Resumão 2

Vetores, Matrizes, Strings
Structs
Funções e Bibliotecas

Profª. Rose Yuri Shimizu

Roteiro

O que são Algoritmos?

- Método para resolver **problemas** possíveis de serem traduzidos para um programa de computador
- Em Ciência da Computação:

O que são Algoritmos?

- Método para resolver **problemas** possíveis de serem traduzidos para um programa de computador
- Em Ciência da Computação:
  1. Sequência de **instruções** (passos);
  2. Que produz, em tempo finito, a **solução do problema**;

O que são Algoritmos?

- Método para resolver **problemas** possíveis de serem traduzidos para um programa de computador

- Em Ciência da Computação:
  1. Sequência de **instruções** (passos):
     - Ordenada,
     - Finita,
     - Não ambígua,
  2. Que produz, em tempo finito, a **solução do problema**;

O que são Algoritmos?

- Método para resolver **problemas** possíveis de serem traduzidos para um programa de computador

- Em Ciência da Computação:
  1. Sequência de **instruções** (passos):
     - ⋆ **Ordenada**,
     - ⋆ **Finita**,
     - ⋆ **Não ambígua**,
  2. Que produz, em tempo finito, a **solução do problema**;

O que são Algoritmos?

- Método para resolver **problemas** possíveis de serem traduzidos para um programa de computador

- Em Ciência da Computação:
  1. Sequência de **instruções** (passos):
     - ⋆ **Ordenada**,
     - ⋆ **Finita**,
     - ⋆ **Não ambígua**,
  2. Que produz, em tempo finito, a **solução do problema**;

O que são Algoritmos?

- Método para resolver **problemas** possíveis de serem traduzidos para um programa de computador

- Em Ciência da Computação:
  1. Sequência de **instruções** (passos):
     - ⋆ **Ordenada**,
     - ⋆ **Finita**,
     - ⋆ **Não ambígua**,
  2. Que produz, em tempo finito, a **solução do problema**;

O que são Algoritmos?

- Método para resolver **problemas** possíveis de serem traduzidos para um programa de computador

- Em Ciência da Computação:
  1. Sequência de **instruções** (passos):
     - ⋆ **Ordenada**,
     - ⋆ **Finita**,
     - ⋆ **Não ambígua**,
  2. Que produz, em tempo finito, a **solução do problema**;

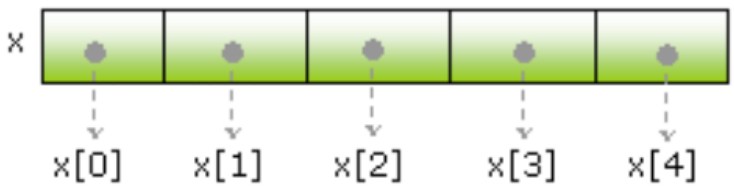
Roteiro

Vetor ou Array unidimensional

- 1 variável → vários valores do mesmo tipo
- Declaração:
  - TIPO VARIAVEL[i];
  - i: constante inteira → quantidade de posições

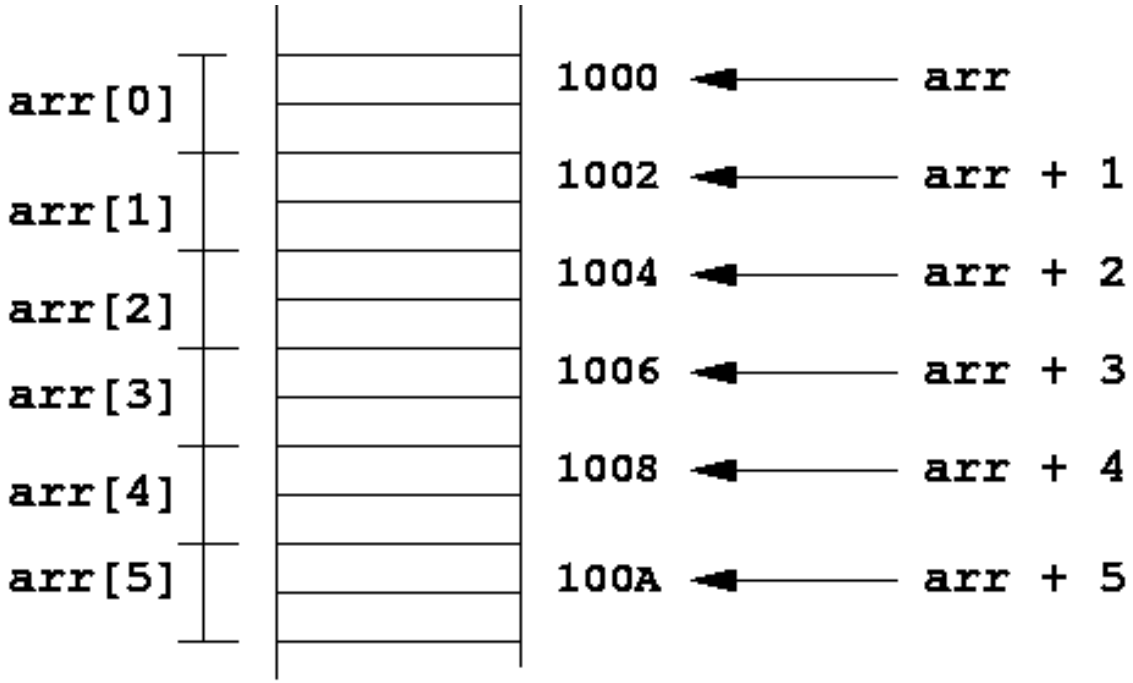
```
1 int produtos[5];
2 char palavra[50];
3
4 //variaveis como indices
5 int i=5;
6 int x[i];
7
8 scanf("%d", &n);
9 int valor[n];
```

