Edwar Saliba Júnior

Dicas, Comandos e Exemplos Comparativos entre Linguagem Algorítmica e Linguagem C

Belo Horizonte
2010

Sumário

1 Nota:

Os códigos em Linguagem C apresentados neste trabalho foram criados utilizando a ferramenta conhecida por "Code::Blocks" versão "10.05". Esta ferramenta poderá ser adquirida gratuitamente no site abaixo:

http://download.berlios.de/codeblocks/codeblocks-10.05mingw-setup.exe

2 Comandos e Palavras Reservadas:

Como mecanismo facilitador, vejamos a tabela abaixo relativa a alguns comandos básicos e palavras reservadas existentes em algoritmos, e respectivamente em Linguagem C.

Comando em Algoritmo	Comando em Linguagem C
início	{
fim	}
declare	(não existe)
Real	float
inteiro	int
carácter	char
escreva	printf
leia	scanf
se	if
senão	else
caso	switch ... case
enquanto	while
para	for
repita ... enquanto	do ... while
←	=

Alguns subcomandos mais utilizados:

Caracteres especiais	Significado
\n	Salto de linha.
\0	Nulo.
\r	Retorno do carro.
\\	Barra invertida.

Operadores:

Operador	Símbolo
Adição	+
Subtração	-
Divisão	/
Multiplicação	*
Módulo (resto da divisão inteira)	%

No capítulo 4 serão apresentados diversos problemas, seus algoritmos e seus respectivos programas em linguagem C.

3 Dicas

3.1 Strings em Linguagem C

3.1.1 Criando Variáveis do Tipo *String*:

Para campos de texto, deixar sempre o número exatos de caracteres para o armazenamento do valor em questão.

Por exemplo: Tomemos um campo para guardar data com o seguinte formato:

dd/mm/aaaa

Podemos verificar que para armazenarmos a data com o formato acima, são necessários 10 posições de memória. Exemplo:

```
char data[10];
```

3.1.2 Problema: Em Tempo de Execução o Programa está Pulando Dois ou Mais Campos ao Mesmo Tempo:

Ao trabalharmos com *strings* e/ou caracteres em linguagem C devemos tomar certos cuidados. A linguagem C não é muito amigável quando se trata de *strings*.

Ao usarmos o seguinte programa (Fig. 1):

*main.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int identificador;
7       char nome[50];
8       char dataNascimento[11];
9       char sexo;
10
11      printf("Digite o identificador: ");
12      scanf("%i", &identificador);
13      printf("Digite o nome: ");
14      gets (nome);
15      printf("Digite a data de nascimento: ");
16      gets (dataNascimento);
17      printf("Digite o sexo (M/F): ");
18      sexo = getchar();
19
20      printf("\n\n\n=== Os Dados Digitados Foram ===\n\n\n");
21
22      printf("Identificador  : %i ", identificador);
23      printf("\nNome           : %s ", nome);
24      printf("\nData Nascimento: %s ", dataNascimento);
25      printf("\nSexo (M/F)     : %c \n\n", sexo);
26
27      return 0;
28  }
29
```

Fig. 1 – Programa Exemplo: "Salto Indevido" de Campo do Tipo *String*

O usuário deste *software* notará que ao apertar a tecla <*Enter*> após ter digitado o número do "identificador" (linha 12), o programa saltará diretamente para o campo "dataNascimento" (linha 16) sem pedir os dados para o campo "nome" (linha 14).

Isto acontece porque ao apertar a tecla <*Enter*>, junto com o conteúdo digitado, também são enviados ao *buffer* caracteres de controle. Destes, o comando *scanf* lê o conteúdo e deixa pra trás o caractere '\n', ou seja, o *buffer* ainda tem algo. Esse caractere que ficou pra trás é lido pelo comando *gets*. O caracter '\n' é o comando <*Enter*> que nada mais é do que um "salto de linha", o que provoca o "salto" do campo em questão.

Porém, isto não acontecerá no programa abaixo:

main.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int identificador;
7       char nome[50];
8       char dataNascimento[11];
9       char sexo;
10
11      printf("Digite o identificador: ");
12      scanf("%i", &identificador);
13      fflush(stdin);
14      printf("Digite o nome: ");
15      gets (nome);
16      printf("Digite a data de nascimento: ");
17      gets (dataNascimento);
18      printf("Digite o sexo (M/F): ");
19      sexo = getchar();
20
21      printf("\n\n\n=== Os Dados Digitados Foram ===\n\n\n");
22
23      printf("Identificador  : %i ", identificador);
24      printf("\nNome           : %s ", nome);
25      printf("\nData Nascimento: %s ", dataNascimento);
26      printf("\nSexo (M/F)     : %c \n\n", sexo);
27
28      return 0;
29  }
30
```

Fig. 2 – Programa Exemplo de Utilização do Comando *fflush*

O programa mostrado na Fig. 2 é idêntico ao mostrado na Fig. 1, com apenas uma modificação, a linha 13.

O comando *fflush*, apesar de criticado por muitos programadores, ainda é o único que conheço que funciona em DOS (Windows), e funciona muito bem! Na seção 3.1.4.3 comento sobre a polêmica de usar ou não o comando *fflush* e as críticas dos programadores em relação a este comando.

