MC-102 — Aula 19
Ponteiros II

Instituto de Computação – Unicamp

20 de Outubro de 2016

Roteiro

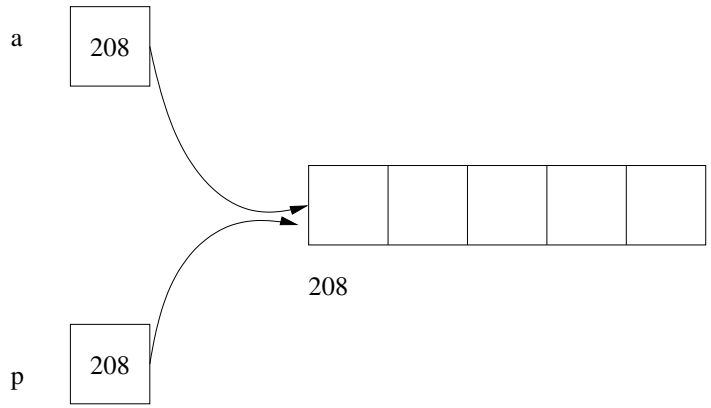
Ponteiros e Alocação Dinâmica

- Lembre-se que uma variável vetor possui um endereço, e que podemos atribuí-la para uma variável ponteiro:

```
int a[] = {1, 2, 3, 4, 5};
int *p;
p = a;
```

- E podemos então usar **p** como se fosse um vetor:

```
for(i = 0; i<5; i++)
    p[i] = i*i;
```



Ponteiros e Alocação Dinâmica

- Em aulas anteriores, ao trabalhar com vetores e matrizes, assumíamos que estes tinham dimensões máximas.

```
#define MAX 100
.
.
.
int vet[MAX];
int m[MAX][MAX];
```

- O que acontece se o usuário precisar trabalhar com vetores ou matrizes maiores?
- Temos que mudar o valor de MAX e recompilar o programa?

Ponteiros e Alocação Dinâmica

- Alocação Dinâmica refere-se a possibilidade de alocar mais memória durante a execução de um programa conforme haja necessidade.
- Pode-se alocar dinamicamente uma quantidade de memória contígua e associá-la com um ponteiro por exemplo. Este será usado como um vetor.
- Desta forma podemos criar programas sem saber a priori o número de dados a ser armazenado.

Ponteiros e Alocação Dinâmica

Na biblioteca **stdlib.h** existem duas funções para se fazer alocação dinâmica de memória: **malloc** e **calloc**.

- Função **malloc :** O seu único parâmetro é o número de bytes que deve ser alocado. A função devolve o **endereço de memória** do início da região que foi alocada ou **NULL** caso aconteça algum erro.
- Exemplo de alocação dinâmica de um vetor de 100 inteiros:

```
int *p, i;
p = malloc(100*sizeof(int));
for(i=0; i<100; i++)
   p[i] = 2*i;
```

Ponteiros e Alocação Dinâmica

- Função **calloc:** Nesta função são passados como parâmetro o número de blocos de memória para ser alocado e o tamanho em bytes de cada bloco. A função devolve o **endereço de memória** do início da região que foi alocada ou **NULL** caso aconteça algum erro.
- Exemplo de alocação dinâmica de um vetor de 100 inteiros:

```
int *p, i;
p = calloc(100, sizeof(int));
for(i=0; i<100; i++)
   p[i] = 2*i;
```

Ponteiros e Alocação Dinâmica

Diferença entre **malloc** e **calloc**.

- A função **calloc** *"zera"* todos os bits da memória alocada enquanto que o **malloc** não. Logo se não for necessária uma inicialização (com zeros) da memória alocada, o **malloc** é preferível por ser um pouco mais rápido.

Ponteiros e Alocação Dinâmica

Juntamente com estas funções, está definida a função **free** na biblioteca **stdlib.h**.

- **free :** Esta função recebe como parâmetro um ponteiro, e libera a memória previamente alocada e apontada pelo ponteiro.
  - Exemplo:

```
int *p;
p = calloc(100, sizeof(int));
....
free(p);
```

- **Regra para uso correto de alocação dinâmica:** Toda memória alocada durante a execução de um programa e que não for mais utilizada deve ser desalocada (com o **free**)!

Exemplo 1 de Alocação Dinâmica de Vetores

Exemplo: Produto escalar de 2 vetores.

