Revisão da Linguagem C

Vetores e Matrizes

Clodoaldo A. M. Lima

Vetor - Declaração

<tipo> identificador [<número de posiçoes>];

- Primeira posição tem índice 0;
- A última posição tem índice <numero de posicao> - 1;
- Exemplos:
  - **int n[10]**
  - **char c[100]**

Exemplo - Vetor

```
#include <stdio.h>

int main() {
  int num[100];
  int count=0;
  int totalnums;
  do {
    printf("\nEntre com um numero (-999 p/ terminar): ");
    scanf("%d", &num[count]);
    count++;
  } while (num[count-1]!=-999);
  totalnums=count-1;
  printf("\n\n\n\t Os numeros que voce digitou foram:\n\n");
  for (count=0; count<totalnums; count++) {
    printf(" \n%d", num[count]);
  }
  return (0);
}
```

Vetores

- Cuidados importantes ao utilizar vetores
  - O compilador não verifica se o índice é válido:

```
int a[10];
int b = a[32];
int b = a[-32];
// converte f para int
int c = a['F'];
```

  - O código acima compila sem problemas, mas o que irá acontecer quando ele for rodado?
  - **Ninguém sabe!!!**

Matriz - Declaração

- **`tipo_da_variável nome_da_variável [linha][coluna];`**

M[0][0]	M[0][1]
M[1][0]	M[1][1]
M[2][0]	M[2][1]

- Matriz de Strings
  - **`char nome_da_variável [num_de_strings][compr_das_strings];`**

	0	1	2	3	4	5	6	7
0	‘U’	‘T’	‘F’	‘P’	‘R’	‘\0’	lixo	lixo
1	‘E’	‘N’	‘G’	‘\0’	lixo	lixo	lixo	lixo
2	‘c’	‘e’	‘f’	‘e’	‘t’	‘P’	‘R’	‘\0’

Matrizes Multidimensionais

- Sintaxe:
  - *tipo nomoDaMatriz[dim1][dim2]...[dimN];*
- Exemplos:

```
  // Matriz Bidimensional

  int m1[2][2] = { 1, 2, 3, 4 };

  // Outra maneira

  int m2[2][2] = { { 1, 2 }, { 3, 4 } };
// Matriz Tridimensional
int m3[2][2][2] = { 1, 2, 3, 4, 5, 6, 7, 8 };

printf("%d\n", m1[1][1]);
printf("%d\n", m3[1][0][0]);
```

Exemplo - Matriz

```c
#include <stdio.h>

int main() {
   int mtrx [20][10];
   int i, j, count;
   count=1;
   for (i=0; i<20; i++) {
      for (j=0; j<10; j++) {
         mtrx[i][j]=count;
         count++;
      }
   }
   return (0);
}
```

String

Clodoaldo A M Lima

String

- Uma **string** em C é um **vetor de caracteres** terminado com um caractere nulo.
- O caracter nulo é um caractere com valor inteiro igual a zero
- O terminador nulo também pode ser representato em C por '**\0**'.
- O comprimento da string deve ser pelo menos 1 caractere maior que o que pretendemos armazenar, pois um caractere é reservado ao terminador nulo.
- A função **`gets()`** lê uma string e insere o terminador nulo na string quando a tecla **`Enter`** for pressionada.

String

- Usamos um índice para acessar o caractere desejado dentro da string.
  - `str[1] = 'a';`
- Em C, o índice inicia em zero.
- `char str[10] = "Joao";`
  - A declaração acima inicializa a string `str` com os caracteres `'J' 'o' 'a' 'o'` e `'\0'`.
- O código de controle `%s` na função `printf()` é usado para exibir uma string.