Bem, voltando ao nosso problema, este comando foi feito para limpar o *buffer* de saída, no nosso caso, estamos utilizando-o para limpar o *buffer* do teclado (*buffer* de entrada), após ter ocorrido o evento de "apertar da tecla *<Enter>*" do comando *scanf* na linha 12. Este polêmico comando garantirá que nosso programa funcione corretamente, pois, ele eliminará o '\n' que sobra no *buffer*, após a leitura e remoção de quase todos os dados do *buffer* do teclado pelo comando *scanf*.

Sempre antes de usar um comando para leitura de caracteres ou *strings* via teclado (o comando *gets*, por exemplo), escreva um comando *fflush(stdin)*, para garantir que o *buffer* estará vazio no momento da execução do comando de leitura de caracteres ou *strings*. Caso isto não seja feito, o comando tomará o que estiver no *buffer* do teclado

como sendo a *string* ou caractere digitado. Como já explicado anteriormente, o caractere que fica sobrando no *buffer* é o '\n', por isto acontece o "salto" de campo sem nenhuma digitação no programa. Veja exemplo na Fig. 01.

3.1.3 Como Apagar um Campo *String*:

Uma *string* na mais é do que uma sequência de caracteres. Podemos declarar um campo *string* da seguinte forma:

```
char nome[50];
```

A variável 'nome', nada mais é do que um vetor de caracteres.

Uma *string* em linguagem C deve ser finalizada pelo carácter '\0' (barra invertida zero). Que é o carácter que representa fim de *string* na linguagem C. Seu uso é indispensável quando se faz uso de ponteiros na declaração de *strings*.

Portanto, sabendo disto, para apagarmos uma *string* basta atribuirmos o carácter de fim de *string* para a primeira posição do vetor. Exemplo:

```
nome[0] = '\0';
```

Lembrando que todo vetor em linguagem C começa na posição 0 (zero) e vai até a quantidade reservada menos 1 (um). No nosso caso, o vetor "nome" vai da posição 0 (zero) até a posição 49 (quarenta e nove), o que nos dá 50 posições de memória.

3.1.4 Como Atribuir Valores a Uma Variável do Tipo *String*:

As variáveis do tipo 'char' têm tratamento especial pela linguagem C. Para atribuirmos valores a variáveis deste tipo, precisamos utilizar alguns comandos especiais.

Vejamos a seguir alguns exemplos de atribuição de valores para variáveis do tipo **char**.

3.1.4.1 Exemplo de leitura e escrita de um caractere:

Para ler e imprimir um único caractere, não há tratamento especial, podemos tratar a variável do tipo *char* como se fosse uma variável do tipo *int*, ou seja, os comandos são idênticos aos que usamos para trabalhar com valores do tipo inteiro.

Vejamos no exemplo adiante (Fig. 3), um programa onde são criadas duas variáveis do tipo *char* (Letra1 e Letra2), sendo que a variável Letra1 recebe um valor digitado pelo usuário e a variável Letra2 recebe seu

valor internamente. Após receberem seus valores, ambas são impressas na tela.

Podemos observar também (Fig. 3), que ao atribuirmos um valor a uma variável do tipo *char*, este valor deverá estar entre aspas simples e, quando formos ler o valor do teclado, poderemos usar o comando *scanf* normalmente.

main.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       char Letra1, Letra2;
7
8       Letra2 = 'E';
9
10      printf("Digite uma letra qualquer: ");
11      scanf("%c", &Letra1);
12
13      printf("\nA letra digitada foi: %c", Letra1);
14      printf("\nA letra atribuida a variável internamente foi: %c \n\n", Letra2);
15
16      return 0;
17  }
18
```

Fig. 3 – Exemplo de Leitura e Escrita de Um Caractere

3.1.4.2 Exemplo de leitura e escrita de vários caracteres:

Na Fig. 4 podemos observar um programa onde foram criadas três variáveis do tipo *char* de 30 posições de memória cada. As três variáveis são: Palavra1, Palavra2 e Palavra3.

Conforme podemos observar, o programa atribui internamente através da função *strcpy* as palavras "Belo Horizonte - 1" a variável Palavra1. Em seguida é solicitado que o usuário do software entre com mais duas palavras via teclado separadas por espaço em branco (esta operação se repete, na primeira vez utilizando-se o comando *scanf* e na segunda vez utilizando-se o comando *gets*). Observe que a variável Palavra2 receberá o valor digitado do teclado através do comando *scanf*, e a variável Palavra3 receberá o valor digitado do teclado através do comando *gets*.

Ao executarmos o código do programa (Fig. 4), podemos observar (Fig. 5) que o comando *scanf* não trabalha bem com "espaços", em se tratando de manipulação de *strings*, pois, foi digitado "Belo Horizonte – 2", porém o comando *scanf* só armazenou na variável "Palavra2" a primeira

palavra que foi digitada antes de haver um espaço, ou seja, "Belo". Já no comando *gets* este problema não ocorre.

main.c

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

int main()
{
    char Palavra1 [30], Palavra2 [30], Palavra3 [30];

    /* Comando de atribuição para String. */
    strcpy(Palavra1, "Belo Horizonte");

    printf("Digite duas palavras quaisquer, separadas por espaço: ");
    scanf("%s", &Palavra2);
    fflush(stdin);

    printf("Digite duas palavras quaisquer, separadas por espaço: ");
    gets(Palavra3);

    printf("\nA palavra atribuída à variável Palavra1 foi: %s ", Palavra1);
    printf("\nAs palavras digitadas na variável Palavra2 foram: %s ", Palavra2);
    printf("\nAs palavras digitadas na variável Palavra3 foram: %s \n\n", Palavra3);

    return 0;
}