- O programa lê inicialmente a dimensão dos vetores e em seguida faz a alocação dos mesmos.

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
    double *v1, *v2, prodEsc;   int n, i;

    printf("Digite dimensão dos vetores:");
    scanf("%d", &n);
    v1 = malloc(n*sizeof(double));
    v2 = malloc(n*sizeof(double));
    ...
}
```

Exemplo 1 de Alocação Dinâmica de Vetores

Exemplo: Produto escalar de 2 vetores.

- O programa faz então a leitura dos dados dos dois vetores.

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
    double *v1, *v2, prodEsc;    int n, i;

    printf("Digite dimensão dos vetores:");
    scanf("%d", &n);
    v1 = malloc(n*sizeof(double));
    v2 = malloc(n*sizeof(double));

    printf("Digite dados de v1: ");
    for(i=0; i<n; i++)
      scanf("%lf", &v1[i]);
    printf("Digite dados de v2: ");
    for(i=0; i<n; i++)
      scanf("%lf", &v2[i]);

    ...
}
```

Exemplo 1 de Alocação Dinâmica de Vetores

Exemplo: Produto escalar de 2 vetores.

- Finalmente o programa calcula o produto e imprime o resultado.
- Note que no final, os dois vetores têm suas memórias liberadas.

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
    double *v1, *v2, prodEsc;    int n, i;

    printf("Digite dimensão dos vetores:");
    scanf("%d", &n);
    v1 = malloc(n*sizeof(double));
    v2 = malloc(n*sizeof(double));
     ...
    prodEsc=0;
    for(i=0; i<n; i++)
      prodEsc = prodEsc + (v1[i]*v2[i]);

    printf("Resposta: %.2lf\n", prodEsc);
    free(v1);
    free(v2);
}
```

Exemplo 1 de Alocação Dinâmica de Vetores

Exemplo completo: Produto escalar de 2 vetores.

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
    double *v1, *v2, prodEsc;  int n, i;

    printf("Digite dimensão dos vetores:");
    scanf("%d", &n);
    v1 = malloc(n*sizeof(double));
    v2 = malloc(n*sizeof(double));

    printf("Digite dados de v1: ");
    for(i=0; i<n; i++)
      scanf("%lf", &v1[i]);
    printf("Digite dados de v2: ");
    for(i=0; i<n; i++)
      scanf("%lf", &v2[i]);

    prodEsc=0;
    for(i=0; i<n; i++)
      prodEsc = prodEsc + (v1[i]*v2[i]);

    printf("Resposta: %.2lf\n", prodEsc);
    free(v1);
    free(v2);
}
```

Exemplo 2 de Alocação Dinâmica de Vetores

Exemplo: Concatenação de strings.

- Criar uma função que recebe duas strings de tamanhos quaisquer e que devolve a concatenação destas.
- Lembre-se que uma função não pode devolver um vetor (uma string é um vetor de caracteres), mas ela pode devolver um ponteiro.
- O protótipo da função será:

```
char * concatena(char *s1, char *s2);
```

Exemplo 2 de Alocação Dinâmica de Vetores

Exemplo: Concatenação de strings.

- Primeiramente devemos alocar a string resposta **sres** com tamanho suficiente para armazenar a concatenação de **s1** com **s2**.

```
char * concatena(char *s1, char *s2){
  char *sres=NULL;
  int t1, t2, i;
  t1 = strlen(s1);
  t2 = strlen(s2);

  sres = malloc((t1+t2+1)*sizeof(char));
  ...
}
```

Exemplo 2 de Alocação Dinâmica de Vetores

Exemplo: Concatenação de strings.

- Depois fazemos a cópia de **s1** e **s2** para **sres** e devolvemos o ponteiro **sres**.

```
char * concatena(char *s1, char *s2){
  char *sres=NULL;
  int t1, t2, i;
  t1 = strlen(s1);
  t2 = strlen(s2);
  sres = malloc((t1+t2+1)*sizeof(char));

  for(i=0; i<t1; i++)
    sres[i] = s1[i];
  for(i=0; i<t2; i++)
    sres[i+t1] = s2[i];
  sres[t1+t2]='\0';

  return sres;
}
```

Exemplo 2 de Alocação Dinâmica de Vetores

Exemplo: Concatenação de strings.