- Acesso pelo índice: 0 → n-1



Vetor ou Array unidimensional

Vetor x Endereço Vetor aponta para o endereço da sua primeira posição.
Índice = deslocamento da primeira posição
Conteúdo x Endereço



Vetor ou Array unidimensional

Conteúdo x Endereço

```
1   int arr[2];
2
3   &arr[0] //endereco da primeira posicao do vetor (COM &)
4   &arr[1] //endereco da segunda posicao do vetor (COM &)
5
6   arr+0 //endereco da primeira posicao do vetor (SEM &)
7   arr+1 //endereco da segunda posicao do vetor (SEM &)
8
9   arr[0] //conteudo da posicao 0
10  arr[1] //conteudo da posicao 1
11
12  *(arr+0) //conteudo da posicao 0
13  *(arr+1) //conteudo da posicao 1
```

Vetor ou Array unidimensional

Inicialização na declaração

```
1   float dinheiro[3] = {23.4, 123.0, 55.0};
2   char letras[4] = {'a', 'b', 'c', 'd'};
3
4   //variaveis como indice
5   int a=5;
6   int y[a] = { 1, 4, 6, 99, 2}; //erro
7
8   //so podem ser inicializados integralmente na declaracao
9   int erro[5];
10  erro = { 2, 4, 6, 8, 10 }; //erro
11  erro[0] = 2; //ok
12
13  //o tamanho do array pode ser omitido quando inicializados
14  int peso[] = { 153, 135, 170 }; //compilador aloca
15  int b[]; //erro
```

Vetor ou Array unidimensional

Inicialização pela entrada padrão

```
1     float dinheiro[100]; //0 -> 99
2     char letras[4]; //0 -> 3
3
4     int i=0;
5     while(i<100) {
6         scanf("%f", &dinheiro[i]);
7         i++;
8     }
9
10    for(i=0; i<4; i++) {
11        scanf(" %c", &letras[i]);
12    }
```

Vetor ou Array unidimensional

Inicialização por índice

```
1   int total[5];
2
3   //atribuindo valores pelos indices
4   total[2] = 1;
5   total[3] = 2;
6
7   //acessando valor pelo indice
8   int x = total[3];
9
10  //acessando e atribuindo valores pelos indices
11  int i = 4;
12  total[i] = total[i-1] + total[i-2];
13  total[4]++;
14
15  float tamanho[42];
16
17  //entrada de valor pelo indice
18  scanf("%f", &tamanho[41]);
```

Vetor ou Array unidimensional

Igualar vetores

```
1     int vetorA[10], vetorB[10];
2     int indice;
3
4     // inicializando o vetor A
5     for (indice = 0; indice < 10; indice++) {
6         scanf("%d", &vetorA[indice]);
7     }
8
9     // copiar o conteudo do vetor B para o vetor A
10    vetorA = vetorB;    // ERRADO!
11
12    // copiar o conteudo do vetor B para o vetor A
13    for (indice = 0; indice < 10; indice++) {
14        vetorA[indice] = vetorB[indice];
15    }
16
```

Vetor ou Array unidimensional

Exemplo de quando usar: necessidade de guardar os elementos

- Ler uma certa quantidade de valores inteiros e os imprimir na ordem inversa da leitura;
- Isto é, se os dados de entrada forem: 2, 5, 3, 4, 9, queremos imprimir na saída: 9, 4, 3, 5, 2;
- Este tipo de problema é impossível de ser resolvido com o uso de apenas uma variável pois, quando se lê o segundo número, já se perdeu o primeiro da memória.

Vetor ou Array unidimensional

Exemplo de quando não usar: processar em tempo de execução

- Ler uma certa quantidade de valores inteiros e os imprimir o quadrado dos valores;
- Isto é, se os dados de entrada forem: 2, 5, 3, 4, 9, queremos imprimir na saída: 4, 25, 9, 16, 81;
- Este tipo de problema é recomendado resolver logo após a leitura.

Vetor ou Array unidimensional

Exemplo: procurar x em um vetor v com n posições

```c
int achou = 0, k = 0;
while (k < n && achou == 0) { //testa condicao
    if (v[k] == x) achou = 1;  //testa condicao (mais uma)
    else k++;
}
```

```c
int k = 0;
while (k < n && v[k] != x) k++; //unifica as condicoes
```

```c
int k = 0;
while (v[k] != x && k < n) k++; // errado! (?)
```

