



INSTITUTO FEDERAL
Sul-rio-grandense

Câmpus
Charqueadas

EDUCAÇÃO
PÚBLICA
100%
GRATUITA

Alocação Dinâmica

Programação Estruturada

Prof. André del Mestre

Principais Conceitos

Alocação Dinâmica

Preliminares

- Ao escrever um programa, é preciso reservar espaço para as informações que serão processadas.
 - Para isso utilizamos as **variáveis**
 - Uma **variável** é uma posição de memória que armazena uma informação que pode ser modificada pelo programa.
 - Ela deve ser definida antes de ser usada.

```
//Nao funciona  
int i, tam;  
float vet[tam];
```

```
//Nao eh recomendado  
int i, tam;  
scanf("%i", &tam);  
float vet[tam];
```

Flashback

Operações aritméticas com ponteiros

```
int main () {  
    char *ptr_ch=NULL, num_ch='A';  
    float *ptr_fl=NULL, num_fl=1.5;  
    double *ptr_db=NULL, num_db=4.5;  
  
    ptr_ch = &num_ch; //ptr_ch=120  
    ptr_fl = &num_fl; //ptr_fl=122  
    ptr_db = &num_db; //ptr_db=130  
  
    ptr_ch++; ptr_fl++; ptr_db++;  
  
}
```

Cada tipo de dado ocupa
um número de bytes
variável na memória

Endereco	Variavel	Valor
119		
120	num_ch	A
121		
122	num_fl	1.5
123		
124		
125		
126		
127		
128		
129		
130	num_db	3.5
131		
132		
133		
134		
135		
136		
137		
138		
139		
140		

Função sizeof()

Medindo ocupação de memória

```
int main () {
    int a;      char ch;
    printf("tamanho de a = %i\n", sizeof(a));
    printf("tamanho de ch = %i\n", sizeof(ch));
    printf("tamanho de int = %i\n", sizeof(int));
    printf("tamanho de 1 = %i\n", sizeof(1));
    printf("tamanho de double = %i\n", sizeof(double));
    return 0;
}
```

- Função **sizeof()**
 - Retorna o tamanho, em BYTES, que o parâmetro passado ocupa na memória
 - Parâmetro pode ser
 - variavel, tipo, ou constante

```
tamanho de a = 4
tamanho de ch = 1
tamanho de int = 4
tamanho de 1 = 4
tamanho de double = 8
```

Função sizeof()

Medindo ocupação de memória

```
int main (){
    int a;      char ch;
    printf("tamanho de a = %i\n", sizeof(a));
    printf("tamanho de ch = %i\n", sizeof(ch));
    printf("tamanho de int = %i\n", sizeof(int));
    printf("tamanho de 1 = %i\n", sizeof(1));
    printf("tamanho de double = %i\n", sizeof(double));

    char str[15]="Programacao";
    int arr[5]={2,4,6,8,0};
    printf("\ntamanho de str = %i\n", sizeof(str));
    printf("tamanho de arr = %i\n", sizeof(arr));

    return 0;
}
```

- Função **sizeof()**

- Retorna o tamanho, em BYTES, que o parâmetro passado ocupa na memória
- Parâmetro pode ser
 - variavel, tipo, ou constante
 - **vetores**

```
tamanho de a = 4
tamanho de ch = 1
tamanho de int = 4
tamanho de 1 = 4
tamanho de double = 8
```

```
tamanho de str = 15
tamanho de arr = 20
```

Flashback

Relação entre vetores e ponteiros


- O número entre colchetes é o deslocamento a partir do início do array
 - Exemplo:
 - `*(ptr+4)` é equivalente a `ptr[4]`

```
int vet[5]={4,2,0,1,3}, *ptr;
```

```
ptr=vet;
```

```
printf("ptr[0]=%i %i\n", *ptr, vet[0]);  
printf("ptr[0]+4=%i %i\n", *ptr+4, vet[0]+4);  
printf("ptr[4]=%i %i\n", *(ptr+4), vet[4]);
```