- Considere esta versão onde fazemos a liberação de memória alocada.
- Esta versão está correta?

```
char * concatena(char *s1, char *s2){
  char *sres=NULL;
  int t1, t2, i;
  t1 = strlen(s1);
  t2 = strlen(s2);
  sres = malloc((t1+t2+1)*sizeof(char));

  for(i=0; i<t1; i++)
    sres[i] = s1[i];
  for(i=0; i<t2; i++)
    sres[i+t1] = s2[i];
  sres[t1+t2]='\0';

  free(sres); //Libera Memoria
  return sres;
}
```

Exemplo 2 de Alocação Dinâmica de Vetores

Exemplo: Concatenação de strings.

- Esta versão onde fazemos a liberação de memória alocada está correta?
- Não. A liberação de memória deve ser feita pelo invocador da função após se ter certeza que a string resposta não será mais usada.

```
char * concatena(char *s1, char *s2){
  char *sres=NULL;
  int t1, t2, i;
  t1 = strlen(s1);
  t2 = strlen(s2);
  sres = malloc((t1+t2+1)*sizeof(char));

  for(i=0; i<t1; i++)
    sres[i] = s1[i];
  for(i=0; i<t2; i++)
    sres[i+t1] = s2[i];
  sres[t1+t2]='\0';

  free(sres); //Libera Memoria. Errado!!
  return sres;
}
```

Exemplo 2 de Alocação Dinâmica de Vetores

- Exemplo de implementação e uso correto da função.

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

char * concatena(char *s1, char *s2){
  char *sres=NULL;
  int t1, t2, i;
  t1 = strlen(s1);
  t2 = strlen(s2);
  sres = malloc((t1+t2+1)*sizeof(char));
  for(i=0; i<t1; i++)
    sres[i] = s1[i];
  for(i=0; i<t2; i++)
    sres[i+t1] = s2[i];
  sres[t1+t2]='\0';
  return sres;
}

int main(){
  char s1[100], s2[100], *s3;

  fgets(s1, 100, stdin);
  s1[strlen(s1)-1]='\0'; //Remove '\n'
  fgets(s2, 100, stdin);
  s2[strlen(s2)-1]='\0'; //Remove '\n'

  s3 = concatena(s1, s2);
  printf("%s\n", s3);
  free(s3); //So aqui pode-se liberar a memoria
}
```

Erros Comuns ao Usar Alocação Dinâmica

- Você pode fazer ponteiros distintos apontarem para uma mesma região de memória.
  - Tome cuidado para não utilizar um ponteiro se a sua região de memória foi desalocada!

```
double *v1, *v2;

v1 = malloc(100 * sizeof(double));
v2 = v1;
free(v1);

for(i=0; i<n; i++)
  v2[i] = i*i;
```

- O código acima está errado e pode causar erros durante a execução, já que **v2** está acessando posições de memória que foram liberadas!

Erros Comuns ao Usar Alocação Dinâmica

- O programa abaixo imprime resultados diferentes dependendo se comentamos ou não o comando **free(v1)**. Por que?

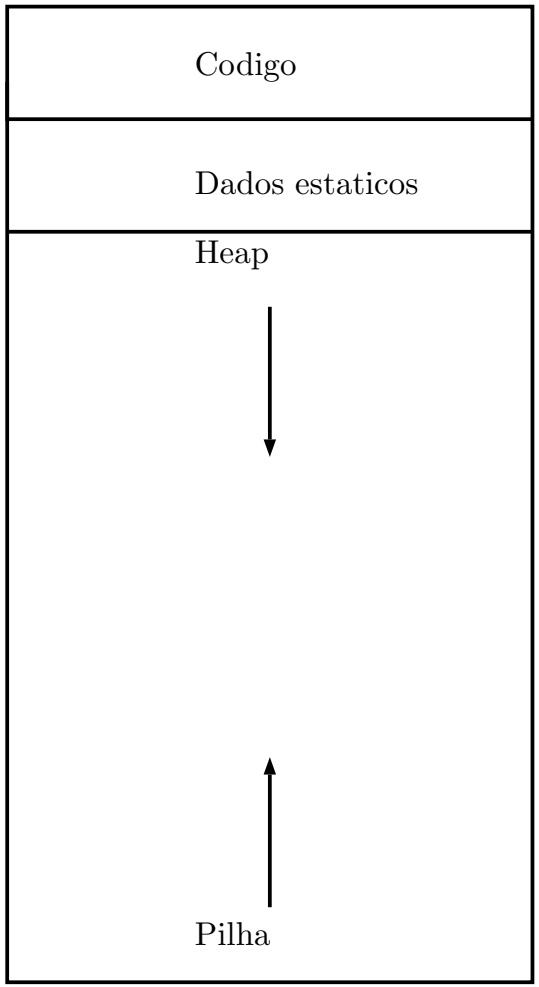
```
#include <stdio.h>
#include <stdlib.h>

