Aula 8 e meio: Alocação Dinâmica e Ponteiros

Professor(a): João Eduardo Montandon (103)
Virgínia Fernandes Mota (106)
jemaf.github.io
http://www.dcc.ufmg.br/∼virginiaferm

INTRODUÇÃO A PROGRAMAÇÃO - SETOR DE INFORMÁTICA

Alocação Dinâmica

- A alocação dinâmica é o processo que aloca memória em tempo de execução.
- Ela é utilizada quando não se sabe ao certo quanto de memória será necessário para o armazenamento das informações, podendo ser determinada em tempo de execução conforme a necessidade do programa.
- Dessa forma **evita-se o desperdício de memória**.

Alocação Dinâmica

- No padrão C ANSI existem 4 funções para alocações dinâmica pertencentes a biblioteca *stdlib.h*.
- São elas: **malloc()**, **calloc()**, **realloc()** e **free()**.
- Existem ainda outras que não serão abordadas nesta aula, pois não são funções muito utilizadas.

Utilização e funções

- A alocação dinâmica é muito utilizada em problemas de estrutura de dados, por exemplo, listas encadeadas, pilhas, filas, arvores binárias e grafos (AEDS II).
- **malloc()** e **calloc()**: são responsáveis por alocar memória;
- **realloc()**: responsável por realocar a memória;
- **free()**: responsável por liberar a memória alocada.

A função **malloc()**

```
1 tipo *variavel;
2
3 variavel = (tipo *) malloc (tamanho * sizeof(tipo));
```

- Esta função recebe como parâmetro "tamanho"que é o número de bytes de memória que se deseja alocar.
- O interessante é que esta função retorna um **ponteiro** do tipo *void* podendo assim ser atribuído a qualquer tipo de *ponteiro*.
- PONTEIRO?!?!?!?!

Ponteiros

- O ponteiro nada mais é do que uma variável que guarda o endereço de uma outra variável.
- Declaração:
  tipo *nome;
- Como atribuir valor ao ponteiro declarado?
  ***nome = valor;**
- Lembrando: Quando usamos uma passagem de parâmetros por referência estamos utilizando ponteiros!!
  - Usamos o operador & para retornar o endereço de memória que está localizado o valor da variável contida no ponteiro.
  - Usamos o operador * para retornar o valor da variável (conteúdo) que está localizada no ponteiro.
- Podem existir ainda ponteiros de ponteiros!
  tipo **nome;

Motivação

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int num_elementos, vet[99] , i;
    printf("Digite a quantidade de elementos que deseja: ");
    scanf("%d", &num_elementos);
    for ( i = 0; i < num_elementos; i++){
        printf("\n Digite o elemento da posição %d: ", i);
        scanf("%d", &vet[i]);
    }
    return 0;
}
```

- E se o usuário colocar apenas 3 elementos no vetor? Desperdício!
- E se o usuário colocar mais de 100 elementos no vetor? Estouro de memória!

Solução

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int num_elementos, i;
    int *vet; //utilizando um ponteiro
    printf("Digite a quantidade de elementos que deseja: ");
    scanf("%d", &num_elementos);
    //Alocando apenas o espaço necessário
    vet = (int *) malloc (num_elementos * sizeof(int));
    for ( i = 0; i < num_elementos; i++){
        printf("\n Digite o elemento da posição %d: ", i);
        scanf("%d", &vet[i]);
    }
    return 0;
}
```

- Agora o usuário pode digitar o tamanho do vetor que desejar e não haverá desperdício de memória!
- Mas ainda temos um problema: Sempre que alocamos dinamicamente a memória precisamos obrigatoriamente liberar essa memória.
    - Na alocação estática isso é feito automaticamente.

A função **free()**

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int num_elementos, i;
    int *vet; //utilizando um ponteiro
    printf("Digite a quantidade de elementos que deseja: ");
    scanf("%d", &num\_elementos);
    //Alocando apenas o espaço necessário
    vet = (int *) malloc (num_elementos * sizeof(int));
    for ( i = 0; i <  num_elementos; i++){
        printf("\n Digite o elemento da posição %d: ", i);
        scanf("%d", &vet[i]);
    }

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

- Libera a memória alocada.
- Deve ser chamada quando o ponteiro não for mais utilizado.