String

- Podemos ler uma string usando **`scanf()`**.
  - Não usamos o e comercial (&) para strings, pois o nome de um vetor já é um endereço de memória do começo do vetor.
    - **`scanf("%s", texto);`**
- Infelizmente **`scanf()`** lê somente até o primeiro espaço, ou seja, lê somente uma palavra.
- Para contornar isso, usamos a função gets que lê até encontrar o caracter de fim de linha (enter).
  - **`gets(texto);`**

String

```c
#include <stdio.h>
main() {
   char nome[6];
   printf("Digite um nome: ");
   gets(nome);
   printf("Ola, %s\n", nome);
   getchar(); //

}
```

String

- O problema de gets é que ele pode provocar sérios problemas de segurança, pois permite o armazenamento de caracteres além da capacidade da string.
- Uma solução mais segura é usar a função fgets que limita o tamanho máximo a ser lido.
  - `fgets(texto, 50, stdin);`

String

```c
#include <stdio.h>
main() {
   char nome1[21], nome2[21];
   printf("Digite um nome: ");
   gets(nome1);

   printf("Digite um nome: ");
   fgets(nome2,21,stdin);

   printf("\nNomes:\n%s - %s\n\n", nome1, nome2);
}
```

String

```c
#include <stdio.h>
main(){
     char nome[10] = "Joao";
     printf("String: %s\n", nome);
     printf("Terceira letra: %c\n", nome[2]);
     printf("Quarta letra: %c\n", nome[3]);
     nome[2] = 'h';
     nome[3] = 'n';
     printf("Agora a terceira letra eh: %c\n", nome[2]);
     printf("Agora a quarta letra eh: %c\n", nome[3]);
     printf("String resultante: %s\n", nome);
     system("pause");
}
```

String

```
#include <stdio.h>
main(){
     char nome[10];
     printf("\n\nDigite um nome: ");
     gets(nome);
     printf("\nString: %s\n", nome);
     printf("Terceira letra: %c\n", nome[2]);
     printf("Quarta letra: %c\n", nome[3]);
     printf("o tamanho da string eh: %d\n",
             strlen(nome));
     printf("o ultimo caractere eh: %c\n",
             nome[strlen(nome)-1]);
     nome[2] = 'h';
     nome[3] = 'n';
     printf("Agora a terceira letra eh: %c\n", nome[2]);
     printf("Agora a quarta letra eh: %c\n", nome[3]);
     printf("String resultante: %s\n", nome);
     system("pause");
}
```

String – funções

- **`strlen(texto)`** — Retorna o tamanho da string texto em número de caracteres.
- **`strcpy(destino, fonte)`** — Copia a string fonte para a string destino.
- **`strcat(destino, fonte)`** — Concatena a string fonte no fim da string destino.
- **`strcmp(str1, str2)`** — Compara duas cadeias e caracteres e retorna um valor
  - = 0 - se str1 e str2 forem iguais
  - < 0 - se str1 for menor que str2
  - > 0 - se str1 for maior que str2

strlen

- `int strlen(str);`
- Retorna o tamanho da string passada como parâmetro
- Exemplo:

```
char string[50] = “Linguagem”;

// imprime 9

printf(“%d”, strlen(string));
```

- Obs.: O caractere ‘\0’ não é contado.

strcmp

- `int strcmp(str1, str2)`
- Compara duas strings

Condição	Retorno
<0	Se str1 é menor que str2
0	Se str1 é igual à str2
>0	Se str1 é maior que str2

A ordem lexicográfica é utilizada para a comparação

Exemplo

```
#include <stdio.h>
#include <string.h>
int main() {
    char pergunta[] =
        “qual é a sua linguagem favorita?”;
    char resposta[15];

    do {
        printf(“%s”, pergunta);
        scanf(“%s”, resposta);
    } while (strcmp(resposta, “C”));
    return 0;
}
```

strcpy

- string strcpy(destino, origem)
- Copia o segundo parâmetro no primeiro
- Exemplo:

```
char ori[] = “ABC”;

char dest[12] = “”;

strcpy(dest, ori);

// dest = “ABC”;
```

strcat

- string strcat(destino, origem)
- Concatena o segundo parâmetro no primeiro
- Exemplo:

```
char ori[] = “DEF”;

// Não esqueça de inicializar:

char dest[12] = “ABC”;

// dest = “ABCDEF”

strcat(dest, ori);
```

