Alocação dinâmica de memória

Franciny Medeiros
franciny@ufj.edu.br

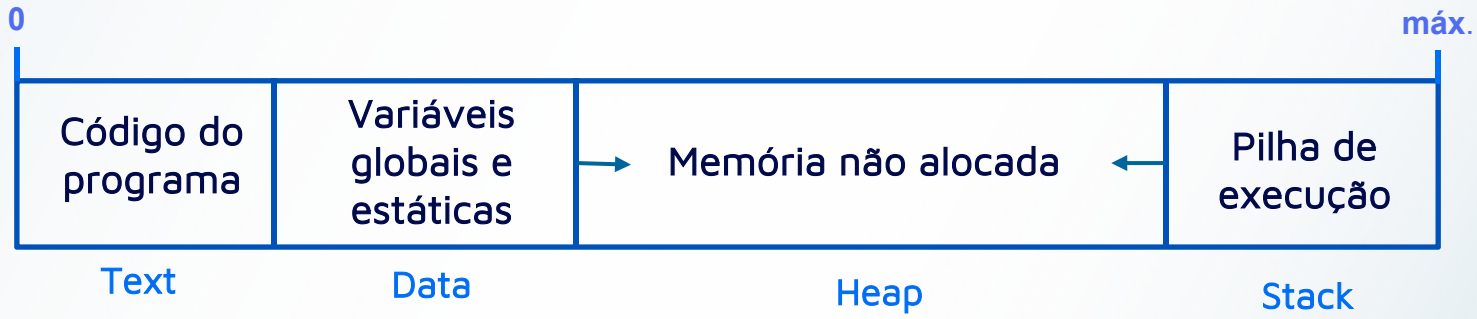
ICET
INSTITUTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

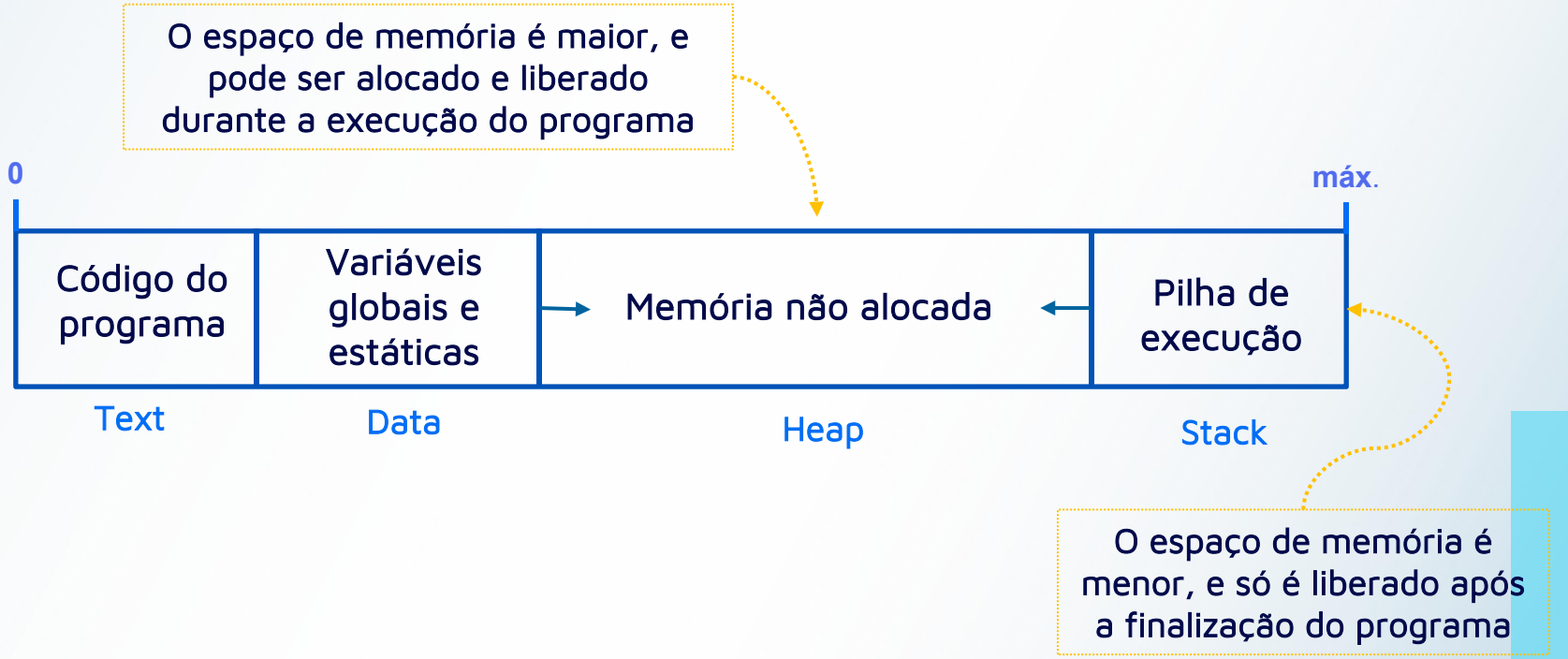
UFJ
UNIVERSIDADE FEDERAL DE JATAÍ

Alocação de memória

- O espaço de um processo em execução é divido em vários segmentos lógicos. Os mais importantes são:
  - **Text**: contém o código do programa e suas constantes.
  - **Data**: este segmento é a memória de trabalho do processo, aonde ficam alocadas as variáveis globais e estáticas.
  - **Stack**: contém a pilha de execução, onde são armazenadas os parâmetros, endereços de retorno e variáveis locais de funções. Pode variar de tamanho durante a execução do processo.
  - **Heap**: contém blocos de memória alocadas dinamicamente, a pedido do processo, durante sua execução. Varia de tamanho durante a vida do processo.

