Estrutura de Dados 1

Alocação Dinâmica

Prof. Lucas Boaventura
lucas.boaventura@unb.br

Resumo Ponteiros

- Ponteiros podem ser declarados com operador *
- int *p;
- Os ponteiros armazenam o valor de uma área de memória
- Normalmente, esse valor pode ser acessado novamente com o operador *
- *p = 0;

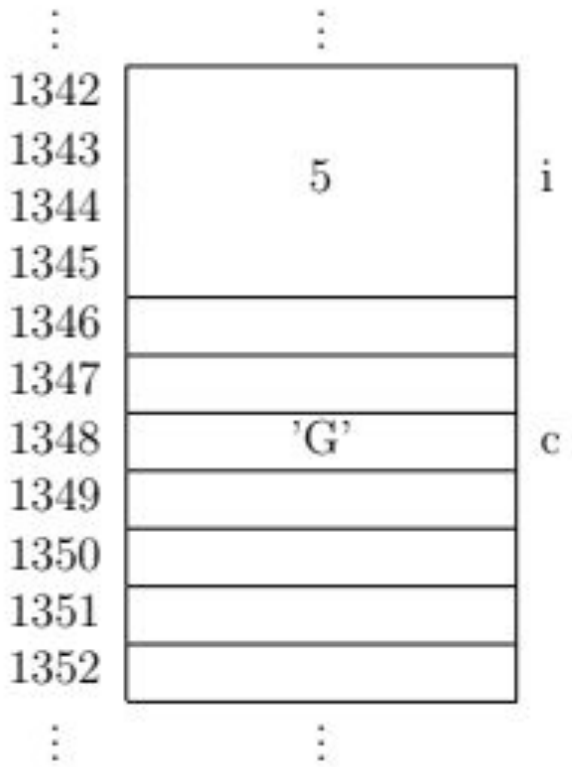
Resumo Ponteiros

- Funções podem alterar o valor do argumento se forem ponteiros: passagem por referência
- Quando uma função não pode alterar o valor do argumento é passagem por valor
- Arrays e ponteiros são conceitos muito ligados e pode-se utilizar os operador * ou [] para acessar o conteúdo

Resumo Ponteiros

- Em um programa em execução, a manipulação de variáveis indicam escrita e armazenamento de dados em memória

```
1 int i = 5;
2 char c = 'G';
```

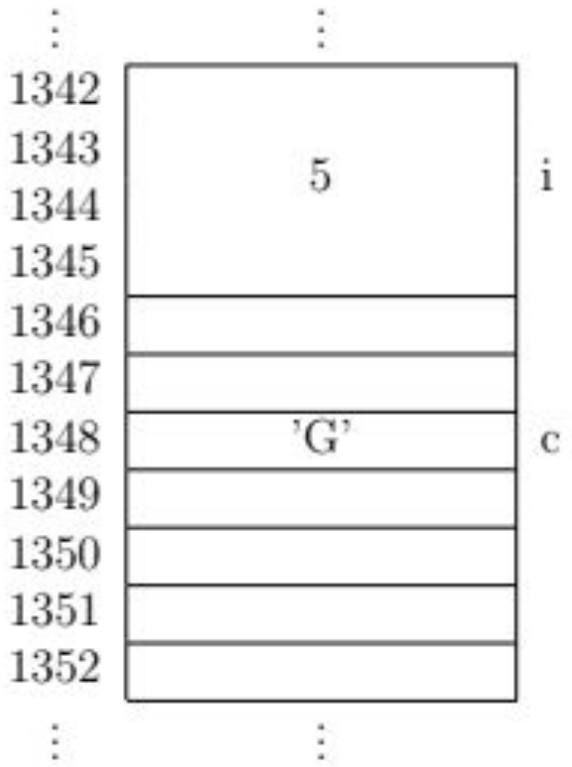


Resumo Ponteiros

- O operador unário & é o operador de endereço. Retorna o endereço que uma variável ocupa na memória.

```c
1 int i = 5;
2 char c = 'G';
```

```c
1 &i; /* Cont´em 1342 */
```