Endereco	Variavel	Conteudo
116		
120	int *ptr	128
124		
128	int vet[0]	4
132	int vet[1]	2
136	int vet[2]	0
140	int vet[3]	1
144	int vet[4]	3
148		
152		



```
ptr[0] = 4 4  
ptr[0]+4 = 8 8  
ptr[4] = 3 3
```

Alocação Estática

Garencia de Memória

- Considere um problema que PODE precisar de um vetor de 5000 elementos
 - Exemplo clássico de subutilização: strings
- O tamanho de todos os vetores são definidos em tempo de compilação e não podem ser modificados tempo de execução

```
int vet[5000], *ptr, i, tam;
ptr=vet;
scanf("%i", &tam);
for(i=0; i<tam; i++){
    scanf("%f", &vet[tam]);
}
```

20KB de
memória
apenas para
vetor vet

Endereco	Variavel	Conteudo
116		
120	int *ptr	128
124		
128	int vet[0]	?
132	int vet[1]	?
136	int vet[2]	?
140	int vet[3]	?
144	int vet[4]	?
...
20128	int vet[4999]	?



Alocação Dinâmica

Alocação Dinâmica

Definição

- Alocar dinâmica de memória para novas variáveis é realizada
 - quando o programa está sendo executado (tempo de execução)
- Quantidade de memória é **alocada** sob demanda (quando o programa precisa)
 - Menos desperdício de memória
 - Espaço é reservado até **liberação** explícita
 - Depois de **liberada**, memória estará disponibilizada para outros usos e não pode mais ser acessada
 - Espaço **alocado** e não **liberado** explicitamente é automaticamente **liberado** ao final da execução
- OK, mas **como alocar, realocar e liberar memória dinamicamente?**

Alocação Dinâmica

Função `malloc()`

- Protótipo de `malloc()`
`void * malloc(size_t tam);`
- Para utilizar `malloc()`, você precisa
 - De um ponteiro para receber o retorno `void *`
 - Uso de **casting** é recomendado
 - Definir a quantidade de memória `tam` a ser alocada (utilizando `sizeof()`)

- Exemplo:

```
float *vet;      int i, tam;
scanf("%i", &tam);

vet = (float *) malloc(tam * sizeof(float));

for(i=0; i<tam; i++){
    scanf("%f", vet+tam);
}
```

Alocação Dinâmica

Função `malloc()`

- Verifique se foi possível alocar a memória
 - Função `malloc()` retorna `NULL` em caso de erro

- Exemplo:

```
float *vet;      int i, tam;
scanf("%i", &tam);

//(float *) eh o casting
// tam*sizeof(float) eh qtd de elementos float desejo alocar

vet = (float *) malloc(tam * sizeof(float));
if(vet == NULL){
    printf("Erro ao alocar memoria!");
    return -1;
}
//Faca algo com seu vetor dinamico
```

Alocação Dinâmica

Função `free()`

- Para liberar memória, use

```
void free(void *p);
```

Ao passar o vetor dinâmico `*p` como parâmetro

- Exemplo:

```
float *vet;      int i, tam;  
scanf("%i", &tam);
```

```
vet = (float *) malloc(tam * sizeof(float));  
if(vet == NULL){  
    printf("Erro ao alocar memoria!");  
    return -1;  
}
```

```
for(i=0; i<tam; i++){  
    scanf("%f", vet+i);  
}
```

```
free(vet);
```