```

Fig. 4 – Exemplo de Programa de Leitura e Atribuição de Palavras

```
C:\Private\Eddie\Empresas\Pitagoras\Apostilas\Programacao\Linguagem_C_Dicas\CodigosEmC_Co...
Digite duas palavras quaisquer, separadas por espaco: Belo Horizonte - 2
Digite duas palavras quaisquer, separadas por espaco: Belo Horizonte - 3

As palavras atribuidas a variavel Palavra1 foram: Belo Horizonte - 1
As palavras digitadas na variavel Palavra2 foram: Belo
As palavras digitadas na variavel Palavra3 foram: Belo Horizonte - 3


Process returned 0 (0x0)   execution time : 12.681 s
Press any key to continue.
```

Fig. 5 – Execução do Programa de Leitura e Atribuição de Palavras com Espaço

3.1.4.3 Comando fflush x (fpurge ou __fpurge)

O comando *fflush* como já mencionado anteriormente, não é bem visto aos olhos dos programadores C para limpar o *buffer* de entrada. Isto quando utilizado para programação em sistemas operacionais como FreeBSD, Linux, Unix, MacOS dentre outros. Não lhes tiro a razão; visto que, originalmente este comando foi criado para atualização (limpeza) do *buffer* de saída.

Ao consultarmos a biblioteca padrão de C[1], descobriremos que o comando *fflush*, segundo a documentação, pode causar um efeito imprevisível no código, quando usado para limpar *buffers* que não são de saída. Bem, para os sistemas operacionais não Windows, recomenda-se o uso do comando *fpurge(stdin)* ou *__fpurge(stdin)* ao invés do comando *fflush(stdin)*. Para o Windows, ainda não descobri como fazer o comando *fpurge* ou mesmo o *__fpurge* funcionar. E, por outro lado, já faz algum tempo que utilizo e também vejo diversos programadores utilizar o comando *fflush* para limpeza do *buffer* de entrada com sucesso (no sistema operacional Windows).

Então, se você está utilizando o Windows, eu recomendo que utilize o *fflush* para limpeza do *buffer* de entrada. Caso você esteja utilizando um sistema operacional diferente do Windows, sugiro que opte por *fpurge* ou *__fpurge*.

Algumas referências a seguir desmotivam o uso do *fflush* para limpeza do *buffer* de entrada. Veja:
<http://www.velocityreviews.com/forums/t698367-fpurge-fflush.html>,
<http://groups.google.com.br/group/cefetmgedc/msg/480a6585354f12cd>,
< http://cboard.cprogramming.com/c-programming/76326-newbie-question-fflush-stdin-fpurge-stdin-mac-pc.html>,
<http://www.guiadohardware.net/comunidade/limpar-buffer/739172/>,
<http://faq.cprogramming.com/cgi-bin/smartfaq.cgi?answer=1052863818&id=1043284351>.

Apesar de todas estas referências desmotivarem o uso do *fflush* para limpeza do *buffer* de entrada, nenhuma delas ensina como fazê-lo de outra forma; principalmente em Windows. Todas estas referências criticam, mas nenhuma aponta uma solução, ou melhor, algumas até apontam soluções paliativas (também conhecidas vulgarmente como: gambiarra ou, pra ficar mais bonito, recurso técnico alternativo), como colocar um comando *getchar* após um *scanf* ou mesmo alterar a parametrização do *scanf* para que ele despreze o caractere '\n'. Isto pode até funcionar. Contudo, eu ainda sugiro colocar o *fflush* para limpeza do *buffer* de entrada em ambiente Windows, e se você algum dia descobrir como utilizar o *fpurge*, *__fpurge* ou qualquer outro comando que seja realmente feito para a limpeza do *buffer* de entrada (em ambiente Windows), não se esqueça de me avisar, pois, farei questão de passar a utilizá-lo.

[1] ***C Standard Library***. Disponível em: <http://www.utas.edu.au/infosys/info/documentation/C/CStdLib.html> Acesso em: 12 set. 2010.