int main(){
  double *v1, *v2, *v3;
  int i;

  v1 = malloc(100 * sizeof(double));
  v2 = v1;

  for(i=0; i<100; i++)
    v2[i] = i;
  free(v1); //Comente e nao comente este comando

  v3 = calloc(100,sizeof(double));
  for(i=0; i<100; i++)
    printf("%.2lf\n", v2[i]);
  free(v3);

}
```

Organização da Memória do Computador

A memória do computador na execução de um programa é organizada em quatro segmentos:

- **Código executável:** Contém o código binário do programa.
- **Dados estáticos:** Contém variáveis globais e estáticas que existem durante toda a execução do programa.
- **Pilha:** Contém as variáveis locais que são criadas na execução de uma função e depois são removidas da pilha ao término da função.
- **Heap:** Contém as variáveis criadas por alocação dinâmica.

Organização da Memória do Computador



Organização da Memória do Computador

- Em C99 podemos declarar vetores de tamanho variável em tempo de execução declarando este com tamanho igual ao valor de uma variável.
- No exemplo abaixo declaramos o vetor **v** com tamanho igual ao valor da variável **n** que foi lida do teclado.

```
#include <stdio.h>

int main(){
  long n, i;

  printf("Digite o tamanho do vetor:");
  scanf("%ld", &n);
  double v[n]; //Vetor alocado com tamanho n não pré-estabelecido

  for(i=0; i<n; i++){
    v[i] = i;
  }
  for(i=0; i<n; i++){
    printf("%.2lf\n", v[i]);
  }
}
```

Organização da Memória do Computador

- Porém a criação de vetores desta forma faz a alocação de memória na pilha que possui um limite máximo.
- Execute o programa digitando 1000000 e depois 2000000.

```
#include <stdio.h>

int main(){
  long n, i;

  printf("Digite o tamanho do vetor:");
  scanf("%ld", &n);
  double v[n]; //Vetor alocado com tamanho n não pré-estabelecido

  for(i=0; i<n; i++){
    v[i] = i;
  }
  for(i=0; i<n; i++){
    printf("%.2lf\n", v[i]);
  }
}
```

Organização da Memória do Computador

- O programa anterior será encerrado (*segmentation fault*) se for usado um valor grande o suficiente para n.
- Isto se deve ao fato de que o SO limita o que pode ser alocado na pilha na execução de uma função.
- Este limite não existe para o Heap (com exceção do limite de memória do computador).

Organização da Memória do Computador

Utilizando alocação dinâmica não temos o problema de erro do programa anterior.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    long n=2000000, i;
    double *v = malloc(n*sizeof(double));

    for(i=0; i<n; i++){
      v[i] = i;
    }
    for(i=0; i<n; i++){
      printf("%.2lf\n", v[i]);
    }
    free(v);
}
```

Exercício

Qual o resultado da execução do programa abaixo? Ocorre algum erro?

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

int * misterio(int n){
  int i, *vet;

  vet = malloc(n*sizeof(int));
  vet[0] = 1;
  for(i=1; i<n; i++){
    vet[i] = i*vet[i-1];
  }
  return vet;
}

int main(){
  int i, n, *v;

  printf("Digite n:");
  scanf("%d", &n);

  v = misterio(n);

  for(i=0;i<n;i++)
    printf("%d\n", v[i]);

  free(v);
}
```

Exercício

- Faça um programa que leia a dimensão *n* de um vetor, em seguida aloca dinamicamente dois vetores do tipo *double* de dimensão *n*, faz a leitura de cada vetor e finalmente e imprime o resultado da soma dos dois vetores.

Exercício

- Faça uma função que recebe como parâmetro dois vetores de inteiros representando conjuntos de números inteiros, e devolve um outro vetor com o resultado da união dos dois conjuntos. O vetor resultante deve ser alocado dinamicamente.
- O protótipo da função é

```
int * uniao(int v1[], int n1, int v2[], int n2);
```

onde **n1** e **n2** indicam o número de elementos em **v1** e **v2** respectivamente.