Linguagem C

1 - Introdução

História:

- criada por Dennis Ritchie em 1972 na Bell Laboratories
- 1ª utilização importante – reescrita do Unix (que era escrito em Assembly)
- década de 80, várias versões do compilador C oferecidas por várias empresas
- 1983 – ANSI C (American National Standard Institute) – C padrão com pouca diferença do C original (define também bibliotecas)

Vantagens:

- eficiência (na geração de código)
- muito conhecida e usada
- pode ser utilizada para desenvolver leque amplo de programas – desde Sistemas Operacionais, editores até aplicações de usuário
- pode ser usada em varias plataformas embora tenha sido originalmente desenvolvida para Unix
- linguagem enxuta, fácil de usar

2- Exemplos de Programas Simples em C

Um programa em C consiste de funções e variáveis:

1. Função
   Contém comandos que especificam operações a serem realizadas.
   Todo programa em C tem pelo menos uma função chamada *main* que é obrigatória.
   A execução do programa inicia pelo primeiro comando da função *main*.

2. Variáveis
   Local na memória onde um valor pode ser armazenado e recuperado posteriormente.

2.1 Programa que escreve "bom dia"

```
/* programa que escreve bom dia na tela */

#include <stdio.h>        /* para compilador incluir informação sobre biblioteca padrão de entrada e saída */

main ( )                  /* declara função principal */

{
        printf("bom dia\n");
}
```

Esse programa escreve bom dia na tela do computador.

Os textos entre /* e */ são comentários para usuários e ignorados pelo compilador.

A chamada da função printf da biblioteca stdio.h escreve uma seqüência de caracteres entre aspas na saída padrão (tela). \n representa o caráter para mudança de linha.

Os comandos da função main são especificados entre chaves { }.

2.2 Programa que realiza uma operação aritmética e escreve resultado na tela

Um valor obtido após uma operação aritmética pode ser armazenado em uma variável. Variáveis devem ser declaradas antes de ser usadas. Por ex.:

```
int b;
```

Essa declaração aparece geralmente no início da função e indica que uma localização de memória chamada b pode armazenar um valor inteiro. A variável declarada tem um nome (b) e um tipo (int).

Para armazenar um valor na variável, o comando de atribuição é utilizado:

```
b = 5;            /* armazena valor 5 na variável b */
```

Para mostrar o valor da variável na tela, a função printf da biblioteca stdio.h é utilizada:

```
printf("%d", b);
```

O formato %d indica que o valor a ser exibido é um inteiro decimal.

Exemplo de programa que utiliza variáveis:

```
#include <stdio.h>
main ()
{
        int a,b,c;              /* declara 3 variáveis inteiras */
        a = 5;                  /* atribui valor 5 para a variável a */
        b = 7;
        c = a + b;              /* atribui valor da expressão 12 para a variável c */
        printf("%d + %d = %d\n" a,b,c);
}
```

2.3 Programa que obtém um valor de entrada do teclado

Para obter um valor da entrada padrão (teclado) e armazenar em um variável, a função scanf da biblioteca stdio.h é utilizada:

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

Com esse comando o programa lê um valor digitado pelo usuário no teclado e armazena na variável a. São especificados o formato do valor (inteiro decimal) e o endereço da variável (indicado pelo símbolo &).

Exemplo de programa com entrada pelo teclado:

```
#include <stdio.h>
main ()
{
        int a,b,c;

        printf("entre com o valor:");
        scanf("%d",&a);                 /* leitura do primeiro valor inteiro em a */
        printf("entre com outro valor:");
        scanf("%d",&b);                 /* leitura do segundo valor inteiro em b */
        c = a + b;
        printf("%d + %d = %d\n", a,b,c);
}
```

3- Comando de Atribuição

3.1 Variáveis e tipos

Nomes de variáveis:

- primeiro caráter deve ser letra ou sublinhado "_"
- os 32 primeiros caracteres são significativos
- letras maiúsculas são diferentes de letras minúsculas
- tradicionalmente usa-se minúsculo para variáveis e maiúsculas para constantes
- não podem ser usadas palavras reservadas como if, else, int, float, etc
- nomes devem ser escolhidos conforme o objetivo da variável

Tipos de variáveis: uma variável está sempre associada a um tipo que determina os valores que podem ser armazenados na variável e as operações que podem ser executadas sobre ela.