Outras funções

- **`string gets(string)`**
  - Lê uma string do dispositivo de entrada padrão
  - Também lê espaços
  - **Não verifica o tamanho máximo da string**
- **`int puts(string)`**
  - Imprime uma string no dispositivo de saída padrão
- Ambas estão definidas no header stdio.h

String – funções

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

main() {
   char nome1[] = "Regis";
   printf("%s\n\n",nome1);

   char nome2[100];
   strcpy(nome2, "Isaac");
   printf("%s\n\n",nome2);

   strcat(nome2," Newton");
   printf("%s\n\n",nome2);

   strcpy(nome2,"Maria");
   printf("%s\n\n",nome2);
}
```

Funções não padronizadas

- Há algumas funções muito úteis para manipulação de Strings que não estão disponíveis no C ANSI:
  - strupr
    - Converte para caixa alta
  - strlwr
    - Converte para caixa baixa
  - strrev
    - Retorna

Exemplo – Matriz de Strings

```c
#include <stdio.h>

int main() {
  char nomes[5][100];
  int count;
  for (count=0; count<5; count++) {
    printf("\n\nDigite uma string: ");
    gets(nomes[count]);
  }
  printf("\n\n\nAs strings que voce digitou foram:\n\n");
  for (count=0; count<5; count++) {
    printf("%s\n", nomes[count]);
  }
  return (0);
}
```

Ponteiros

Clodoaldo A. M. Lima

Ponteiros

- Ponteiros são variáveis que contém endereços.
- Estas variáveis apontam para algum determinado endereço da memória.
- Em geral, o ponteiro aponta para o endereço de alguma variável declarada no programa.
- Para acessar o conteúdo de um ponteiro usamos:
  - *nome_do_ponteiro
- O operador * determina o conteúdo de um endereço.
- A um ponteiro pode ser atribuído o valor nulo (constante NULL da biblioteca stdlib.h)
  - Um ponteiro com valor NULL não aponta para lugar nenhum.

Ponteiros

- Guardam o endereço de uma variável
- Sintaxe:
  - *tipo *nomeDoPonteiro;*
- Exemplos:

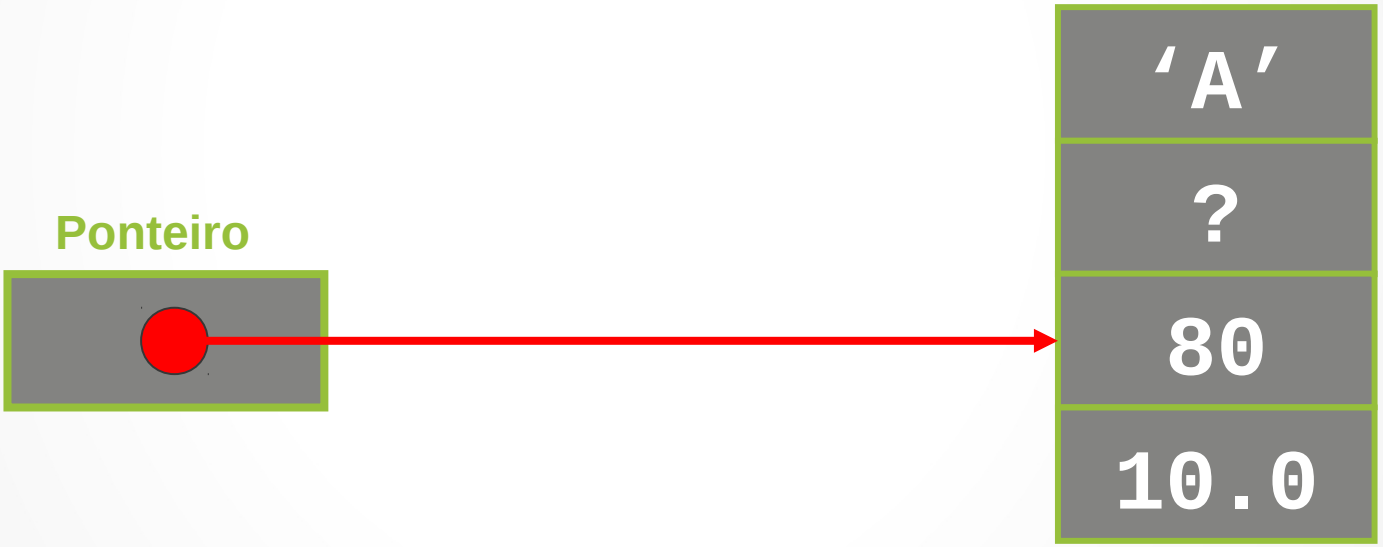
```
int *a;