4 Exemplos de Algoritmos em Linguagem Algorítmica e Seu Respectivo Código em Linguagem C

4.1 *Algoritmo usando estrutura LINEAR*

```
inicio
     declare val1, val2, soma : real;

      escreva "Digite o primeiro valor: ";
      leia val1;
      escreva "Digite o segundo valor: ";
      leia val2;

      soma ← val1 + val2;

      escreva "O total da soma é: ", soma;
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

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

int main()
{
    float val1, val2, soma;

    printf("Digite o primeiro valor: ");
    scanf("%f", &val1);
    printf("Digite o segundo valor: ");
    scanf("%f", &val2);

    soma = val1 + val2;

    printf("\n\nO total da soma e: %f \n\n", soma);

    return 0;
}
```

4.2 *Algoritmo usando estrutura condicional SE*

Elabore um algoritmo para ler 4 notas de um aluno (no intervalo de 1 a 10). Após a leitura, calcular a média das notas e apresentar a mensagem: "aprovado" se o aluno tiver obtido média maior ou igual a 6, caso contrário, apresentar "reprovado".

```
início
      declare n1, n2, n3, n4, media : real;

      escreva "Digite o valor da primeira nota entre 1 e 10: ";
      leia n1;
      escreva "Digite o valor da segunda nota entre 1 e 10: ";
      leia n2;
      escreva "Digite o valor da terceira nota entre 1 e 10: ";
      leia n3;
      escreva "Digite o valor da quarta nota entre 1 e 10: ";
      leia n4;

      media ← (n1 + n2 + n3 + n4) / 4;

      se (media >= 6) então
            escreva "O aluno foi APROVADO!";
      senão
            escreva "O aluno foi REPROVADO!";
      fim se
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

```
==================================================

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

int main()
{
    float n1, n2, n3, n4, media;

    printf("Digite o valor da primeira nota entre 1 e 10: ");
    scanf("%f", &n1);
    printf("Digite o valor da segunda nota entre 1 e 10: ");
    scanf("%f", &n2);
    printf("Digite o valor da terceira nota entre 1 e 10: ");
    scanf("%f", &n3);
    printf("Digite o valor da quarta nota entre 1 e 10: ");
    scanf("%f", &n4);

    media = (n1 + n2 + n3 + n4) / 4;

    if(media >= 6){
        printf("\n\nO aluno foi APROVADO!\n\n");
    }
    else{
        printf("\n\nO aluno foi REPROVADO!\n\n");
    }

    return 0;
}
```

4.3 Algoritmo usando estrutura de exclusão múltipla CASE

Faça um algoritmo que mostre o menu de opções a seguir, receba a opção do usuário e os dados necessários para executar cada operação. No final, apresente o resultado da operação escolhida.

```
Menu de opções:
1 – Somar
2 – Subtrair
3 – Multiplicar
4 - Dividir

início
      declare valor1, valor2, res : real;
      declare operacao : inteiro;

      escreva "=== Calculadora de 4 Operações Básicas ===";

      escreva "Digite o 1°. valor: ";
      leia valor1;
      escreva "Digite o 2°. valor: ";
      leia valor2;

      escreva "Escolha a operação a ser realizada sobre os valores:";
      escreva "1 - Somar";
      escreva "2 - Subtrair";
      escreva "3 - Multiplicar;
      escreva "4 - Dividir;
      escreva "Escolha uma opção: ";
      leia operacao;

      caso (operacao) faça
            1 : início
                     res ← valor1 + valor2;
                fim
            2 : início
                     res ← valor1 - valor2;
                fim
            3 : início
                     res ← valor1 * valor2;
                fim
            4 : início
                    se (não (valor2 = 0)) então
                         res ← valor1 / valor2;
                     senão
                         escreva "Erro: Divisão por zero.";
                         res ← 0;
                     fim se
                fim
            exceção : início
                             escreva "Opção escolhida é inválida!";
                        fim
      fim caso
```

```
      se ((operacao >= 1) e (operacao <= 4)) então
            escreva "O resultado final é: ", res;
      fim se
fim

O algoritmo acima descrito em linguagem C, abaixo:
==================================================

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

int main()
{
    float valor1, valor2, res = 0;
    int operacao;

    printf("=== Calculadora de 4 Operações Básicas ===");

    printf("\n\nDigite o 1o. valor: ");
    scanf("%f", &valor1);
    printf("Digite o 2o. valor: ");
    scanf("%f", &valor2);

    printf("\n\nEscolha a operação a ser realizada sobre os
valores:");
    printf("\n\n1 - Somar");
    printf("\n2 - Subtrair");
    printf("\n3 - Multiplicar");
    printf("\n4 - Dividir");
    printf("\n\nEscolha uma opção: ");
    scanf("%d", &operacao);

    switch (operacao) {
        case 1:
             res = valor1 + valor2;
             break;
        case 2:
             res = valor1 - valor2;
             break;
        case 3:
             res = valor1 * valor2;
             break;
        case 4:
             if(! (valor2 == 0)){
                 res = valor1 / valor2;
             }
             else{
                 printf("\n\nErro de divisão por zero!\n\n");
             }
            break;
        default:
             printf("\n\nOpção escolhida é inválida!\n\n");
    }

    if((operacao >= 1) && (operacao <= 4)){
```

```
        printf("\n\nO resultado final é: %f \n\n", res);
    }

    return 0;
}
```

4.4 Algoritmo usando estrutura de repetição ENQUANTO

Elabore um algoritmo que solicite que o usuário entre com dois números (inicial e final). O algoritmo deverá apresentar o valor total da soma de todos os números do intervalo digitado pelo usuário.