Resumo Ponteiros

- Não esqueçam! Os endereços de memória costumam ser representados em notação hexadecimal.

```c
#include <stdio.h>

int main() {
  int i = 5;
  printf( "&i (dec.): %ld\n", (long int) &i );
  printf( "&i (hexa.): %p\n", (void *) &i );
  return 0;
}
```

Resumo Ponteiros

- Passagem de parâmetros por referência torna possível o “retorno” de vários valores.

```c
#include <math.h>
#include <stdio.h>
int calcula (double n, double *quadrado, double *raiz) {
  *quadrado = n*n;
  *raiz = sqrt (n);
}

int main () {
  double n, quadrado, raiz;
  scanf ("%lf", &n);
  calcula (n, &quadrado, &raiz);
  printf ("Quadrado = %lf\nRaiz = %lf\n", quadrado, raiz);
  return 0;
}
```

Alocação Dinâmica de memória

- Quando uma variável ou vetor é declarado em C, ele é alocado na memória pelo sistema operacional na inicialização do programa. Esse processo é chamado de alocação estática.

- Por outro lado, considere a necessidade de alocar um vetor cujo tamanho não é conhecido previamente.

Alocação Dinâmica de memória

- Nesse caso, podemos usar alocação dinâmica de memória, que consiste em reservar espaço de memória durante a execução do programa.

- Em C, existem três funções para alocação dinâmica de memória.

Alocação Dinâmica - malloc

- A função malloc (abreviação de memory allocation) reserva um bloco de bytes consecutivos de memória com o tamanho especificado em bytes e retorna um ponteiro para o início desse bloco.

```
1 void *malloc (size_t tamanho);
```

Alocação Dinâmica - malloc

- No trecho a seguir, a função malloc aloca 1 byte.

```
1 char *ptr;
2 ptr = malloc (1);
3 scanf ("%c", ptr);
```

Alocação Dinâmica - calloc

- A função calloc é semelhante à malloc: ela aloca memória para um vetor com N elementos, cada um com um tamanho de bytes. A diferença em relação à malloc, além dos parâmetros, é que todas as posições alocadas são inicializadas com zero.

```
1 void *calloc (size_t nmemb, size_t tamanho_tipo);
```

Alocação Dinâmica - calloc

- Neste exemplo, é alocado um vetor de inteiros com 10 posições, todas as 10 posições zeradas.

```
1 int *v;
2 v = calloc (10, sizeof (int));
```

Alocação Dinâmica - realloc

- A função realloc ajusta o tamanho do vetor apontado por ptr para o novo tamanho, em bytes. É importante ter cuidado ao utilizá-la, pois existe um custo implícito: se não houver memória livre suficiente após o bloco original, o vetor precisa ser copiado para outra região da memória.

```
1 void *realloc (void *ptr, size_t tamanho);
```

Alocação Dinâmica - realloc

- No código a seguir, o vetor v é realocado para passar a ter 20 posições.

```
1 v = realloc (v, 20*sizeof (int));
```

Alocação Dinâmica

- As funções de alocação dinâmica retornam um ponteiro do tipo **void** *. Caso a alocação de memória falhe por algum motivo, o valor retornado é **NULL**. Por isso, sempre que realizarmos alocação dinâmica de memória, é necessário verificar se a alocação foi bem-sucedida.

```
1 double *ptr;
2 ptr = malloc(sizeof (double));
3 if (ptr == NULL) {
4     printf("Ponteiro não pode ser alocado\n");
5 }
```

Alocação Dinâmica - free

- A alocação de memória transfere para o programador a responsabilidade pelo espaço alocado. Por isso, é necessário liberar esse espaço ao final do seu uso. Para isso, utiliza-se a função free. Para liberar o espaço alocado para ptr nos exemplos anteriores, basta fazer:

```
1 double *ptr;
2 ptr = malloc(sizeof (double));
3 if (ptr == NULL) {
4     printf("Ponteiro não pode ser alocado\n");
5 }
6 free (ptr);
```