Alocação Dinâmica

Utilizando alocação dinâmica corretamente

- A ordem importa
 - 1 - **Aloque** memória
 - 2 - verifique se o vetor foi alocado
 - 3 - faça algo
 - 4 - **libere** memória
 - 5 - Volte ao passo 1

```
int *vet, i=0, tam, op;
do{
    printf("Quantos nums vai digitar?");
    scanf("%i", &tam);

    *vet = (int *) malloc(tam * sizeof(int));
    if(vet == NULL) return -1;

    for(i=0; i<tam; i++){
        printf("diga um num");
        scanf("%i", vet+i);
    }
    free(vet);

    printf("continuar? 0-nao, 1-sim");
    scanf("%i", &op);
}while(op);
```

Alocação Dinâmica

Função `realloc()`

- Protótipo de `realloc()`
`void * realloc(void *p, size_t tam);`
- Função `realloc()`, modifica o tamanho da memória dinâmica previamente alocada
 - A memória a ser realocada é apontada por `*p`
 - O valor `tam` num pode ser maior ou menor que o original.

• Exemplo:

```
float *vet;      int i, tam;
scanf("%f", &tam);

vet = (float *) malloc(tam * sizeof(float));
if(vet == NULL) return -1;

for(i=0; i<tam; i++) scanf("%f", vet+tam);

//ao realocar, remove o ultimo elemento do vetor vet
vet = (float *) realloc(vet, (tam-1) * sizeof(float));
if(vet == NULL) return -1;

free(vet);
```

Alocação Dinâmica

Utilizando alocação dinâmica corretamente

- A ordem importa
 - 1 - **Aloque** memória
 - 2 - verifique se o vetor foi alocado
 - 3 - faça algo
 - 4 - **RE-aloque** memória
 - 5 - verifique se o vetor foi realocado
 - 6 - faça algo
 - 7 - **libere** memória
 - 8 - Volte ao passo 4

```
int *vet, i=0, op;
*vet = (int *) malloc(sizeof(int));
if(vet == NULL) return -1;

do{
    printf("diga um numero");
    scanf("%i", vet+i);

    i++;
    *vet = (int *) realloc(vet, (i+1) * sizeof(int));
    if(vet == NULL) return -1;

    printf("continuar? 0-nao, 1-sim");
    scanf("%i", &op);
}while(op);
free(vet);
```


Alocação Dinâmica e Funções

Alocação Dinâmica e Funções

Contextualização

- Funções retornam `int`, `float` ou `char`
- Funções que manipulam vetores e strings retornam `void`
 - Não é possível retornar vetores
- Isso é problemático do ponto de vista semântico, pois o código não é intuitivo

Alocação Dinâmica e Funções

Contextualização

- Funções retornam `int`, `float` ou `char`
- Funções que manipulam vetores e strings retornam `void`
 - Não é possível retornar vetores
- Isso é problemático do ponto de vista semântico, pois o código não é intuitivo
- Exemplo: Copiar uma string

```
char str[15];
```

```
// certo  
strcpy(str, "Programacao");
```

```
// errados, mas mais intuitivos  
str = "Programacao";  
str = strcpy("Programacao");
```

Alocação Dinâmica e Funções

Exemplo 1 - malloc()

- Função que gera vetor de números aleatórios

```
int * geraVetor(int tam);
```

- Função retorna `int *`, que é endereço do início de um vetor dinâmico de tamanho `tam`
- Note que
 - o `malloc()` é chamado dentro da função `geraVetor()`
 - A função deve retornar o ponteiro alocado dinamicamente, no caso, `*vet`

```
int * geraVetor(int tam) {  
    int *vet, i;  
    vet = (int *)malloc( tam * sizeof(int) );  
    if(vet == NULL) {  
        printf("ERRO\n");  
        return NULL;  
    }  
    for(i = 0; i < tam; i++){  
        *(vet+i) = rand() % 100;  
    }  
    return vet;  
}
```

Alocação Dinâmica e Funções

Exemplo 1 - malloc()

