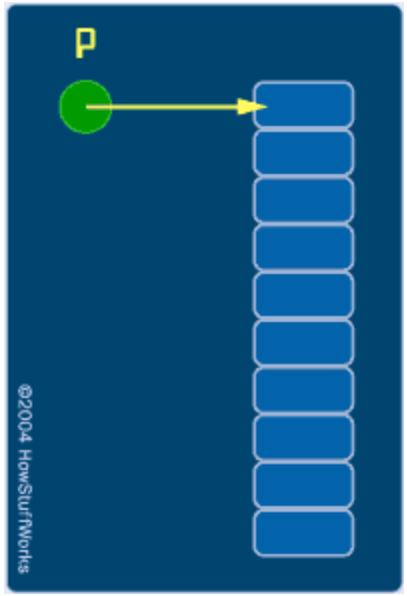
Ponteiros para Vetores

- É possível criar um vetor em tempo de execução, a partir de ponteiros;
- O vetor só seria criado após o comando *malloc()*.

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

int main(int argc, char *argv[]) {
  int *p;

 // Estão sendo alocados 40 bytes
  p = (int *) malloc(10*sizeof(int));
  …
  free(p);
}
```

⇔

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

int main(int argc, char *argv[]) {
  int *p;

 // Estão sendo alocados 40 bytes
  p = (int *) malloc(sizeof(int[10]));
  …
  free(p);
}
```



Ponteiros para Vetores

- É possível percorrer o vetor de duas formas.

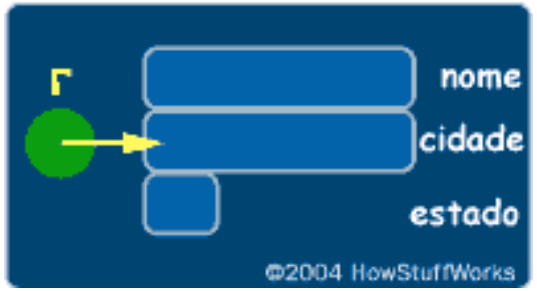
```
int *p;
int i;

p = (int *)malloc(sizeof(int[10]));

if(p == 0) return(0);

???

for (i=0; i<10; i++)
    p[i] = 0;

free(p);
```

```
int *p;
int i;

p = (int *)malloc(sizeof(int[10]));

if(p == 0) return(0);

???

for (i=0; i<10; i++)
    *(p+i) = 0;

free(p);
```

Ponteiro para Estruturas

- É possível criar ponteiro para qualquer tipo em C, inclusive os tipos definidos pelos usuários:
  - As estruturas.

```
/* Definindo a estrutura */
typedef struct {
  char nome[21];
  char cidade[21];
  char estado[3];
} Rec;

typedef Rec *RecPointer;
RecPointer r;

r = (RecPointer) malloc(sizeof(Rec));
```



Ponteiro para Estruturas

- Para preencher os campos da estrutura, deve-se escrever a variável "r", seguido da seta (->):
  - Mas funciona com (*r) também.

```
/* Definindo a estrutura */
typedef struct {
  char nome[21];
  char cidade[21];
  char estado[3];
} Rec;

typedef Rec* RecPointer;
RecPointer r;

r = (RecPointer) malloc(sizeof(Rec));

r->nome = "Joao";
r->cidade = "Campina Grande";
```

```
/* Definindo a estrutura */
typedef struct {
  char nome[21];
  char cidade[21];
  char estado[3];
} Rec;

typedef Rec* RecPointer;
RecPointer r;

r = (RecPointer) malloc(sizeof(Rec));

(*r).nome = "Joao";
(*r).cidade = "Campina Grande";
```

Alocação dinâmica de matrizes

- Assim como feito em vetores, é possível alocar dinamicamente matrizes:
  - Deve-se usar um ponteiro apontando para outro ponteiro:
    - int **p.
  - O primeiro ponteiro pode representar, por exemplo, a linha da matriz;
  - O segundo ponteiro a coluna;
  - O elemento da i-ésima linha e j-ésima coluna, da matriz, será acessado por p[i-1][j-1]:
    - Elemento da 2ª linha e 1ª coluna é acessado por p[1][0].

Alocação dinâmica de matrizes

- Alocar uma matriz de inteiros *m x n*.

```
int main(){
    int m, n, i;
    int **matriz;

    scanf("%d", &m);
    scanf("%d", &n);

    matriz = (int**) malloc(m*sizeof(int*));  // linhas da matriz alocadas
    if(matriz == 0) return(0);

    for(i = 0; i < m; i++){
        matriz[i] = (int*) malloc(n*sizeof(int));  // colunas da i-ésima linha alocadas
        if(matriz[i] == 0) return(0);
    }
    // E para liberar???
    return(0);
}
```

Alocação dinâmica de matrizes

- É possível, e até mesmo necessário para evitar o desperdício de memória, criarmos matrizes de ponteiros;
- Dessa forma, só se aloca o espaço em memória se houver a necessidade:
  - Por meio da função *malloc()*.
- Na prática, é mais utilizado com estruturas, do que com tipos primitivos das linguagens.

Alocação dinâmica de matrizes

- A criação da estrutura só ocorrerá se houver necessidade.

```
/* Definindo a estrutura */
typedef struct {
  char s1[21];
  char s2[21];
  char s3[3];
} Rec;

Rec *a[11];

a[0]= (Rec*) malloc(sizeof(Rec));

strcpy(a[0]->s1, "Joao");
strcpy(a[0]->s2,  "Jose");

free(a);
```

Dez ponteiros não foram inicializados! Economia de memória!