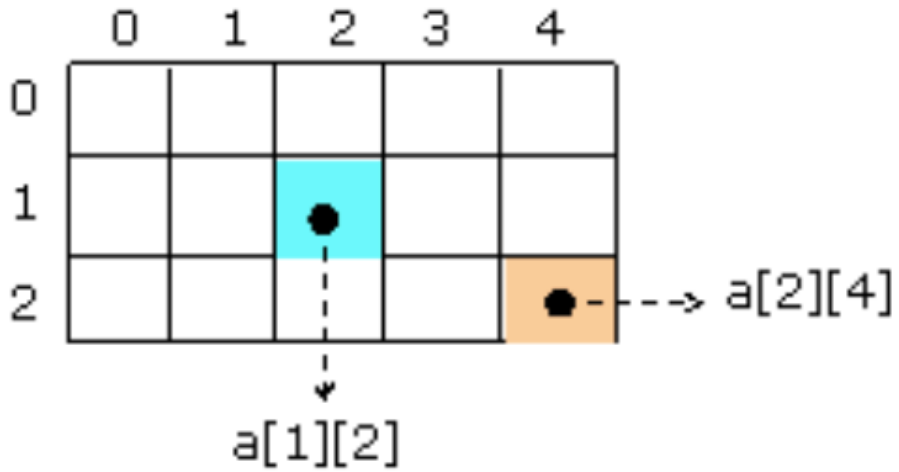
```c
int v[6] = {1, 3, 2, 5, 6}; //uma posicao a mais
int k = 0, x = 2;

v[5] = x;  // sentinela

while (v[k] != x) k++;
```

Matriz ou Array multidimensional

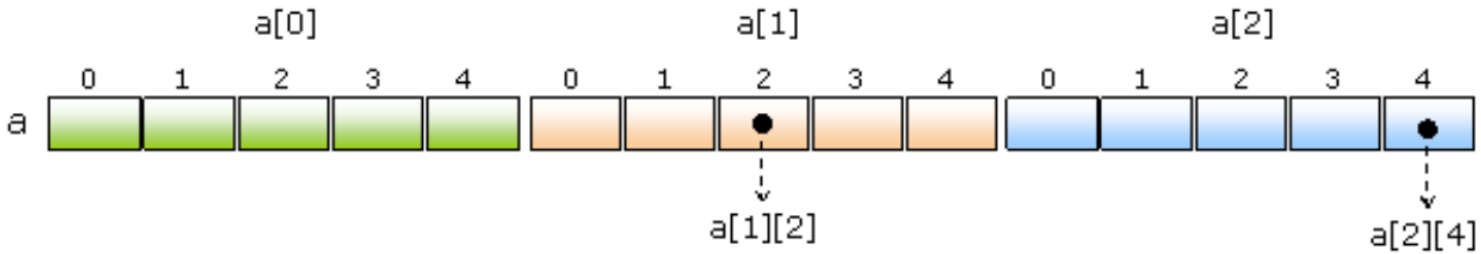
- Matrizes são arrays bidimensionais;
- Acessar os elementos: nome da variável + deslocamento vertical + deslocamento horizontal
- matriz[linha][coluna]

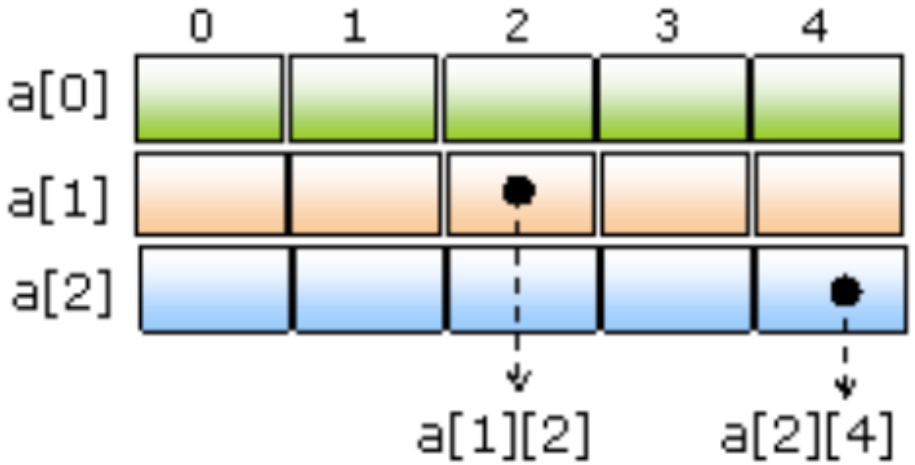


Matriz ou Array multidimensional

Declaração

```
1    int a[3][5];
2
```

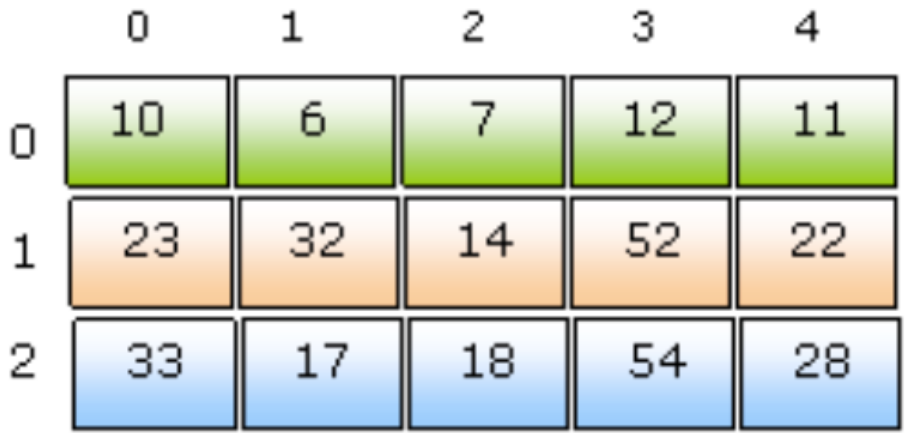




Matriz ou Array multidimensional

Inicialização na declaração e por atribuição

```
1  int a[3][5] = {{10,6,7,12,11}, {23,32,14,52,22},
       {33,17,18,54,28}};
2  int a[][5] = {{10,6,7,12,11}, {23,32,14,52,22}, {33,17,18,54,28}};
3  int a[][5] = {10,6,7,12,11,23,32,14,52,22,33,17,18,54,28};
4
```




```
1  a[0][0] = 3;
2  a[1][4] = 12;
```