```
início
     declare valini, valfin, soma : real;

     escreva "Digite o valor inicial: ";
     leia valini;
     escreva "Digite o valor final: ";
     leia valfin;

     soma ← valini;

     enquanto (valini <= valfin) faça
          valini ← valini + 1;
          soma ← soma + valini;
     fim enquanto

     escreva "A soma total é: ", soma;
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

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

int main()
{
    int valini, valfin, soma;

    printf("Digite o valor inicial: ");
    scanf("%d", &valini);
    printf("Digite o valor final: ");
    scanf("%d", &valfin);

    soma = valini;

    while (valini < valfin) {
          valini = valini + 1;
          soma = soma + valini;
    }

    printf("\n\nA soma total é: %d \n\n", soma);
```

```
    return 0;
}
```

4.5 *Algoritmo usando estrutura de repetição REPITA ... ENQUANTO*

Elabore um algoritmo que mostre ao usuário um *menu* com 4 opções: 1 – Inclusão, 2 – Exclusão, 3 – Alteração e 4 – Sair. Obrigue o usuário a escolher uma das opções. Por fim, emita uma mensagem notificando o usuário da opção escolhida.

```
início
     declare opcao : inteiro;

     repita
          escreva "Opções";
          escreva "1 - Inclusão";
          escreva "2 - Exclusão";
          escreva "3 - Alteração;
          escreva "4 - Sair;
          escreva "Escolha uma opção: ";
          leia opcao;
     enquanto ((opcao < 1) ou (opção > 4));

     caso (opcao) faça
          1 : início
                   escreva "A opção escolhida foi: Inclusão.";
              fim
          2 : início
                   escreva "A opção escolhida foi: Exclusão.";
              fim
          3 : início
                   escreva "A opção escolhida foi: Alteração.";
              fim
          4 : início
                   escreva "A opção escolhida foi: Sair.";
              fim
     fim caso
fim
```

O algoritmo acima descrito em linguagem C, abaixo:
==========

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

int main()
{
    int opcao;

    do {
        printf("\n\nOpções");
```

```
        printf("\n\n1 - Inclusão");
        printf("\n2 - Exclusão");
        printf("\n3 - Alteração");
        printf("\n4 - Sair");
        printf("\n\nEscolha uma opção: ");
        scanf("%d", &opcao);
    } while ((opcao < 1) || (opcao > 4));

    switch(opcao){
        case 1:
             printf("\n\nA opção escolhida foi: Inclusão.\n\n");
             break;
        case 2:
             printf("\n\nA opção escolhida foi: Exclusão.\n\n");
             break;
        case 3:
             printf("\n\nA opção escolhida foi: Alteração.\n\n");
             break;
        case 4:
             printf("\n\nA opção escolhida foi: Sair.\n\n");
    }

    return 0;
}
```

4.6 *Algoritmo usando estrutura de repetição PARA*

Elabore um algoritmo que solicite que o usuário entre com um número inteiro qualquer. Após a entrada do número, o algoritmo deverá imprimir na tela os 30 números subsequentes ao número digitado.

```
início
     declare num, cont : inteiro;

     escreva "Digite um número inteiro: ";
     leia num;

     para cont de 1 até 30 passo 1 faça
          escreva num + cont;
     fim para
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

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

int main()
{
    int num, cont;

    printf("Digite um número inteiro: ");
```

```
    scanf("%d", &num);

    for (cont = 1; cont <= 30; cont++) {
        printf("\n %d", num + cont);
    }
    printf("\n\n");

    return 0;
}
```

4.7 *Algoritmo usando VETOR*

Elabore um algoritmo que solicite que o usuário entre com 10 números inteiros quaisquer. Após a entrada de dados, imprimir os números digitados na ordem inversa a da digitação.

```
início
     declare num[10], i : inteiro;

     para i de 0 até 9 passo 1 faça
          escreva "Digite um número: ";
          leia num[i];
     fim para

     escreva "Os números digitados, na ordem inversa a da
digitação:";

     para i de 9 até 0 passo -1 faça
          escreva "O número digitado foi: ", num[i];
     fim para
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

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

int main()
{
    int num[10], i;

    for (i = 0; i < 10; i++) {
        printf("Digite o %do. valor: ", i + 1);
        scanf("%d", &num[i]);
    }

    printf("\n\nOs números digitados, na ordem inversa a da
digitação:\n");

    for (i = 9; i >= 0; i--) {
        printf("\nO %do. valor digitado foi: %d", i + 1, num[i]);
    }
```

```
    return 0;
}
```

4.8 Algoritmo usando MATRIZ

Elabore um algoritmo que solicite que o usuário entre com 16 números inteiros quaisquer. Após a entrada de dados, imprimir o quadrado dos números digitados na ordem inversa a da digitação.

```
início
     declare num[4][4], i, j : inteiro;

     para i de 0 até 3 passo 1 faça
          para j de 0 até 3 passo 1 faça
               escreva "Digite um número: ";
               leia num[i][j];
          fim para
     fim para

     escreva "O quadrado dos números digitados, na ordem inversa a da
digitação:";

     para i de 3 até 0 passo -1 faça
          para j de 3 até 0 passo -1 faça
               escreva (num[i][j] * num[i][j]);
          fim para
     fim para
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

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

int main()
{
    int num[4][4], i, j;

    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            printf("Digite o um número: ");
            scanf("%d", &num[i][j]);
        }
    }

    printf("\n\nO quadrado dos números digitados, na ordem inversa a
da digitação:\n");

    for (i = 3; i >= 0; i--) {
        for (j = 3; j >= 0; j--) {
            printf("\n %d ", (num[i][j] * num[i][j]));
```

```
        }
    }

    printf("\n\n");

    return 0;
}
```