char *b;

float *c;

double *d;
```

Ponteiros



Declaração de Ponteiros

- Utiliza-se o operador unário *
  - `int *ap_int;`
  - `char *ap_char;`
  - `float *ap_float;`
  - `double *ap_double;`
  - `int *ap1, *ap_2, *ap_3;`
  - `int *ap1, int1, int2;`

Exemplo

```c
#include <stdio.h>

int main() {
   int x = 5;
   int *px;
   px = &x;
   printf("x: %d\n", x);
   printf("px: %d\n", *px);
   return 0;
}
```

Exemplo

```c
#include <stdio.h>

int main() {
   int x = 5;
   int *px;
   px = &x;
   printf("x: %d\n", x);
   printf("px: %d\n", *px);
   x = 7;
   printf("x: %d\n", x);
   printf("px: %d\n", *px);
   *px = 3;
   printf("x: %d\n", x);
   printf("px: %d\n", *px);
   return 0;
}
```

Ponteiro para char

```c
#include <stdio.h>

int strtamanho(char *str) {
  int tamanho = 0;
  while (*str) {
    tamanho++;
    str++;
  }
  return tamanho;
}

int main() {
   char palavra[100];
   printf("Digite uma palavra: ");
   gets(palavra);
   printf("O tamanho e: %d\n", strtamanho(palavra));
   return 0;
}
```

Ponteiro para char

```c
#include <stdio.h>

char *strupper(char *str) {
  char *inicio;
  inicio = str;
  while (*str) {
    *str = toupper(*str);
    str++;
  }
  return inicio;
}

int main() {
   char palavra[100];
   printf("Digite uma palavra: ");
   gets(palavra);
   printf("Em caixa alta: %s\n", strupper(palavra));
   return 0;
}
```

Ponteiros

- Operações aritméticas:
  - Igualdade

```
// p1 aponta para o endereço de p2
p1 = p2;
*p1 = *p2; // copia o conteúdo
```

  - Incremento e decremento

```
p++; // p = p + sizeof(tipo_de_p)
p--; // p = p - sizeof(tipo_de_p)
```

Ponteiros

- Operações aritméticas:
  - incremento

```
p = p + 10;
*p = *p + 10;
```

  - Comparação

```
p1 == p2;
p1 > p2;
p1 >= p2;
```

Exemplo

```
#include <stdio.h>
int main() {
    int a = 10, b = 0, *p, *q;
    p = &a;
    a = 12; // *p = 12
    b = *p; // b = 12
    *p = 15; // a = 15
    q = p; // q e p apontam para o mesmo endereço
    *q = 7; // a = 7
    q = &b;
    *q = *p; // b = a = 7
    (*p)++; // a = 8
    printf("a=%d\nb=%d\n*p=%d\n*q=%d\n", a, b, *p, *q);
    return 0;
}
```

Exercícios

1) Escreva um programa que lê strings do teclado até que duas strings iguais sejam digitadas consecutivamente. A saída é a concatenação de todas as strings lidas e os tamanhos da maior a da menor.

2) Faça um programa que calcula o determinante de uma matriz 2*2. O usuário entra com a matriz e o programa imprime o determinante na tela

3) Qual a diferença entre:

```
p++;      (*p)++;      *(p++);
```

Exercícios

4) Qual os valores de x e de y no final do programa?