Matriz ou Array multidimensional

Inicialização pela entrada padrão

```
1   int a[3][5];
2
3   //linha 0
4   for(int col=0; col<5; col++)
5       scanf("%d", &a[0][col]);
6
7   //linha 1
8   for(int col=0; col<5; col++)
9       scanf("%d", &a[1][col]);
10
11  //linha 2
12  for(int col=0; col<5; col++)
13      scanf("%d", &a[2][col]);
14
15  //-----------------------------------------------------------
16  for(int lin=0; lin<3; lin++)      //para cada linha
17      for(int col=0; col<5; col++) //e cada coluna
18          scanf("%d", &a[lin][col]);
```

Matriz ou Array multidimensional

Impressão: percorrendo a matriz

```
1   int a[3][5] = {{10,6,7,12,11}, {23,32,14,52,22},
    {33,17,18,54,28}};
2
3   int lin, col;
4
5   for(lin=0; lin<3; lin++)
6       for(col=0; col<5; col++)
7           printf("%5d", a[lin][col]);
8       printf("\n");
```

Saída:
10 6 7 12 11
23 32 14 52 22
33 17 18 54 28

Matriz ou Array multidimensional

Faça um algoritmo que leia duas matrizes A e B, com tamanho 3x2, e imprima a soma A+B

$$A + B = \begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}.$$

Matriz ou Array multidimensional

```
1   int a[3][2], b;
2   int lin, col;
3   for(lin=0; lin<3; lin++)
4       for(col=0; col<2; col++)
5           scanf("%d", &a[lin][col]);
6
7   for(lin=0; lin<3; lin++)
8   {
9       for(col=0; col<2; col++)
10      {
11          scanf("%d", &b);
12          printf("%d ", a[lin][col] + b);
13      }
14      printf("\n");
15  }
16
```

Matriz ou Array multidimensional

Dada a matriz abaixo e imprimir sua transposta

- Matriz:
  12 45 67 88
  32 56 44 23
  78 56 33 89
- Transposta:
  12 32 78
  45 56 56
  67 44 33
  88 23 89

Matriz ou Array multidimensional

```
1   int v[3][4] = { {12, 45, 67, 88},
2                   {32, 56, 44, 23},
3                   {78, 56, 33, 89} };
4   int col, lin;
5
6   for(col=0; col<4; col++)
7   {
8       for(lin=0; lin<3; lin++)
9       {
10          printf("%d ", v[lin][col]);
11      }
12      printf("\n");
13  }
14
```

String (sequência de caracteres)

- Arrays de caracteres que DEVEM terminar com '\0'.
- Inicialização na declaração

```
1     char x[6]="maria"; //string
2
```

- O compilador automaticamente coloca o '\0';
- Lembre-se: sempre adicionar 1 no tamanho do array;
- String x char: vantagem na manipulação de blocos/sequencias de caracteres

String (sequência de caracteres)

```c
//Inicializacao de strings
char y[] = "maria"; //string
//y = "maria"; erro
char z[] = {'m','a','r','i','a'}; //vetor de caracteres
char nome1[100], nome2[100];

//especificador de formato: %s
//sem & : vetor = endereco da 1a posicao
scanf("%s", nome1); //lendo 1 palavra
                    //ate encontar ' ', \t, \n

//le no maximo 99 caracteres seguidos
scanf("%99s", nome2);

printf("%s e %s e %s\n", y, nome1, nome2);
```

Entradas
Joao
Jose Paulo

Saída: maria e Joao e Jose

String (sequência de caracteres)

Lendo frases

```
0     char nome1[100];
1
2     //leia tudo ate encontrar a quebra de linha
3     //nao \n
4     scanf("%99[^\n]", nome1);
5
6     printf("%s\n", nome1);
7
```

Exemplo:

Entradas	**Saída:**
Jose da Silva	Jose da Silva

String (sequência de caracteres)

Lendo palavras e frases

```
0     char nome1[100], nome2[100], nome3[100];
1
2     scanf("%99s", nome1); //maximo 99 caracteres seguidos
3     scanf("%99s", nome2);  //ou ate encontar ' ', \t, \n
4
5     //le o restante da entrada anterior ate o \n
6     scanf("%99[^\n]", nome3); //leia tudo ate o \n
7
8     //"consumir" o \n anterior: utilize um espaco
9     //scanf(" %99[^\n]", nome3);
10
11    printf("1 %s\n", nome1);
12    printf("2 %s\n", nome2);
13    printf("3 %s\n", nome3);
```

Entradas	**Saída:**
Joao Maria	1 Joao 2 Maria
Jose da Silva	3

String (sequência de caracteres)

Percorrendo a string

```
0     char nome1[100];
1     scanf("%s", nome1); //sem & : vetor = endereco da 1a posicao
2
3     i = 0;
4     while(nome1[i]!= '\0')
5     {
6         //especificador %c
7         printf("%c", nome1[i]);
8         i++;
9     }
10    printf("\n");
11
12
```