A função **calloc()**

```
1 tipo *variavel;
2 variavel = (tipo *) calloc (tamanho, sizeof(tipo));
```

- Esta função inicia o espaço alocado com 0.
- No exemplo anterior, a alocação ficaria:
  vet = (int *) calloc (num_elementos, sizeof(int));

A função **realloc()**

```
1 tipo *variavel;
2 variavel = (tipo *) realloc (variavel, tamanho);
```

- Esta função altera o tamanho da memória anteriormente alocado.
- No exemplo anterior, a realocação para um novo_tamanho ficaria:
  vet = (int *) realloc (vet, novo_tamanho);

Alocação dinâmica e Subrotinas

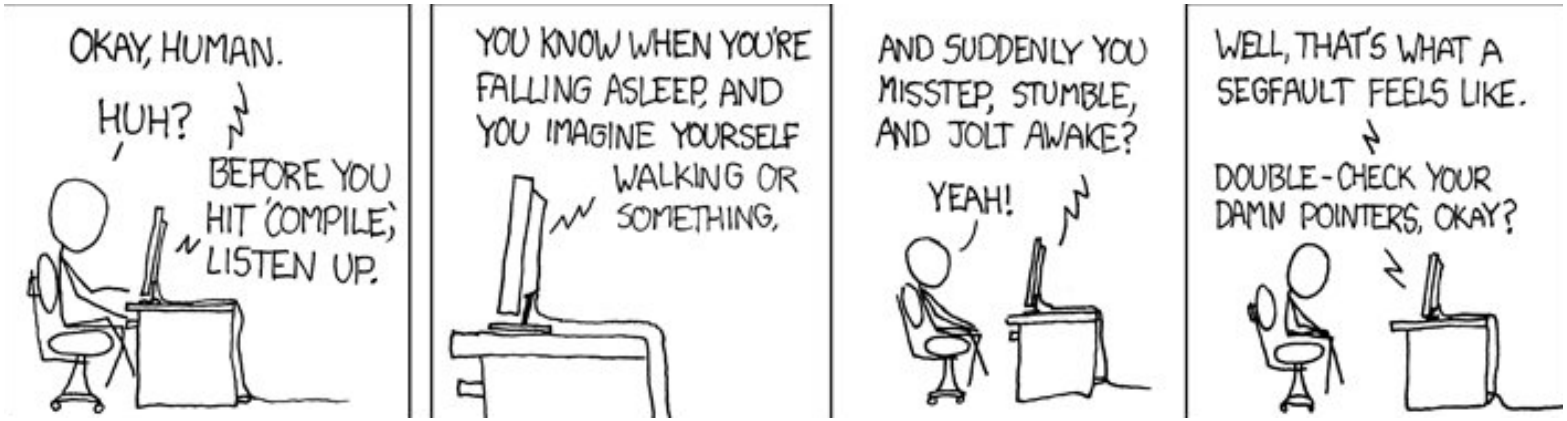
```c
#include <stdio.h>
#include <stdlib.h>

float* alocaVetor(int n){
    float *v;
    v = (float *) malloc (n * sizeof(float));
    return v;
}
void leVetor(float *v, int n){
    int i;
    for (i = 0; i < n; i++)
        scanf("%f", &v[i]);
}

int main(){
    int n;
    float *v;
    printf("\n Digite o numero de elementos do vetor \n");
    scanf("%d", &n);
   v = alocaVetor(n);
    printf("\n Digite seu vetor \n");
    leVetor(v, n);
    free(v);
    return 0;
}
```

- Importante: Não se pode retornar vetores em função, mas pode-se retornar ponteiros!

Resumindo

- A alocação dinâmica reduz o desperdício de memória e torna o programa mais portável.
- As funções mais utilizadas: **malloc()**, **calloc()**, **realloc()** e **free()**.

Resumindo


OKAY, HUMAN.
HUH?
BEFORE YOU HIT 'COMPILE', LISTEN UP.
YOU KNOW WHEN YOU'RE FALLING ASLEEP, AND YOU IMAGINE YOURSELF WALKING OR SOMETHING,
AND SUDDENLY YOU MISSTEP, STUMBLE, AND JOLT AWAKE?
YEAH!
WELL, THAT'S WHAT A SEGFAULT FEELS LIKE.
DOUBLE-CHECK YOUR DAMN POINTERS, OKAY?

Na próxima aula...

Matrizes