Tipos inteiros (tamanho dependente da máquina)

- char – 1 byte no intervalo [0,128)
- signed char – 1 byte no intervalo (-128,128)
- unsigned char – 1 byte no intervalo (0,256)
- short – 2 bytes no intervalo (-2^{-15}, 2^{15})
- unsigned short – 2 bytes no intervalo[0, 2^{16})
- int – 2 ou 4 bytes no intervalo (2^{-15}, 2^{15}) ou (2^{-31}, 2^{31}) respectivamente
- unsigned int – 2 ou 4 bytes no intervalo [0, 2^{16}) ou [0, 2^{32}) respectivamente
- long – 4 bytes no intervalo (2^{-31}, 2^{31})
- unsigned long - 4 bytes no intervalo [0, 2^{32})

Tipos flutuantes

- float – pelo menos 6 dígitos de precisão decimal – 4 bytes
- double – pelo menos 10 dígitos decimais de precisão – 8 bytes
- long double – pelo menos 10 dígitos decimais precisão – 12 bytes

Para saber o número de bytes ocupados por um variável, pode-se chamar a função sizeof.

```
main()
{
        unsigned char c;
        printf("%d", sizeof(c));
}
```

Para saber o valor máximo e mínimo de um certo tipo, pode-se utilizar as constantes definidas nas bibliotecas limits.h para inteiros (INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, etc) e float.h para ponto flutuante.

Exemplos de declarações:

```
int i;
int j= 0;
float pi = 3.14;
float eps = 1.0 e-5;
int k=1, n=2;
```

3.2 Expressões e comandos de atribuição aritméticos

Operadores aritméticos:

- +
- -
- *
- / divisão inteira caso ambos operandos sejam inteiros e real caso contrário
- % módulo (resto de divisão inteira)

Operadores para incremento e decremento:

- ++x incrementa x antes de utilizar seu valor
- x++ incrementa x depois de utilizá-lo
- --x
- x--

Precedência de operadores:

- ++ -- (mais alta prioridade)
- / % *
- + -

Comando de atribuição:

```
i = i + 2;
```

O comando de atribuição calcula a expressão à direita do operador = e atribui à variável i à esquerda. Esse comando pode ser escrito também como:

```
i += 2;
```

Essa notação pode ser utilizada também com os outros operadores (-, *, /, %).

Exemplo de comandos de atribuição:

```
i =20;
x*= y+1;          // equivale a x=x*(y+1)
```

4. Funções Básicas de Entrada e Saída

4.1 Comando de saída

printf – escreve no dispositivo padrão (tela)

Sintaxe:

```
printf("expressão de controle", argumentos);
```

A expressão de controle pode conter caracteres que serão exibidos na tela e/ou códigos de formatação para os argumentos. Os argumentos são separados por vírgula.

Exemplo de alguns caracteres especiais (são precedidos por \) que podem ser incluídos na expressão de controle:

- \n nova linha
- \t tab
- \b retrocesso
- \" aspas
- \\ barra

Exemplo de formatos que podem ser incluídos na expressão de controle:

- %c para caráter
- %d para inteiros em decimal
- %6d para inteiro decimal com pelo menos 6 caracteres
- %f para ponto flutuante
- %6f para ponto flutuante com pelo menos 6 caracteres
- %.2f com 2 caracteres depois do ponto decimal
- %6.2f com 6 caracteres no total e 2 após o ponto
- %o octal
- %x hexa
- %s cadeia
- %u decimal sem sinal

4.2 Comando de entrada

scanf() – lê dados da entrada padrão (teclado)

Sintaxe:

```
scanf("expressão de controle", argumentos);
```

A expressão de controle especifica o formato dos argumentos. Os argumentos são separados por vírgula e precedidos pelo operador & (para endereço).