4.9 *Algoritmo usando REGISTRO (tipo)*

Elabore um algoritmo para cadastrar um funcionário. Para o funcionário devem-se cadastrar os seguintes dados: código, nome, data de admissão, data de nascimento e sexo. Após cadastrar o funcionário, mostrar os dados.

```
tipo funcionario
     codigo : inteiro;
     nome : texto;
     dataAdmissao : texto;
     dataNascimento : texto;
     sexo : carácter;
fim tipo;

início
     declare func : funcionario;
     declare resp : carácter;

     escreva "Digite o código do funcionário: ";
     leia func.codigo;
     escreva "Digite o nome do funcionário: ";
     leia func.nome;
     escreva "Digite a data de admissão do funcionário: ";
     leia func.dataAdmissao;
     escreva "Digite a data de nascimento do funcionário: ";
     leia func.dataNascimento;
     escreva "Digite o sexo do funcionário (M/F): ";
     leia func.sexo;

     escreva "Deseja visualizar os dados cadastrados? (S/N): ";
     leia resp;

     se (resp = 's') então
         escreva "=== Dados do funcionário ===";
         escreva "Código: ", func.codigo;
         escreva "Nome : ", func.nome;
         escreva "Data de Admissão: ", func.dataAdmissao;
         escreva "Data de Nascimento: ", func.dataNascimento;
         escreva "Sexo : ", func.sexo;
     fim se
fim
```

O algoritmo acima descrito em linguagem C, abaixo:
==========

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

struct funcionario {
       int codigo;
       char nome[50];
       char dataAdmissao[11];
       char dataNascimento[11];
       char sexo;
};

int main()
{
      struct funcionario func;
      char resp;

      printf("Digite o código do funcionário: ");
      scanf("%d", &func.codigo);
      fflush(stdin);  /* fflush - Limpa o buffer do teclado. */
      printf("Digite o nome do funcionário: ");
      gets (func.nome);
      printf("Digite a data de admissão do funcionário: ");
      gets (func.dataAdmissao);
      printf("Digite a data de nascimento do funcionário: ");
      gets (func.dataNascimento);
      printf("Digite o sexo do funcionário (M/F): ");
      scanf("%c", &func.sexo);
      fflush(stdin);

      printf("\n\nDeseja visualizar os dados cadastrados? (S/N): ");
      resp = getche();  /* Lê o valor digitado sem a necessidade da
tecla <Enter> ser pressionada. */
      fflush(stdin);

      if ((resp == 's') || (resp == 'S')) {
          printf("\n\n=== Dados do funcionário ===");
          printf("\nCódigo: %d", func.codigo);
          printf("\nNome : %s", func.nome);
          printf("\nData de Admissão: %s", func.dataAdmissao);
          printf("\nData de Nascimento: %s", func.dataNascimento);
          printf("\nSexo : %c \n\n", func.sexo);
     }

     return 0;
}
```

4.10 Algoritmo usando REGISTRO em VETOR

Elabore um algoritmo para cadastrar 1000 funcionários. Para cada funcionário devem-se cadastrar os seguintes dados: código, nome, data de admissão, data de nascimento e sexo. Após cadastrar o(s) funcionário(s), mostrar os dados.

```
tipo funcionario
      codigo : inteiro;
      nome : texto;
      dataAdmissao : texto;
      dataNascimento : texto;
      sexo : carácter;
fim tipo;

início
      declare func[1000] : funcionario;
      declare resp : carácter;
      declare cont, aux_cont : inteiro;

      cont ← 0;
      resp ← 's';

      enquanto ((cont < 1000) e (resp = 's')) faça
            escreva "Digite o código do funcionário: ";
            leia func[cont].codigo;
            escreva "Digite o nome do funcionário: ";
            leia func[cont].nome;
            escreva "Digite a data de admissão do funcionário: ";
            leia func[cont].dataAdmissao;
            escreva "Digite a data de nascimento do funcionário: ";
            leia func[cont].dataNascimento;
            escreva "Digite o sexo do funcionário (M/F): ";
            leia func[cont].sexo;

            cont ← cont + 1;

            escreva "Deseja cadastrar outro funcionário? (S/N): ";
            leia resp;
      fim enquanto

      resp ← 'n';
      escreva "Deseja visualizar o(s) cadastro(s) realizado(s)? (S/N):
";
      leia resp;

      se (resp = 's') então
            aux_cont ← 0;

            enquanto (aux_cont < cont) faça
                  escreva "=== Dados do funcionário ===";
                  escreva "Código: ", func[aux_cont].codigo;
                  escreva "Nome : ", func[aux_cont].nome;
                  escreva "Data de Admissão: ",
func[aux_cont].dataAdmissao;
                  escreva "Data de Nascimento: ",
func[aux_cont].dataNascimento;
                  escreva "Sexo : ", func[aux_cont].sexo;