String (sequência de caracteres)

Array de Strings

```
0  char nomes[5][20] = {
1        "Jose Silva",
2        "Maria Silva",
3        "Antonio dos Santos",
4        "Pedro dos Santos",
5        "Joao da Silva"};
6
7  //for(int i=0;i<5;i++)
8    //scanf(" %19[^\n]", nomes[i]);
9
10 for(int i = 0; i < 5; i += 1)
11   printf("%s\n", nomes[i]);
```

Roteiro

Registro

- Agrupa **várias variáveis** e **de vários tipos** em uma **única estrutura**;
- **Definindo um novo tipo de dados**:
  tipo_pessoas clientes, estudantes;

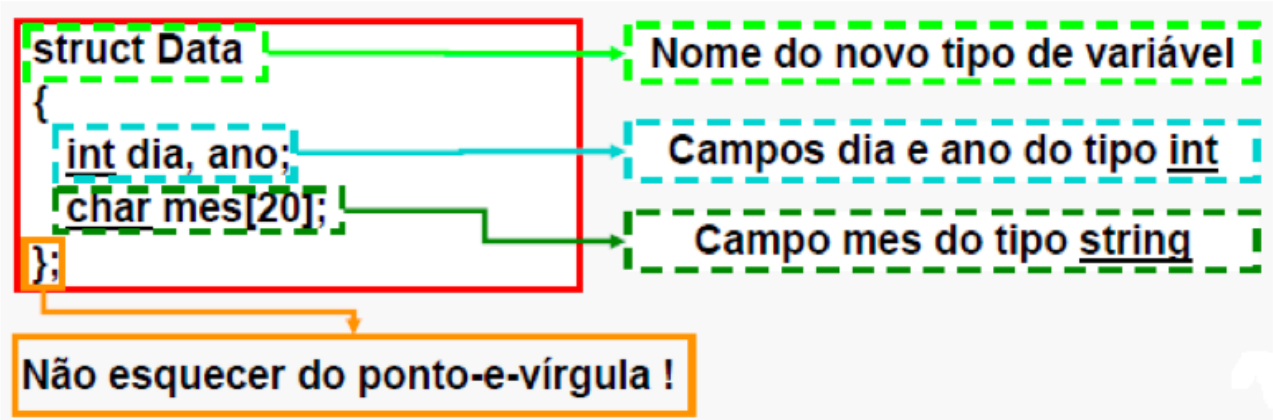
  Pessoa tem nome (string), endereco (string), idade (inteiro)

Registro

Registros em C: struct

- Tipo **struct** em linguagem C
- Cada elemento é denominado campo
- Formato:

```
0   struct nome_da_estrutura {
1     tipo_campo1 nome campo1;
2     tipo_campo2 nome campo2;
3     ...
4   };
```



Registro

- Declaração do **Tipo cadastroAluno**

```
0     //Criacao do tipo
1     struct cadastroAluno {
2       char nome[50];
3       int ra, idade;
4     };
5
6     //Declaracao da variavel
7     struct cadastroAluno Alu, Alu1 , Alu2;
8
9     //OU
10
11    //Criacao do tipo e declaracao das variaveis
12    struct cadastroAluno {
13        char nome[50];
14        int ra, idade;
15    } Alu, Alu1, Alu2;
```

- Uma **struct não é uma variável**, é o nome um **novo tipo de dados**;

Registro - Exemplo

- **Inicialização dos campos** da variável do tipo **cadastroAluno**;
- **Cada campo do registro** é acessado através do **ponto '.'**:

```
0   struct cadAluno {
1     char nome[50];
2     int ra, idade;
3   };
4   int main() {
5       struct cadAluno Alu[2], Alu1 , Alu2;
6       struct cadAluno Alu3 = {"Maria Julia", 111};
7
8       scanf("%[^\n]", Alu1.nome);
9       scanf("%d", &Alu1.ra);
10
11      Alu1.idade = 20;
12
13      printf("%s\n", Alu1.nome);
14      printf("%d\n", Alu1.ra);
15
16      Alu[0] = Alu3; //por copia
17  }
```

Registro - Exemplo

Registro dentro de registro

```
0   struct data {
1     int dia, mes, ano;
2   };
3   struct ficha_cadastral {
4     char nome[50];
5     int id;
6     struct data nascimento;
7   };
8
9   int main()
10  {
11      struct ficha_cadastral alunos;
12
13      scanf("%[^\n]", alunos.nome);
14      scanf("%d", &alunos.id);
15
16      scanf("%d %d %d", &alunos.nascimento.dia,
17                        &alunos.nascimento.mes,
18                        &alunos.nascimento.ano);
19      return 0;
20  }
21
```

Registro - Exemplo

Typedef: nome para o TAD

```
0     struct cadrastroAluno {
1         char nome[20];
2         int idade;
3     };
4     struct cadrastroAluno aluno1;
5
```

```
0     typedef struct cadrastroAluno {
1         char nome[20];
2         int idade;
3     } cadAluno;
4     cadAluno aluno1;
5
```

```
0     typedef struct cadrastroAluno cadAluno;
1     cadAluno aluno1;
2
```