4.3 Exercícios

Escreva programas para:

1. calcular o quadrado de um número lido.
2. transformar uma temperatura em ºF (dado de entrada) para ºC.
3. calcular a área de um retângulo sendo que os lados são lidos do teclado.
4. calcular a área de um círculo após leitura do seu raio.
5. calcular a média de um aluno dadas as notas de 2 provas.
6. calcular o salário de um empregado dados horas trabalhadas e salário-hora.
7. transformar polegadas em centímetros.

5 – Comandos Condicionais

5.1 Expressões condicionais

Uma expressão condicional é considerada falsa se seu valor for zero e verdadeira se diferente de zero.

Operadores relacionais:

- >
- >=
- <
- <=
- = = testa igualdade
- != testa desigualdade

Operadores lógicos:

- && (and)
- || (or)
- ! (not)

Precedência de operadores:

- ( ) mais alta prioridade
- !, ++, --, + (unário), – (unário), &, sizeof
- *, /, %
- +, - (binário)
- <, <=, >, >=
- = =, !=
- &&
- ||
- =, +=, -=, *=, /=, %= menor prioridade

5.2 Comando if

Sintaxe:

```
if (expressão condicional)
        comando1;
else
        comando2;
```

Se a expressão for verdadeira (qualquer valor ≠ 0) comando1 é executado. Se falsa (= 0) comando2 é executado.

A parte iniciada por else é opcional.

comando1 ou comando2 podem ser comandos simples ou blocos de comandos entre chaves{ }.

Exemplos de comandos if:

```
if (x % 2)
        printf("impar \n");
else
        printf("par \n");

if (nota > =0 && nota <= 10 )
        printf("válida \n");
else
        printf("inválida \n");
```

5.3 Operador ternário

Sintaxe:

```
Condição ? exp1 : exp2
```

É uma forma compacta de if – else.

Exemplo:

```
maior = (a > b) ? a: b;
```

Equivale ao comando if abaixo:

```
if (a > b)
        maior = a;
else
        maior = b;
```

5.4 Comando switch

5.5 Exercícios

Escreva programas para:

1. dados dois números inteiros a e b, verificar se a é divisor de b.
2. calcular salário de um empregado dados horas trabalhadas e salário-hora. Para salários maiores que R$1000,00 existe um desconto de 8%.
3. calcular a média de um aluno de mc102 que realiza duas provas com mesmo peso e um exame caso a média das provas seja menor que 5.
4. calcular as raízes de uma equação do segundo grau, dados a, b e c.

5. verificar se os valores a, b e c lidos podem ser lados de um triângulo e qual triângulo (eqüilátero, isósceles, escaleno).
6. verificar se os valores a, b e c lidos formam um triângulo retângulo.

OBS: você vai precisar de funções da biblioteca de matemática math.h que contém pow(x,y), cos(x), sin(x), tan(x), exp(x), log(x), sqrt(x), fabs(x), entre outras.

6 – Comandos de Repetição

6.1 Comando for

Sintaxe:

```
for (exp1; exp2; exp3) comando;
```

exp1 é um comando de atribuição para inicializar variável de controle do laço
exp2 é uma expressão condicional que testa a variável de controle para determinar quando o laço terminará e o controle passe para o comando seguinte ao for
exp3 define a alteração na variável de controle para cada iteração

comando pode ser vazio, simples ou um bloco de comandos.