                  aux_cont ← aux_cont + 1;
            fim enquanto
```

```
     fim se
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

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

struct funcionario {
       int codigo;
       char nome[50];
       char dataAdmissao[11];
       char dataNascimento[11];
       char sexo;
};

int main()
{
    struct funcionario func[1000];
    char resp;
    int cont, aux_cont;

    cont = 0;
    resp = 's';

    while ((cont < 1000) && ((resp == 's') || (resp == 'S'))) {
           printf("\n\nDigite o código do funcionário: ");
           scanf("%d", &func[cont].codigo);
           fflush(stdin);
           printf("Digite o nome do funcionário: ");
           gets (func[cont].nome);
           printf("Digite a data de admissão do funcionário: ");
           gets (func[cont].dataAdmissao);
           printf("Digite a data de nascimento do funcionário: ");
           gets (func[cont].dataNascimento);
           printf("Digite o sexo do funcionário (M/F): ");
           scanf("%c", &func[cont].sexo);
           fflush(stdin);

           cont++;   /* cont = cont + 1; */

          printf("\n\nDeseja cadastrar outro funcionário? (S/N): ");
           resp = getche();
    }

    resp = 'n';
    printf("\n\nDeseja visualizar os dados cadastrados? (S/N): ");
    resp = getche();
    fflush(stdin);

    if ((resp == 's') || (resp == 'S')) {
        aux_cont = 0;

        while (aux_cont < cont) {
```

```
                printf("\n\n=== Dados do funcionário ===");
                printf("\nCódigo: %d", func[aux_cont].codigo);
                printf("\nNome : %s", func[aux_cont].nome);
                printf("\nData de Admissão: %s",
func[aux_cont].dataAdmissao);
                printf("\nData de Nascimento: %s",
func[aux_cont].dataNascimento);
                printf("\nSexo : %c \n\n", func[aux_cont].sexo);

                aux_cont++;
        }
    }

    return 0;
}
```

4.11 *Algoritmo usando REGISTRO em MATRIZ*

Elabore um algoritmo para cadastrar 1200 funcionários numa matriz 30 x 40. Para cada funcionário devem-se cadastrar os seguintes dados: código, nome, data de admissão, data de nascimento e sexo. Após cadastrar o(s) funcionário(s), mostrar os dados.

```
tipo funcionario
     codigo : inteiro;
     nome : texto;
     dataAdmissao : texto;
     dataNascimento : texto;
     sexo : carácter;
fim tipo;

início
     declare func[30][40] : funcionario;
     declare resp : carácter;
     declare x, y, aux_x, aux_y : inteiro;

     resp ← 's';
     y ← 0;
     enquanto ((y < 30) e (resp = 's')) então
          x ← 0;
          enquanto ((x < 40) e (resp = 's')) faça
               escreva "Digite o código do funcionário: ";
               leia func[y][x].codigo;
               escreva "Digite o nome do funcionário: ";
               leia func[y][x].nome;
               escreva "Digite a data de admissão do funcionário: ";
               leia func[y][x].dataAdmissao;
               escreva "Digite a data de nascimento do funcionário:
";
               leia func[y][x].dataNascimento;
               escreva "Digite o sexo do funcionário (M/F): ";
               leia func[y][x].sexo;
```

```
                x ← x + 1;

                escreva "Deseja cadastrar outro funcionário? (S/N):
";
                leia resp;
           fim enquanto

           y ← y + 1;
     fim enquanto

     resp ← 'n';
     escreva "Deseja visualizar o(s) cadastro(s) realizado(s)? (S/N):
";
     leia resp;

     se (resp = 's') então
          aux_y ← 0;
          enquanto (aux_y < y) faça
                aux_x ← 0;
                enquanto (aux_x < x) faça
                     escreva "=== Dados do funcionário ===";
                     escreva "Código: ", func[aux_y][aux_x].codigo;
                     escreva "Nome : ", func[aux_y][aux_x].nome;
                     escreva "Data de Admissão: ",
func[aux_y][aux_x].dataAdmissao;
                     escreva "Data de Nascimento: ",
func[aux_y][aux_x].dataNascimento;
                     escreva "Sexo : ", func[aux_y][aux_x].sexo;