Registro - Exemplo

```
0  typedef struct endereco {
1    char rua[50];
2    char cidade_estado_cep[50];
3  }endereco;
4
5  struct estudante {
6    char id[10];
7    int idade;
8    endereco casa;
9    endereco escola;
10 };
11 typedef struct estudante Aluno;
12 Aluno pessoa;
13
```

Roteiro

Definição de Função

- Pequenos programas que fazem uma tarefa específica;
- Podem ser utilizadas por qualquer outra função
- Exemplos: printf, scanf, strlen, strcmp, etc.
- Componentes:

```
0    tipo_retorno nome_funcao()
1    {
2        ...
3        tipo_retorno a;
4        ...
5        return a;
6    }
7
```

  - **tipo_retorno**: o que a função devolve para o código que chamou:
    - ★ int, double, float, char;
    - ★ void (vazio, nada);
  - **nome_funcao**: o nome utilizado para chamar a função;
  - **return**: palavra reservada que indica o que será retornado;

Definição de Função

Retorno do scanf

- A função **scanf** retorna:
  - man scanf
  - **Número de itens** de entrada **combinados e atribuído** com sucesso;
  - O valor **EOF** é retornado se o final da entrada é alcançado antes da primeira leitura ou falha de correspondência ou erro de leitura.

```
0 while(scanf("%d", &d) != EOF) { } //ctrl+d = end of file
1 while(scanf("%d", &d) == 1)   { }
2
```

```
0 int d;
1 while(scanf("%d", &d) != EOF) //ctrl+d = end of file
2 {
3     printf("%d\n", d);
4 }
5
```

Definição de Função

Retorno do printf

- A função **printf** retorna:
  - man 3 printf
  - Sucesso: **número de caracteres impressos**
  - Erro: **número negativo**

```
0  int a = printf("alo\n");
1  printf("%d\n", a);
2  /*
3  Saida
4  alo
5  4
6  */
7
```

Retorno das funções

```
0   // funcao que soma as entradas ate EOF
1   int somas() {
2       int a, s=0;
3       while(scanf("%d", &a) != EOF)
4           s=s+a;
5       return s;
6   }
7
8   void imprime() {
9       printf("Ola mundo!\n");
10  }
11
12  float obtem_valor()
13  {
14      float valor;
15      printf("Entre um valor:");
16      scanf("%f", &valor);
17      return valor;
18  }
```

Retorno das funções

Localização da função: antes da "invocadora"

- Dever ser definida/criada antes de quem invoca a função

```
0   float f1() {
1       float a, b;
2       scanf("%f %f", &a, &b);
3       if(b!=0) return a/b;
4       else return 0;
5   }
6
7   int main() {
8       char x;
9       scanf("%c", &x);
10
11      while(x!='s') {
12          printf("%f\n", f1());
13          scanf(" %c", &x);
14      }
15      return 0;
16  }
```

Retorno das funções

Definição depois "invocadora" com declaração do protótipo

- "Tudo que usa, tem que declarar";
- Onde declarar?
  - Antes da função (antes da main) ou em arquivos cabeçalho .h
- Como declarar? Através de seu protótipo:

```
0   #include <stdio.h>
1   //prototipo da funcao soma()
2   //tipo_retorno nome_funcao();
3   float soma(float, float); //ponto e virgula no final
4
5   int main() {
6       printf("%f\n", soma(1.2, 3.0));
7       return 0;
8   }
9   float soma(float a, float b) {
10      return a+b;
11  }
12
```

Exemplo

Faça uma função que leia, repetidamente até EOF, a quantidade de pessoas que entra e sai, respectivamente, de um elevador e imprima quantas pessoas restaram.

```
0   #include <stdio.h>
1   //declaracao do prototipo
2   void elevador();
3
4   int main() {
5       elevador();
6       return 0;
7   }
8
9   //definicao da funcao elevador
10  void elevador() {
11      int entra, sai, e=0;
12      while( scanf("%d%d", &entra, &sai) != EOF ) {
13          e = e + entra - sai;
14      }
15      printf("%d\n", e);
16  }
```

Parâmetros: passando valores

- **Passagem de argumentos**: **a função pode receber valores externos**;

```
0    tipo_retorno nome_funcao ( lista de parametros ) {
1
2    }
```

- **Lista de parâmetros**:
  - **Lista de tipos e nomes de variáveis** separados por vírgulas
  - Função chamadora: passa os **argumentos**(valores)

```
0    void imprime() {
1        printf("Ola mundo!\n");
2    }
3
4    void soma(int a, int b) {
5        printf("%d\n", a+b);
6    }
7
8    int quad(int n) {
9        return n*n;
10   }
```

Parâmetros: declarando o protótipo

```
0    tipo_retorno nome_funcao ( lista de parametros );
```

```
0    //prototipos
1    void imprime();              //procedimentos
2    void soma(int a, int b); //ou void soma(int, int);
3    int quad(int n);             //ou int quad(int);
```

```
0    int quad(int n) {
1        return n*n;
2    }
3    void imprime() {
4        printf("Ola mundo!\n");
5    }
6    void soma(int a, int b) {
7        int c=a+b;
8        printf("%d\n", c);
9    }
```

Passagem de argumentos por valor

- **Cópia** do valor original
- Não altera a variável original
- Variável da função $\neq$ variável original

```c
void teste (char k) {
    printf("%c\n", k); //podemos utilizar o valor de k
    k='a';             //podemos alterar o valor de k
    printf("%c\n", k); //a
}

int main() {
    teste('p'); //passando o valor direto

    char k = 'b';

    teste(k); //passando uma copia do valor de k
    printf("%c\n", k); //b - nao altera o valor de k

    return 0;
}
```

