Alocação Dinâmica de Memória

Victor Turrisi

Introdução

- Quando definimos uma variável em C no formato `int c = 10;`, por exemplo, a memória para a variável c, que no caso é um inteiro, é reservada no momento de “início do programa”

Introdução

```c
#include <stdio.h>


int main(){
    int i = 100;

    for (int i = 0; i < 10; i++){

    }

    printf("Valor de i: %d\n\n", i);

    return 0;
}
```

```
Valor de i: 100
```

```c
#include <stdio.h>


int main(){
    for (int i = 0; i < 10; i++){

    }

    printf("Valor de i: %d\n\n", i);

    return 0;
}
```

```
malloc.c: In function ‘main’:
malloc.c:9:34: error: ‘i’ undeclared (first use in this function)
    printf("Valor de i: %d\n\n", i);
```

Introdução

```c
#include <stdio.h>

int main(){
    int j = 10;

    if (j == 10){
        int i = 5;
    }

    printf("Valor de i: %d\n\n", i);

    return 0;
}
```

```
malloc.c: In function ‘main’:
malloc.c:11:34: error: ‘i’ undeclared (first use in this function)
    printf("Valor de i: %d\n\n", i);
                                 ^
malloc.c:11:34: note: each undeclared identifier is reported only once for each function it appears in
```

Introdução

- Quando definimos uma variável em C no formato `int c = 10;`, por exemplo, a memória para a variável c, que no caso é um inteiro, é reservada no momento de “início do programa”

- Note que na prática cada variável “vive” durante toda a execução do código que faz parte do ***escopo*** onde ela é declarada

- Alguns exemplos de escopos em C:
  - Uma função
  - Corpo interno de um loop
  - Corpo interno de um if
  - Um namespace (em C++)

Introdução

- Entretanto, em diversos casos não sabemos a quantidade de memória que devemos alocar até o momento da execução do próprio programa

- Por exemplo, vamos supor um programa que lê um arquivo de entrada e cria um vetor de structs onde cada posição do vetor é um linha do arquivo

- Como saberemos quantas linhas tem o arquivo até que o programa esteja em execução?

- Será que podemos simplesmente alocar um vetor “muito maior do que o que esperamos que será necessário”?

Introdução

- Para lidar com esses casos, devemos alocar memória dinamicamente
- Em C, isso é feito pelo uso da função ***malloc***
- Além disso, C não possui um garbage collector (GC), ou seja, variáveis alocadas dinamicamente devem ser explicitamente desalocadas pelo programador por meio da função ***free***
- Regra geral: para cada ***malloc*** no código existe um ***free***

Malloc

```
void* malloc (size_t size);
```

Allocate memory block

Allocates a block of *size* bytes of memory, returning a pointer to the beginning of the block.

The content of the newly allocated block of memory is not initialized, remaining with indeterminate values.

If *size* is zero, the return value depends on the particular library implementation (it may or may not be a *null pointer*), but the returned pointer shall not be dereferenced.

http://www.cplusplus.com/reference/cstdlib/malloc/

Free

```
void free (void* ptr);
```

Deallocate memory block

A block of memory previously allocated by a call to `malloc`, `calloc` or `realloc` is deallocated, making it available again for further allocations.

If *ptr* does not point to a block of memory allocated with the above functions, it causes *undefined behavior*.

If *ptr* is a *null pointer*, the function does nothing.

Notice that this function does not change the value of *ptr* itself, hence it still points to the same (now invalid) location.

http://www.cplusplus.com/reference/cstdlib/free

Exemplos

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


int main(){
    char *ptr;
    ptr = malloc(1); // casting implicito
    // ptr = (char *) malloc(1); // casting explicito
    scanf ("%c", ptr);

    printf("Valor de ptr: %c\n\n", *ptr);

    free(ptr);

    return 0;
}
```

Exemplos

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


int main(){
    int *ptr;
    ptr = malloc(sizeof(int));
    scanf ("%d", ptr);

    printf("Valor de ptr: %d\n\n", *ptr);

    free(ptr);

    return 0;
}
```

Exemplos

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


int main(){
    void *ptr;
    ptr = malloc(sizeof(int));
    scanf ("%d", (int *)ptr);

    printf("Valor de ptr: %d\n\n", * (int *) ptr);

    free(ptr);

    return 0;
}
```

Exemplos

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


int main(){
    int *ptr;
    ptr = malloc(10 * sizeof(int));

    for (int i = 0; i < 10; i ++){
        scanf ("%d", ptr+i);
    }

    printf("Valores: \n");
    for (int i = 0; i < 10; i ++){
        printf("%d ", *(ptr+i));
    }
    free(ptr);

    return 0;
}
```

```
10
20
30
40
50
60
70
80
90
100
Valores:
10 20 30 40 50 60 70 80 90 100
```

Exemplos

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

int main(){
    int **ptr;

    // alocação
    ptr = malloc(10 * sizeof(int *));
    for (int i = 0; i < 10; i ++){
        ptr[i] = malloc(10 * sizeof(int));
    }

    for (int i = 0; i < 10; i ++){
        for (int j = 0; j < 10; j ++){
            ptr[i][j] = i + j + i * j;
        }
    }
    printf("Valores: \n");
    for (int i = 0; i < 10; i ++){
        for (int j = 0; j < 10; j ++){
            printf("%d ", ptr[i][j]);
        }
        printf("\n");
    }

    // desalocações
    for (int i = 0; i < 10; i ++){
        free(ptr[i]);
    }
    free(ptr);

    return 0;
}
```

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

int main(){
    int **ptr;

    // alocação
    ptr = malloc(10 * sizeof(int *));
    for (int i = 0; i < 10; i ++){
        ptr[i] = malloc(10 * sizeof(int));
    }

    for (int i = 0; i < 10; i ++){
        for (int j = 0; j < 10; j ++){
            ptr[i][j] = i + j + i * j;
        }
    }
```

```c
    printf("Valores: \n");
    for (int i = 0; i < 10; i ++){
        for (int j = 0; j < 10; j ++){
            printf("%d ", ptr[i][j]);
        }
        printf("\n");
    }

    // desalocações
    for (int i = 0; i < 10; i ++){
        free(ptr[i]);
    }
    free(ptr);

    return 0;
}
```

```
Valores:
0 1 2 3 4 5 6 7 8 9
1 3 5 7 9 11 13 15 17 19
2 5 8 11 14 17 20 23 26 29
3 7 11 15 19 23 27 31 35 39
4 9 14 19 24 29 34 39 44 49
5 11 17 23 29 35 41 47 53 59
6 13 20 27 34 41 48 55 62 69
7 15 23 31 39 47 55 63 71 79
8 17 26 35 44 53 62 71 80 89
9 19 29 39 49 59 69 79 89 99
```

Exemplos

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

struct Teste{
    int a;
    int b;
};

int main(){
    struct Teste *ptr;
    ptr = malloc(sizeof(struct Teste));
    scanf ("%d", &(ptr->a));
    scanf ("%d", &(ptr->b));

    printf("Valor de ptr->a: %d\n\n", ptr->a);
    printf("Valor de ptr->b: %d\n\n", ptr->b);

    free(ptr);

    return 0;
}
```

```
1
2
Valor de ptr->a: 1

Valor de ptr->b: 2
```

Exercícios

- Reimplemente todos os códigos dos trabalhos passados (com exceção do trabalho sobre cidades) utilizando ***apenas*** alocação dinâmica

- Peso 2

- Entrega 04/10