                     aux_x ← aux_x + 1;
                fim enquanto

                aux_y ← aux_y + 1;
           fim enquanto
     fim se
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

```
==================================================

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

struct funcionario {
       int codigo;
       char nome[50];
       char dataAdmissao[11];
       char dataNascimento[11];
       char sexo;
};

int main()
{
    struct funcionario func[30][40];
    char resp;
```

```
    int x, y, aux_x, aux_y;

    resp = 's';
    y = 0;
    while ((y  < 30) && ((resp == 's') || (resp == 'S'))) {
          x = 0;
          while ((x < 40) && ((resp == 's') || (resp == 'S'))) {
                 printf("\n\nDigite o código do funcionário: ");
                 scanf("%d", &func[y][x].codigo);
                 fflush(stdin);
                 printf("Digite o nome do funcionário: ");
                 gets (func[y][x].nome);
                 printf("Digite a data de admissão do funcionário:
");
                 gets (func[y][x].dataAdmissao);
                 printf("Digite a data de nascimento do funcionário:
");
                 gets (func[y][x].dataNascimento);
                 printf("Digite o sexo do funcionário (M/F): ");
                 func[y][x].sexo = getchar();
                 fflush(stdin);

                 x++;   /* Equivale a:  x = x + 1; */

                printf("\n\nDeseja cadastrar outro funcionário?
(S/N): ");
                 resp = getche();
          }

          y++;
    }

    resp = 'n';
    printf("\n\nDeseja visualizar os dados cadastrados? (S/N): ");
    resp = getche();
    fflush(stdin);

    if ((resp == 's') || (resp == 'S')) {
        aux_y = 0;
        while (aux_y < y) {
              aux_x = 0;
              while (aux_x < x) {
                     printf("\n\n=== Dados do funcionário ===");
                     printf("\nCódigo: %d",
func[aux_y][aux_x].codigo);
                     printf("\nNome : %s", func[aux_y][aux_x].nome);
                     printf("\nData de Admissão: %s",
func[aux_y][aux_x].dataAdmissao);
                     printf("\nData de Nascimento: %s",
func[aux_y][aux_x].dataNascimento);
                     printf("\nSexo : %c \n\n",
func[aux_y][aux_x].sexo);

                     aux_x++;
              }
```

```
                aux_y++;
            }
        }

        return 0;
}
```

4.12 Algoritmo usando FUNÇÃO

Elabore um algoritmo para simular uma calculadora de 4 operações básicas.

```
função adicao (v1, v2 : real) : real;
      declare total : real;

      total ← v1 + v2;

      retorne (total);
fim função

função subtracao (v1, v2 : real) : real;
      declare total : real;

      total ← v1 - v2;

      retorne (total);
fim função

função multiplicacao (v1, v2 : real) : real;
      declare total : real;

      total ← v1 * v2;

      retorne (total);
fim função

função divisao (v1, v2 : real) : real;
      declare total : real;

      total ← 0;

      se (v2 <> 0) então
            total ← v1 / v2;
      senão
            escreva "Erro de divisão por zero!";
      fim se

      retorne (total);
fim função

início
      declare valor1, valor2, res : real;
```

```
      declare operacao : inteiro;

      escreva "=== Calculadora de 4 Operações Básicas ===";

      escreva "Digite o 1°. valor: ";
      leia valor1;
      escreva "Digite o 2°. valor: ";
      leia valor2;

      escreva "Operação a ser realizada sobre os valores:";
      escreva "1 - Somar";
      escreva "2 - Subtrair";
      escreva "3 - Multiplicar;
      escreva "4 - Dividir;
      escreva "Escolha uma opção: ";
      leia operacao;

      caso (operacao) faça
            1 : início
                     res ← adicao (valor1,valor2);
                fim
            2 : início
                     res ← subtracao (valor1,valor2);
                fim
            3 : início
                     res ← multiplicacao (valor1,valor2);
                fim
            4 : início
                     res ← divisao (valor1,valor2);
                fim
            exceção : início
                           escreva "Opção escolhida é inválida!";
                      fim
      fim caso

      se ((operacao >= 1) e (operacao <= 4)) então
            escreva "O resultado final é: ", res;
      fim se
fim
```

O algoritmo acima descrito em linguagem C, abaixo:

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

float adicao (float v1, float v2){
      float total;

      total = v1 + v2;

      return (total);
}

float subtracao (float v1, float v2){
```

```
      float total;

      total = v1 - v2;

      return (total);
}

float multiplicacao (float v1, float v2){
      float total;

      total = v1 * v2;

      return (total);
}

float divisao (float v1, float v2){
      float total;

      total = 0;

      if (v2 != 0) {
         total = v1 / v2;
      }
      else {
           printf("\n\nErro de divisão por zero!");
      }

      return (total);
}

int main()
{
    float valor1, valor2, res;
    int operacao;

    printf("\n=== Calculadora de 4 Operações Básicas ===");

    printf("\n\nDigite 1o. valor: ");
    scanf("%f", &valor1);
    printf("Digite 2o. valor: ");
    scanf("%f", &valor2);

    printf("\n\nOperação a ser realizada sobre os valores:");
    printf("\n\n1 - Somar");
    printf("\n2 - Subtrair");
    printf("\n3 - Multiplicar");
    printf("\n4 - Dividir");
    printf("\n\nEscolha uma opção: ");
    scanf("%d", &operacao);

    switch (operacao) {
        case 1:
             res = adicao (valor1,valor2);
             break;
        case 2:
```

```
                res = subtracao (valor1,valor2);
                break;
            case 3:
                res = multiplicacao (valor1,valor2);
                break;
            case 4:
                res = divisao (valor1,valor2);
                break;
            default:
                printf(" \n\nOpção escolhida é inválida!\n\n");
    }

    if ((operacao >= 1) && (operacao <= 4)) {
        printf("\n\nO resultado final é: %f \n\n", res);
    }

    return 0;
}
```