```
int main()  {
    int y, *p, x;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x--;
    (*p) += x;
    printf ("y = %d\n", y);
    return 0;
}
```

5) Crie uma função chamada swap que recebe os dois ponteiros para inteiros como parâmetro inverte seus conteúdos.

Linguagem C

Alocação Dinâmica

Clodoaldo A. M. Lima

Alocação Dinâmica

- Usada sempre que não se sabe exatamente quanto de memória será usado para uma determinada tarefa.
- Assim, reserva-se espaço da memória disponível (HEAP) à medida que mais memória torna-se necessária.
- Também pode-se liberar posições de memória quando não forem mais necessárias.
- A memória é alocada não no início do programa, mas sim no decorrer de sua utilização do sistema.
- É como se pudéssemos definir um ARRAY com o seu tamanho sendo alterado à medida que fosse necessário.

Alocação dinâmica de memória

- A alocação dinâmica de memória é realizada através da função **`malloc()`**
  - Ela aloca um bloco consecutivo de bytes na memória e retorna o endereço deste bloco.
  - Protótipo:
    - void *malloc (unsigned int num);
  - Recebe o número de bytes que queremos alocar e retorna um ponteiro para o primeiro byte alocado.
  - Se não houver memória suficiente, retorna um ponteiro nulo (NULL).

Alocação dinâmica de memória

- Função sizeof()
  - Útil para saber o tamanho de um tipo.
- calloc
  - Semelhante a malloc, mas com protótipo um pouco diferente:
    - void *calloc (unsigned int num, unsigned int size);
  - Aloca uma quantidade de memória igual a num * size.
  - p = (int *)calloc(5,sizeof(int));

Alocação dinâmica de memória

- realloc
  - Protótipo:
    - void *realloc (void *ptr, unsigned int num);
  - Modifica o tamanho da memória previamente alocada apontada por *ptr para aquele especificado por num.
  - O valor de num pode ser maior ou menor que o original.
  - Se não houver memória suficiente para a alocação, um ponteiro nulo é devolvido e o bloco original é deixado inalterado.

Liberação dinâmica de memória

- A liberação dinâmica de memória é realizada através da função **`free()`**
  - Ela libera o uso de um bloco de memória.

Ponteiros para estruturas

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

struct pessoa {
  char nome[50];
  int idade;
};

typedef struct pessoa Pessoa;

int main() {
  Pessoa *p = malloc(sizeof(Pessoa));
  strcpy(p->nome,"Clodoaldo");
  p->idade = 18;
  printf("Nome: %s - Idade: %d\n", p->nome, p->idade);
  free(p);
  return 0;
}
```

Exemplo

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

struct pessoa {
  char nome[50];
  int idade;
};

typedef struct pessoa Pessoa;

int main() {
  Pessoa *p = malloc(2 * sizeof(Pessoa));
  Pessoa *inicio = p;
  strcpy(p->nome,"Regis");
  p->idade = 18;
  p++;
  strcpy(p->nome,"Maria");
  p->idade = 25;

  p = inicio;
  printf("Nome: %s - Idade: %d\n", p->nome, p->idade);
  p++;
  printf("Nome: %s - Idade: %d\n", p->nome, p->idade);
  free(inicio);
  return 0;
}
```

Exemplo

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

struct pessoa {
  char nome[50];
  int idade;
};

typedef struct pessoa Pessoa;

int main() {
  Pessoa *p = malloc(2 * sizeof(Pessoa));
  strcpy(p[0].nome,"Regis");
  p[0].idade = 18;
  strcpy(p[1].nome,"Maria");
  p[1].idade = 25;
  printf("Nome: %s - Idade: %d\n", p[0].nome, p[0].idade);
  printf("Nome: %s - Idade: %d\n", p[1].nome, p[1].idade);
  free(p);
  return 0;
}
```

Passagem parâmetros por referência

- A passagem de parâmetros por referência em C requer o uso de ponteiros.