Esquema de memória




O espaço de memória é maior, e pode ser alocado e liberado durante a execução do programa
0
máx.
Código do programa
Variáveis globais e estáticas
Memória não alocada
Pilha de execução
Text
Data
Heap
Stack
O espaço de memória é menor, e só é liberado após a finalização do programa

Tipos de alocação

- Um programa em C suporta três tipos de alocação de memória:
  - **Alocação estática:** quando são declaradas variáveis globais ou estáticas.
  - **Alocação automática:** quando são alocadas variáveis locais e parâmetros de funções.
  - **Alocação dinâmica:** quando o programa requisita um bloco de memória para armazenar. O programador é responsável por liberar as áreas alocadas.

Alocação estática

- A alocação estática ocorre com variáveis globais (alocadas fora de funções) ou quando variáveis locais (internas a uma função).

- O espaço é alocado no início da execução e tem um endereço e tamanho fixos.

- Liberação de memória feita automaticamente pelo SO após a execução do programa.

Alocação estática

- Toda variável é alocada na Stack da memória.
- Após o final da execução, o Sistema Operacional libera os espaços alocados para elas.

```
int a;
char b;
float c;
int v[10];
float *p;
```

Exemplo

```c
#include <stdio.h>

int a = 0 ;  // variável global

void incrementa(void)
{
        int b = 0 ; // variável local, alocação automática
        static int c = 0 ; // variável local, alocação estática

   printf ("a: %d, b: %d, c: %d\n", a, b, c) ;
   a++ ;
   b++ ;
   c++ ;

}

int main(void)
{
   int i ;

  for (i = 0; i < 5; i++)
      incrementa() ;

   return 0 ;
}
```

Saída

```
PS C:\Users\Franciny\
a: 0, b: 0, c: 0
a: 1, b: 0, c: 1
a: 2, b: 0, c: 2
a: 3, b: 0, c: 3
a: 4, b: 0, c: 4
```

Clique aqui e baixe o código

Alocação dinâmica

- O programa solicita explicitamente a alocação de espaço na memória.
  - O programa usa o espaço e depois deve liberá-lo (caso contrário será liberado após a execução do programa.

- As requisições de memória dinâmica são alocadas no Heap.

- Em C a alocação é feita por meio da função “malloc”.

Alocação dinâmica

- A função malloc:
  - aloca espaço para um bloco de bytes consecutivos na memória RAM;
  - devolve o endereço desse bloco;
  - o número de bytes é especificado no argumento da função.

```
tipo *nome = malloc (sizeof(tipo));
```

Função malloc

tipo do ponteiro

função que faz a alocação da memória

o tipo de objeto ou o tamanho

```
tipo *nome = (tipo *) malloc (sizeof(tipo));
```

casting, ou seja, converter esse ponteiro genérico para um ponteiro do tipo desejado

diz quantos bytes o objeto tem

Função malloc

```
int *a = (int *) malloc (sizeof(int))

float *b = (float *) malloc (sizeof(float))

int *c = (int *) malloc (50 * sizeof(int))
```

- Os ponteiros a, b e c serão armazenados na Stack.
- Os espaços alocados para armazenar os conteúdos ficarão no Heap da memória.
- Lembre-se: ponteiro é uma variável de referência!

Exemplo

```c
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>  //biblioteca para usar o malloc

int main() {

    setlocale(LC_ALL, "Portuguese");
    int n, i;

    printf("Digite o número de elementos: ");
    scanf("%d", &n);
    //requisita n espaços de tamanho int
    int *vetor = (int *) malloc(n * sizeof(int));
    //IMPORTANTE!
    //Sempre verificar se a memória foi alocada corretamente
    if (vetor == NULL) {
        printf("Erro na alocação de memória!\n");
        return 1;
    }
    printf("Digite %d números inteiros:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &vetor[i]);
    }
    printf("Valores armazenados no vetor:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", vetor[i]);
    }
    printf("\n");
    free(vetor);
    return 0;
}
```

Clique aqui e baixe o código

Liberação da memória

- As variáveis alocadas estaticamente dentro de uma função desaparecem quando a função termina.
- As variáveis alocadas dinamicamente não desaparecem, é necessário então liberar a memória após sua utilização.
- A função **free** desaloca a porção de memória alocada por malloc.

Função free

```
int *ptr = (int*) malloc (sizeof(int)) ;
...
free (ptr) ; // libera apenas o bloco alocado.
ptr = NULL ;
```

- O ponteiro ptr continua apontando para o bloco liberado e por isso é aconselhável mudar seu valor para ''NULL'' após a liberação.

Por que alocar dinamicamente?

- A alocação dinâmica acontece em tempo de execução.
- É utilizada quando não se sabe ao certo a quantidade de memória necessária.
- O tamanho de memória é alocado conforme a necessidade.
- Evita desperdício de memória e/ou a falta dela.

Atividades

Enviar as respostas no SIGAA
até o fim da aula

1. Escreva um programa que aloque dinamicamente um vetor **v** e o preencha com v[i] = 100*i, sendo que o número de elementos do vetor é lido do teclado. A área de memória alocada deve ser definida em função do tamanho do vetor.

2. Mude o programa anterior, escrevendo funções separadas para:

a) alocar o vetor e preenchê-lo com zeros;
b) preencher o vetor;
c) imprimir o vetor.