Exemplo:

```
for (i = 1, i < 10, i++)                    /* imprime 1,2,...,9 */
        printf("%d \n", i);

for (;;;)
        printf("infinito\n");
```

Exercícios:

1. Ler 20 números e escrever os números pares somente.
2. Ler um inteiro n e escrever o quadrado e o cubo para inteiros de 2 a n.
3. Ler n números inteiros e escrever quantos são pares e quantos são ímpares.
4. Ler n números e escrever sua somatória.
5. Encontrar a média da classe com n alunos.
6. Ler n números e encontrar o maior deles.
7. Para os números inteiros de 1000 a 9999, imprimir os que tem a característica:
   $30 + 25 = 55$
   $55^2 = 3025$
8. Escrever as potências de 2 de 0 a n (n lido).
9. Calcular o fatorial de um número inteiro lido.
10. Dados x e n, ambos inteiros, calcular x^n usando multiplicações.
11. Imprimir os n primeiros termos da seqüência de Fibonacci (1, 1, 2, 3, 5, 8, 13, 21 ...).
12. Calcular m^3, $m \geq 1$ como soma dos m ímpares consecutivos começando pelo ímpar $m(m - 1) + 1$. Por exemplo:
    $m = 5$
    $m(m-1) + 1 = 21$
    $m^3 = 21 + 23 + 25 + 27 + 29 = 125$
13. Somar todos os divisores de um número inteiro lido.
14. Verificar se um número lido é primo.

O comando for pode ser usado dentro de outro comando for. Por exemplo:

```
for (linha = 1; linha <= 3; linha++)
{
        for(coluna = 1; coluna < 5; coluna++)
                printf("%3d %3d", linha, coluna);
        printf("\n");
}
```

Esse trecho de programa escreve na tela:

```
 1  1  1  2  1  3  1  4
 2  1  2  2  2  3  2  4
 3  1  3  2  3  3  3  4
```

Exercícios com for encadeado:

1. Ler n e escrever n linhas como o desenho abaixo (para n=3)

```
*
* *
* * *
```

2. Ler n e escrever n linhas como o desenho abaixo (para n=3).

```
* * *
* *
*
```

3. Ler n e escrever n linhas como o desenho abaixo (para n=3).

```
* * *
* *
*
```

4. Ler n e imprimir n linhas como as linhas abaixo (para n=3).

```
1
2 2
3 3 3
```

5. Ler n e escrever n linhas como as linhas abaixo (para n=3).

```
1
2 1
3 2 1
4 3 2 1
```

6. Ler n e escrever n linhas como as linhas abaixo (para n=3).

```
1
2 3
4 5 6
7 8 9 10
```

7. Ler n e escrever as tabuadas de 2 a n.
8. Verificar quais os primos de 2 a n.

6.2 Comando while

Sintaxe:

```
while (condição) comando1;
```

A expressão condicional entre parênteses é avaliada e, se verdadeira, o comando comando1 é executado. Esse comando pode ser executado de 0 a n vezes.

comando1 pode ser vazio, simples ou bloco.

O comando while pode ser equivalente a um comando for. Por exemplo:

```
for (exp1; exp2; exp3) comando;
```

equivale a

```
exp1;
while (exp2){
        comando;
        exp3;
}
```

Exemplo:

```
/* somatória de 10 números lidos */
soma = 0;
for (i=1; i<=10; i++)
{
        scanf("%d",&n);
        soma += n;
}
```

equivale ao código abaixo:

```
/* somatória de 10 números lidos */
soma = 0;
i = 1;
while (i <= 10)
{
        scanf("%d",&n);
        soma += n;
        i++;
}
```

Exercícios:

1. Ler números inteiros até encontrar zero e calcular sua somatória.
2. Contar número de dígitos de um número inteiro lido.

3. Verifique se um número inteiro n é um quadrado perfeito. Um número n é quadrado perfeito se a soma dos m ímpares consecutivos (1 + 3 + 5 +...) é igual a n. Nesse caso, o número m de termos ímpares somados será a raiz quadrada de n. Por exemplo:

$16 = 1 + 3 + 5 + 7$

$\sqrt{16} = 4$

4. Imprimir os termos da seqüência de Fibonacci até encontrar o primeiro termo maior que um limite lido.

6.3 Comando do while

Sintaxe:

```
do
{
        comando;
}
while (condição);
```

O comando entre chaves é executado enquanto a condição for verdadeira. Se a condição for falsa, o controle passa para o comando seguinte ao do – while. O comando entre chaves é executado pelo menos uma vez.