Alocação Dinâmica - wrapper

- Em nossos programas, podemos criar funções de encapsulamento (wrapper functions). Por exemplo, para a função malloc, podemos definir uma função auxiliar que encapsule sua chamada.

```
1 void *mallocc( size_t nbytes ) {
2   void *ptr;
3   ptr = malloc (nbytes);
4   if (ptr == NULL) {
5     printf ("Erro de alocacao de memoria.\n");
6     exit (EXIT_FAILURE);
7   }
8   return ptr;
9 }
```

Alocação Dinâmica - Observações

- Para utilizar as funções de alocação dinâmica de memória, é necessário incluir a biblioteca **stdlib.h**

- **NULL** é uma constante definida na biblioteca stdlib.h que representa o valor zero.

Alocação Dinâmica - Observações

- A função malloc retorna um ponteiro para void. Por isso, é comum fazer um cast na hora da alocação dinâmica. Por exemplo:

```
1 int *ptr = (int *) malloc(5 * sizeof(int));
```

- Todavia, o cast é altamente não recomendável, pois o ponteiro do tipo void é automaticamente convertido para outro tipo de ponteiro.

Alocação Dinâmica - Observações

- **size_t** é um tipo de dado geralmente utilizado para representar tamanhos de objetos. É o tipo retornado pelo operador **sizeof** e, muitas vezes, é equivalente a um **unsigned int**.

Alocação Dinâmica - Observações

- A função **exit** pertence à biblioteca stdlib e i**nterrompe a execução do programa**, fechando todos os arquivos que tenham sido abertos. Se o argumento da função for 0, o sistema operacional é informado de que o programa terminou com sucesso; caso contrário, é informado que o programa terminou de forma excepcional.

- As constantes **EXIT_SUCCESS** e **EXIT_FAILURE** valem 0 e 1, respectivamente, e estão declaradas na biblioteca stdlib.h.

Alocação Dinâmica - Exercícios

1. Seja v um vetor com endereço inicial 1000. Considere o seguinte código:

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

Qual o resultado de cada operação a seguir? Justifique.

- ptr + 1;
- (*ptr) + 1;
- *(ptr + 1);
- *(ptr + 10);

Alocação Dinâmica - Exercícios

2. Seja **vet** um vetor de 4 elementos: TIPO vet[4]. Suponha que, após a declaração, vet esteja armazenado no endereço de memória 3000. Considere ainda que, na máquina utilizada, uma variável do tipo **char ocupa 1 byte, do tipo int ocupa 2 bytes, do tipo float ocupa 4 bytes e do tipo double ocupa 8 bytes.** Qual o valor de **vet + 1**, **vet + 2** e **vet + 3** se:

- vet for declarado como **char**?
- vet for declarado como **int**?
- vet for declarado como **float**?
- vet for declarado como **double**?

Alocação Dinâmica - Exercícios

3. O que faz a seguinte função?

```
1 void imprime (char *v, int n) {
2   char *c;
3   for (c = v; c < v + n; c++)
4   printf ("%c", *c);
5 }
```

Alocação Dinâmica - Exercícios

4. Considerando as declarações:

```
1 int vetor[10];
2 int *ponteiro;
```

Diga quais expressões abaixo são válidas ou não e justifique sua resposta:

- **vetor = vetor + 2;**
- **vetor++;**
- **vetor = ponteiro;**
- **ponteiro = vetor;**
- **ponteiro = vetor + 2;**

Alocação Dinâmica - Exercícios

5. Implemente uma função concat() que concatene dois strings recebidos como argumentos. A função deve retornar um ponteiro para uma nova string resultante da concatenação. Utilize ponteiros e alocação dinâmica o máximo possível.

Alocação Dinâmica - Exercícios

6. Faça um programa que receba uma string e retorne um ponteiro para uma nova string com todos os caracteres em maiúsculo.

Dúvidas?

- lucas.boaventura@unb.br