Passagem de argumentos por referência

- **Passar o endereço (referência) de uma variável**
- As **alterações** são feitas **diretamente no endereço** da variável original
- Ponteiro:
  - Variável especial que armazena endereços
  - Identificado pelo * (asterisco)
  - Armazena o endereço do local onde está o conteúdo

Passagem de argumentos por referência

```c
void f1(int , int *);

int main(){
    int a=1, c=0;

    f1(a, &c); //conteudo de a e endereco de c
    printf("%d %d\n", a, c); //1 4
    return 0;
}

void f1(int x, int *i){
    printf("%d %d\n", x, *i); //1 0
    *i = 4; //alterando o conteudo do endereco apontado por i
    x = 5;  //alterando o conteudo da variavel x
}

```

Passagem de argumentos por referência

- **Array**(vetor, matriz, string) aponta para o **endereço da primeira posição**:
  - Passar para uma função = **passar seu endereço**;

```
0  //prototipo
1  void teste(int [], int);
2
3  int main() {
4      int v1[4], i;
5      teste(v1, 4);
6      for(i=0; i<4; i++)
7          printf("%d ", v1[i]);
8      printf("\n");
9      return 0;
10 }
11
12 void teste(int v[], int n) {
13     for(int i=0; i<n; i++)
14         v[i]=0;
15 }
```

Passagem de argumentos por referência

```
0  //prototipos
1  void vetor1(int [], int);
2  void matriz(int [][4], int);
3  void string1(char []);
4
5  void vetor1(int v[], int tam) {
6      for(int i=0; i<tam; i++) scanf("%d", &v[i]);
7  }
8
9  //passando matriz -> colunas obrigatorio
10 //eh preciso informar a quantidade de colunas
11 //pois a alocacao de memoria eh sequencial
12 //assim as colunas indicam quantos espacos devem ser "pulados"
13 //ate a proxima linha
14 void matriz(int a[][4], int i) {
15     for(int t=0; t<i; t++)
16         for(int l=0; l<i; l++) scanf("%d", &a[t][l]);
17 }
18
19 void string1(char s[]) {
20     for(int t=0; s[t]!='\0'; t++) s[t] = 'a';
21 }
```

Resumo

- Passagem por cópia:
  - Passar somente um valor/conteúdo;
  - Alterar o parâmetro não altera a variável original.
- Passagem por referência:
  - Passar um endereço (ponteiro ou array);
  - Alterar o parâmetro = alterar a variável original;

Passagem de argumentos por valor: struct

```
0   struct endereco{ char rua[50]; int num; };
1
2   void imprimir_endereco(struct endereco);
3   struct endereco ler_endereco();
4
5   int main() {
6       struct endereco e;
7       e = ler_endereco();
8       imprimir_endereco(e);
9       return 0;
10  }
11  //retornar struct
12  struct endereco ler_endereco() {
13      struct endereco ender;
14      scanf("%s", ender.rua);
15      scanf("%d", &ender.num);
16      return ender;
17  }
18  //por valor = somente conteudo
19  void imprimir_endereco(struct endereco ender) {
20      printf("%s\n", ender.rua);
21      printf("%d\n", ender.num);
22      ender.num = 20; //nao altera a original
23  }
```

Passagem de argumentos por referência: struct

```
0   struct endereco{ char rua[50]; int num; };
1
2   void imprimir_endereco(struct endereco *);
3   void ler_endereco(struct endereco *);
4
5   int main() {
6       struct endereco e;
7       ler_endereco(&e);
8       imprimir_endereco(&e);
9       return 0;
10  }
11
12  //por referencia = endereco original
13  void imprimir_endereco(struct endereco *ender) {
14      //troca . por ->
15      printf("%s\n", ender->rua);
16      printf("%d\n", ender->num);
17  }
18
19  void ler_endereco(struct endereco *ender) {
20      //troca . por ->
21      scanf(" %[^\n]", ender->rua);
22      scanf("%d", &ender->num);
23  }
```

Roteiro

Bibliotecas - Visão geral

- **Definição**:
  - Conjunto de funções agrupadas;
  - Compartilhamento de funcionalidades;

- **Utilização**:
  - Inclusão do arquivo cabeçalho (.h): contém os protótipos de cada função;
  - Biblioteca padrão da linguagem C (também conhecida como libc): cada compilador C possui sua implementação da biblioteca padrão C;
  - Exemplos:
    - ★ math.h Funções matemáticas comuns em computação. Ex.: sqrt, sin, cos, tan, ceil
    - ★ stdio.h Manipulação de entrada/saída. Ex.: printf, scanf
    - ★ stdlib.h Diversas operações, incluindo conversão, geração de números pseudo-aleatórios, alocação de memória. Ex.: abs, rand, malloc
    - ★ string.h Tratamento de cadeia de caracteres.