Exemplo:

```
/* somatória de 10 números lidos */
soma = 0;
for (i = 1; i <= 10; i++)
{
        scanf("%d",&n);
        soma += n;
}
```

equivale a

```
/* somatória de 10 números lidos */
soma = 0;
i = 1;
do
{
        scanf("%d",&n);
        soma += n;
        i++;
} while (i <= 10);
```

Exercícios:

1. Calcular o máximo divisor comum de dois inteiros x, y lidos pelo método de Euclides.

2. Faça um programa que lê as coordenadas x, y dos vértices de um polígono e que calcule seu perímetro. Os vértices devem ser lidos até que o último vértice seja igual ao primeiro. A distância entre 2 pontos é calculada como:
$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

6.4 Comandos break e continue

O comando break é utilizado para forçar a saída de um laço.

Exemplo:

```
/*  sai do laço se número lido for negativo */
soma = 0;
for (i = 0; i < n; i++)
{
        scanf("%d",&x);
        if(x < 0)
                break;
        soma += x;
}
```

O comando continue ignora os comandos seguintes dentro do laço e passa para a próxima iteração do laço.

Exemplo:

```
/* se número lido for negativo não inclui na somatória e lê próximo número */
soma = 0;
for (i = 0; i < n; i++) {
        scanf("%d",&x);
        if (x < 0)
                continue;
        soma += x;
}
```

7 – Funções

Uma função é uma seqüência de passos que recebe um nome e pode ser invocada uma ou mais vezes durante a execução do programa.

7.1 Motivação

Funções são utilizadas para:

- estruturação dos programas: permite melhor estruturação facilitando a compreensão do programa.
- reutilização de código: uma função é escrita para realizar uma determinada tarefa. Por exemplo, calcular o fatorial de um número. O código para o cálculo do fatorial aparece uma única vez no programa mas a função pode ser invocada diversas vezes de pontos diferentes do programa.

7.2 Tipos de funções

Em C, existem as *funções pré-definidas* cujo código foi escrito previamente e está disponível em uma biblioteca. Por exemplo, a função sqrt calcula a raiz quadrada de um número passado como parâmetro. Essa função está disponível na biblioteca math.h e pode ser invocada dentro de uma expressão. Exemplo de invocação:

```
raiz = (-b + sqrt(delta))/(2*a);
```

Além das funções pré-definidas, existem as *funções definidas pelo programador*. Para esse tipo de função, o programador deve especificar um nome, parâmetros, tipo e código.

Exemplo de definição de função:

```
/* define uma função do tipo inteira, nome maior e dois parâmetros inteiros
   essa função retorna o maior de dois números inteiros
*/
int maior (int x, int y)
{
        if (x > y)
                return(x);
        else
                return(y);
}
```

Obs.: função que não retorna nenhum valor deve ser declarada como void.

Exemplo de invocação de função:

```
/* programa que calcula o maior de três números */
main ()
{
```

```
        int x, y, z, m;
        scanf( "%d %d %d", &x, &y, &z);
        m = maior (x, y);        // primeira invocação
        m = maior (m,z);         // segunda invocação
}
```

Uma chamada para a função provoca um desvio para o corpo da função, os parâmetros formais (x e y) são substituídos pelos reais (x e y) na primeira invocação e por (m e z) na segunda invocação. Após a execução da função, o controle volta para o ponto da chamada.

7.3 Exercícios

Escreva funções para:

1. determinar se um número passado como parâmetro é primo. A função deve retornar 1 se for primo e 0 caso contrário. Usando essa função, escreva programa que imprime todos os números primos de 2 a n (n lido).
2. calcular o fatorial de um número passado como parâmetro.
3. calcular x^n através de multiplicações. Tanto x (float) como n (int) são parâmetros da função.
4. calcular a distância entre dois pontos (x_1, y_1) e (x_2, y_2) passados como parâmetros.
5. calcular o binômio de Newton para n e r passados como parâmetros:

$C(n, r) = n! / ((n – r)! * r!)$

utilizando a função fatorial.

