INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Alocação Dinâmica de Memória

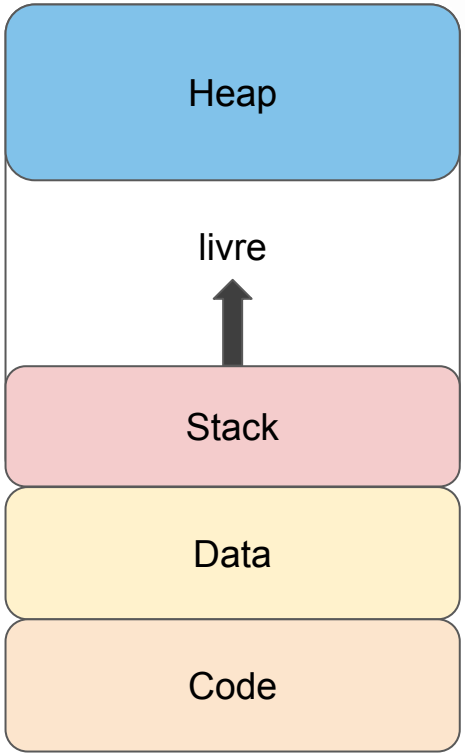
Bruno Brandão

INSTITUTO DE INFORMÁTICA UFG

UFG UNIVERSIDADE FEDERAL DE GOIÁS

Alocação de Memória em C

- **Stack**
  - Pilha de blocos para chamadas de funções
  - Guarda variáveis locais
  - Definida pelo SO
- **Heap**
  - Separada para alocação dinâmica
  - Manuseada pelo programador
  - Dinâmica, com limite definido pelo SO
- **Data**
  - Variáveis globais definidas pelo programador
  - Definida ao compilar o código
- **Code**
  - Guarda o código em linguagem de máquina
  - Definida pelo código compilado



Stack e Heap

Stack

- Veloz (em pilha sabe-se onde alocar)
- Acessa apenas o último endereço
- Liberada quando a função termina

Heap

- Lenta (memória espalhada)
- Flexível e Dinâmica
- O programador deve liberar manualmente

Funções de Alocação

- malloc
  - Alocar memória não inicializada
- calloc
  - Alocar memória inicializada com zeros
- realloc
  - Realocar memória
- free
  - Liberar memória

Função sizeof

- Calcula tamanho de qualquer variável ou tipo
- Retorna valor inteiro (em bytes)

```
char: 1 bytes

int: 4 bytes

float: 4 bytes

double: 8 bytes

pointer: 8 bytes
```

Função malloc

- `void *malloc(size_t size);`
- Aloca memória de tamanho “size”
- Retorna ponteiro para o primeiro byte
- Se não houver memória suficiente, retorna ponteiro nulo
- Sempre verificar se houve sucesso

Função malloc

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


void main() {
    int *p;
    p = malloc(5 * sizeof(int));
    if (p == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    for (int i = 0; i < 5; i++) {
        p[i] = i;
    }
    printf("%d\n", *p);
    printf("%d\n", *(p + 1));
    printf("%d\n", *(p + 2));
    printf("%d\n", *(p + 3));
    printf("%d\n", *(p + 4));
    free(p);
}
```

Função calloc

- `void *calloc(size_t num, size_t size);`
- Aloca memória de tamanho size x num
- Retorna ponteiro para o primeiro byte
- Inicializa a memória com zeros
  - Mais seguro, mas menos eficiente
- Se não houver memória suficiente, retorna ponteiro nulo
- Sempre verificar se houve sucesso

Função calloc

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


void main() {
    int *p;
    p = calloc(5, sizeof(int));
    if (p == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    printf("%d\n", *p);
    printf("%d\n", *(p + 1));
    printf("%d\n", *(p + 2));
    printf("%d\n", *(p + 3));
    printf("%d\n", *(p + 4));
    free(p);
}
```

Função realloc

- `void *realloc(void *ptr, size_t size);`
- Modifica o tamanho de memória previamente alocada apontada por ptr
- Não inicializa a memória
- Pode ser maior ou menor
  - Se maior pode ser necessário mover o bloco de memória
  - Se tamanho for zero, a memória é liberada
- Retorna um ponteiro
- Se não houver memória suficiente, retorna ponteiro nulo
- Sempre verificar se houve sucesso

```c
#include <stdio.h>
#include <stdlib.h>
void main() {
    int *p;
    p = malloc(5 * sizeof(int));
    if (p == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    for (int i = 0; i < 5; i++) {
        p[i] = i;
    }
    printf("Initial allocation:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d\n", p[i]);
    }

    p = realloc(p, 10 * sizeof(int));
    if (p == NULL) {
        printf("Memory reallocation failed\n");
        exit(1);
    }
    for (int i = 5; i < 10; i++) {
        p[i] = i;
    }
    printf("After reallocation:\n");
    for (int i = 0; i < 10; i++) {
        printf("%d\n", p[i]);
    }
    free(p);
}
```

Função free

- `void free(void *ptr);`
- Devolve ao heap a memória apontada por ptr
- Chamar free APENAS com ponteiros criados para alocação dinâmica
- Usar um ponteiro indevido trará comportamentos imprevisíveis no gerenciamento de memória
- Boas práticas: assinalar o ponteiro liberado como NULL

Função free

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


void main() {
    int *p;
    p = malloc(5 * sizeof(int));
    if (p == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    for (int i = 0; i < 5; i++) {
        p[i] = i;
    }
    printf("Allocated memory:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d\n", p[i]);
    }
    free(p);
    p = NULL;
    printf("Memory has been freed.\n");
}
```

Exercício em sala

- Escreva um programa para criar uma matriz com dimensões definidas pelo usuário.O usuário entrará primeiro com o número de linhas, depois de colunas.

Exercício em sala

- Faça um programa que receberá um vetor de elementos inteiros como entrada separados por espaço. O programa deve guardar estes elementos em um vetor.

Exercício em sala

- Escreva um programa para ler dados de sensores em buffers e guardá-los em vetores. Dois sensores tem seus buffers lidos de forma intermitente, podem chegar mais, ou menos leituras em cada entrada, como valores separados por espaço. Para não perdermos o momento em que os dados são colocados, ambos os vetores devem manter o mesmo tamanho. Caso em uma leitura algum tenha menos valores, este deve repetir o último valor.