Biblioteca padrão - string.h

Há funções para manipulação de string já definidas na biblioteca padrão C:

- **strlen**: tamanho de uma string;
- **strcmp**: comparar duas strings;
- **strcpy**: copiar uma string em outra;
- **man string**: outras funções

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

int main() {
    char s1[11] = "alo";
    int i = strlen(s1); //3 caracteres

    char s2[11] = "ala";
    i = strcmp(s1, s2); //0 se iguais
                        //ou a subtracao da primeira diferenca
                        //'o'-'a' = 111-97 = 14
                        //se s1>s2 retorna positivo
                        //se s1<s2 retorna negativo
    char s3[11];
    strcpy(s3, s1); //copia s1 em s3
    strcpy(s3, "ola");
    return 0;
}
```

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);

```
0 #ifndef _TESTE1_H //se prototipos ainda nao foram definidos
1 #define _TESTE1_H //definir
2
3 int soma (int, int);
4 void soma_vetor (float [], float [], int);
5
6 #endif
7
```

2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:
   ./meuprograma

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);

```c
#include <stdio.h>  //biblioteca padrao
#include "teste1.h" //biblioteca propria

int soma (int a, int b)
{
    return a+b;
}

void soma_vetor (float v1[], float v2[], int n)
{
    int i;
    for(i=0; i<n; i++)
        v2[i] = v1[i]+v2[i];
}

```

3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto criado:

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:
   ./meuprograma

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):

```
0  #include <stdio.h>
1  #include "teste1.h"
2
3  int main() {
4      float a[2]={1,2}, b[2]={3,2};
5      int c;
6
7      c = soma(1, 2);
8      soma_vetor(a, b, 2);
9
10     printf("%d %f %f\n", c, b[0], b[1]);
11
12     return 0;
13 }
14
```

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:
   ./meuprograma

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:
   ./meuprograma

Roteiro

Arquivos

- Um arquivo (= file) é uma sequência de bytes que reside na memória secundária (ex., HD)
- Manipular um arquivo: variável do tipo **FILE**
- O tipo FILE é predefinido em stdio.h

```
0     FILE *fp;
1
```

Arquivos

Operações sobre arquivos

- Associando/abrindo um arquivo:

```
0          FILE *fp1, *fp2;
1          fp1 = fopen("data.txt", "r");
2          fp2 = fopen("imagem.pgm", "w");
3
```

  - O primeiro argumento da função é o nome do arquivo;
  - O segundo argumento é modo que será aberto o arquivo:
    - ★ "r": somente leitura (do início do arquivo);
    - ★ "r+": leitura e escrita (do início do arquivo);
    - ★ "w": escrita, apagando o conteúdo do arquivo ou criando (do início do arquivo);
    - ★ "w+": leitura e escrita, apagando o conteúdo do arquivo ou criando (do início do arquivo);
    - ★ "a": escrita, criando se não existir, posicionando no fim do arquivo;
    - ★ "a+": leitura e escrita, criando se não existir, posicionando no fim do arquivo;

Arquivos

```
0     FILE *fp;
1     fp = fopen("data.txt", "r");
2     if(fp==NULL)
3          printf("Erro ao abrir o arquivo\n");
4     else
5          printf("Retornou o endereco do arquivo\n");
6
```

- Observação:
  - O teclado é o arquivo padrão de entrada (= standard input);
  - Ele está permanente aberto e é representado pela constante **stdin**;
  - A saída padrão também é um arquivo e é representando pela constante **stdout**.

Arquivos

Processando arquivo de texto (caracteres)

- Ler ou escrever dados:
  - um caracter por vez, usando as funções da biblioteca padrão fgetc() e fputc();
  - uma linha por vez, usando fgets() e fputs();
  - em um formato específico, usando fscanf() e fprintf();
- Vamos nos limitar ao uso do fscanf e fprintf;
- Dúvidas: man

Arquivos

- A função fscanf() lê o conteúdo do arquivo representado por fp;
- É preciso informar:
  - Qual arquivo será lido;
  - Quais tipos de dados serão lidos através dos especificadores de formato: %d, %c, %f, %s;
  - Em quais variáveis os dados lidos serão gravados.

```
0     FILE *fp;
1     fp = fopen("data.txt", "r");
2     int i, r;
3     char s[11], f[100];
4     if(fp!=NULL) {
5         r = fscanf(fp, "%d %s", &i, s);
6         r = fscanf(fp, "%99[^\n]", f);
7     }
```

Arquivos

- Valor retornado:
  - EOF se o final do arquivo foi atingido ou em caso de erro;
  - Em caso de sucesso: a quantidade de itens convertidos e atribuídos.

```
0     while(fscanf(fp, "%d %s", &i, s) != EOF)
1         printf("%d %s\n", i, s);
2
```

- fscanf (stdin, ...) equivale a scanf(...)

Arquivos

- fprintf(arquivo, ≪mensagem¿")
- fprintf(arquivo, ≪especificador de formato¿", variaveis)

```
0     FILE *t;
1     t = fopen("teste", "w");
2     fprintf(t, "Ola mundo\n");
3
4     int a=10;
5     char b[]="lala";
6     fprintf(t, "%d %s\n", a, b);
7
```

- Retorno: número de caracteres escritos, ou negativo em caso de ocorrência de erros.
- fprintf(stdout, ...) equivale ao printf(...)

Arquivos

- Fechando um arquivo:
  - Função fclose

```
0         FILE *fp;
1         fp = fopen("data.txt", "r");
2         ...
3         fclose(fp);
4
```

  - Retorna 0 no sucesso e EOF no erro:

```
0         FILE *fp;
1         fp = fopen("data.txt", "r");
2         ...
3         if(fclose(fp)==EOF)
4             printf("Erro ao fechar o arquivo\n");
5
```