6. calcular a integral de uma função como somatória das áreas de n retângulos.

Os parâmetros para a função são o limite inferior do intervalo a, limite superior do intervalo b e o número de retângulos n no intervalo. Considere a função

$f(x) = \sqrt{(1 – x^2)}$

7. calcular o MMC entre dois números:

$MMC = (n_1 * n_2) / MDC$

utilizando a função MDC que retorna o MDC entre n_1 e n_2.

7.4 Passagem de parâmetros

Existem dois tipos de passagem de parâmetro:

- passagem por valor: na invocação da função, a expressão é avaliada e copiada em área local da função. Nesse caso, não é possível alterar os parâmetros reais. Esse tipo de passagem de parâmetro é utilizado quando se quer transmitir dados de quem invoca para a função invocada.
- passagem por referência: na invocação da função, é especificado o endereço de uma variável. Dessa forma é possível alterar os parâmetros reais. Esse tipo de passagem de parâmetro é utilizado quando se quer transmitir dados da função para quem invocou.

Exemplo:

```
// essa função tem dois parâmetros passados por valor
void f1(int x, int y)
{
        x = 0; y = 0;
        printf (“em f1: x = %d y = %d \n”, x, y);
}

// essa função tem dois parâmetros passados por referência
void f2(int *px, int *py)
{
        *px = 0; *py = 0;
        printf(“em f2: *px = %d *py = %d \n”, *px, *py);
}

main ()
{
        int x = 1;
        int y = 3;
        printf(“antes de chamar f1: x = %d y = %d \n”, x, y);
        f1(x, y);
        printf(“depois de chamar f1: x = %d y = %d \n”, x, y);
        f2(x, y);
        printf(“depois de chamar f2: x = %d y = %d \n”, x, y);
}
```

O resultado impresso na tela será:

antes de chamar f1: x = 1 y = 3
em f1: x = 0 y = 0
depois de chamar f1: x = 1 y = 3
em f2: x = 0 y = 0
depois de chamar f2: x = 0 y = 0

7.5 Passando funções como argumento

```
main()
{
        int i, j;
        int processa(int (*) (int, int));
        int func1(int, int);
        int func2(int, int);


        i = processa(func1);
        j = processa(func2);
}

int processa (int *pf());
{
        int a, b, c;
        c = (*pf) (a, b);
```

```
        return (c);
}

int func1(int a, int b)
{
        return ( );
}

int func2(int x, int y)
{
        return ( );
}
```

8- Vetores e Matrizes

8.1 Vetores

Uma variável do tipo vetor pode armarzenar uma coleção de valores todos do mesmo tipo. Por exemplo, uma coleção de notas dos alunos de uma classe, uma coleção de telefones de uma agenda ou uma coleção de filmes de uma vídeo-locadora.

Os elementos da coleção são armazenados em posições contíguas de memória. Cada elemento da coleção é identificado por um seletor (ou índice) que indica a posição do elemento na coleção.

Declaração:

tipo identificador[tamanho];

Exemplo de declaração:

```
int a[10];        // aloca espaço para uma coleção de 10 inteiros
                  // referenciados como a[0], a[1], ... , a[9]
```

Exemplo de um programa:

```
/* programa que calcula a média das notas de uma classe de n alunos
   e quantos alunos tiveram notas maiores que a média
*/

main()
{
        int i,n,maior;
        float media,notas[100];

        printf("entre com numero de alunos da classe:");
        scanf("%d",&n);
        media=0;

//  leitura das notas dos alunos
        for (i=0;i<n;i++) {
                printf("entre com a nota de um aluno:");
                scanf("%f",&notas[i]);
                media+=notas[i];
        }
//  cálculo da média
        media/=n;

//  cálculo do número de alunos com nota maior que a média
        maior=0;
        for(i=0;i<n:i++)
                if (notas[i]>media) maior++;
```

```
        printf("media=%10.2f\nmaiores que media=%d\n",media,maior);
}
```