- Função que gera vetor de números aleatórios

```
int * geraVetor(int tam);
```

- Função retorna `int *`, que é endereço do início de um vetor dinâmico de tamanho `tam`
- Note que
 - o `malloc()` é chamado dentro da função `geraVetor()`
 - A função deve retornar o ponteiro alocado dinamicamente, no caso, `*vet`
 - O ponteiro `*vet` no `main()` é o vetor dinâmico
 - A função `free()` é utilizada no `main()`

```
int * geraVetor(int tam) {  
    int *vet, i;  
    vet = (int *)malloc( tam * sizeof(int) );  
    if(vet == NULL) {  
        printf("ERRO\n");  
        return NULL;  
    }  
    for(i = 0; i < tam; i++){  
        *(vet+i) = rand() % 100;  
    }  
    return vet;  
}  
  
int main () {  
    int *vet = gerarVetor(tamanho);  
    printArray(vet, tamanho);  
    free(vet);  
    return 0;  
}
```

Alocação Dinâmica e Funções

Exemplo 1 - malloc()

- A ordem **CONTINUA** importando
 - Antes de chamar a função
 - 1 - Declare um ponteiro para receber o resultado da função
 - Dentro da função
 - 2 - **Aloque** memória
 - 3 - verifique se o vetor foi alocado
 - 4 - faça algo
 - 5 - retorne o vetor dinâmico
 - Após o retorno
 - 6 - **libere** memória
 - 7 - Volte ao passo 1

```
int * geraVetor(int tam) {
    int *vet, i;
    vet = (int *)malloc( tam * sizeof(int) );
    if(vet == NULL){
        printf("ERRO\n");
        return NULL;
    }
    for(i = 0; i < tam; i++){
        *(vet+i) = rand() % 100;
    }
    return vet;
}

int main () {
    int *vet = gerarVetor(tamanho);
    printArray(vet, tamanho);
    free(vet);
    return 0;
}
```

Alocação Dinâmica e Funções

Exemplo 2 - malloc() e strings

- Função que gera uma **NOVA** substring

```
char * subStr(char *str, int inicio, int fim);
```

- Exemplo de uso:

```
char *novaStr, *strDyn, str[15]="Programacao";
```

```
//Ex1 - usando subStr() em string estatica
```

```
novaStr = subStr(str, 3, 7);
```

```
free(novaStr);
```

```
//Ex2 - usando subStr() em string Dinamica
```

```
strDyn = leStr();
```

```
novaStr = subStr(strDyn, 3, 7);
```

```
free(novaStr);
```

```
char * subString(char *str, int inicio, int fim){  
    char *subStr=NULL;  
    int tamSubStr = fim - inicio + 2;  
  
    subStr = (char *) malloc(tamSubStr * sizeof(char));  
  
    if(subStr == NULL)  
        return NULL;  
  
    for(i=inicio; i<=fim; i++){  
        //subStr[j] = str[i];  
        *(subStr+j) = *(str+i); j++;  
    }  
  
    *(subStr+tamSubStr-1) = '\0';  
  
    return subStr;  
}
```

Alocação Dinâmica e Funções

Exemplo 3 - malloc() e realloc()

- Função que gera substring de uma string dinâmica **JÁ EXISTENTE**

```
char * subStrRealloc(char *str, int inicio, int fim);
```

- Exemplo de uso:

```
char *strDyn, *subStr, str [15]="Programacao";
```

```
strDyn = leStr();
```

```
strDyn = subStrRealloc(strDyn, 3, 7);
```

```
free(strDyn);
```

```
char * subStrRealloc(char *str, int inicio, int fim){
    int tam=fim-inicio+2, i, j=0;

    for(i=inicio; i<=fim; i++){
        swapChar(str+i, str+j);
        j++;
    }

    str = (char *) realloc(str, tam * sizeof(char));

    if(str == NULL)
        return NULL;

    *(str+tam-1) = '\0';

    return str;
}
```


MUITO
OBRIGADO

Prof. André del Mestre

www.ifsul.edu.br
almmartins@charqueadas.ifsul.edu.br