Exemplo de uma função:

```
// função que retorna a somatória dos elementos de um vetor de n elementos
int somatoria(int v[ ], int n)
{
        int i, soma=0;

        for (i=0;i<n:i++)
                soma+=v[i];
        return soma;
}
```

Inicialização de um vetor

Pode ser feita na declaração do vetor. Exemplo:

```
int dia[ ] = { 31, 28, 21, 30 };        // declara um vetor com 4 inteiros
                                        // para dia[0] é atribuído o valor 31
                                        // para dia[1] é atribuído o valor 28
```

Exercícios

Escreva trechos de programa para:

1. Contar quantos elementos de um vetor com n elementos são pares.
2. Somar os n elementos de um vetor.
3. Calcular o maior elemento de um vetor e sua posição.

Escreva funções para:

1. Inverter os elementos de um vetor.
2. Buscar um elemento em um vetor não ordenado.
3. Buscar um elemento em um vetor ordenado.
4. Dados dois vetores A e B de n e m elementos, construir um terceiro vetor com intersecção dos vetores A e B.
5. Dados dois vetores A e B de n e m elementos, construir um terceiro vetor com união de A e B.
6. Inserir um elemento em um vetor ordenado. Após inserção, o vetor deve continuar ordenado.
7. Dados dois vetores de n elementos, verificar se são ortogonais.
8. Dados um polinômio de ordem n cujos coeficientes estão armazenados em um vetor, verificar se um determinado valor x passado como parâmetro é uma raiz desse polinômio.

8.2 Matrizes

Uma variável do tipo matriz tem dois seletores: um para linha e outro para coluna.

Declaração:

tipo identificador[tamanho1][tamanho2];

Exemplo de declaração:

```
float notas[40][15];        // armazena as 15 notas de lab de uma classe com 40 alunos
```

Exemplo de uma função utilizando matriz:

```
// função que calcula a soma de duas matrizes de n linhas e m colunas

void soma(int A[  ][10], int B[ ][10], int C[ ][10], in n, int m)
{
        int i,j;

        for (i=0;i<n:i++)
                for (j=0;j<m;j++)
                        C[i][j] = A[i][j] + B[i][j];
}
```

Exercícios

Escreva funções para:

1.Verificar se duas matrizes A_{nxm} e B_{nxm} são iguais.
2. Determinar a soma dos elementos da diagonal principal de uma matriz A.
3. Determinar o maior elemento de uma matriz A_{nxm} e sua posição (linha e coluna).
4. Determinar a transposta de A_{nxn}.
5. Determinar se A_{nxn} é uma matriz probabilidade. Uma matriz probabilidade obedece as seguintes condições:
   a. Todos os elementos são maiores ou iguais a zero.
   b. A soma de cada linha é 1.
   c. A soma de cada coluna é 1.
6. Calcular a multiplicação de A_{nxm} por B_{mxp}. A matriz resultante será C_{nxp} onde

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$

7. Calcular o triângulo de Pascal para um dado n e armazenar em uma matriz. Uma linha i é calculada como:

$P_{i1} = 1$
$P_{ii} = 1$
$P_{ij} = P_{i-1j} + P_{i-1j-1}$

9 – Apontadores

Apontadores podem ser usados para:

- criar estruturas de dados dinâmicas
- passar parâmetro para funções (por referência)
- acessar informação em vetores

Declaração:

```
tipo     *nome;
```

é variável que contém o endereço de outra variável.

Exemplo:

```
int x, int y, *px;          // x e y – variáveis inteiras, px é apontador para inteiro

x = 3;
px = &x;                    // variável px contém endereço de x (aponta para x)
y = *px;                    // equivale a y = x;
*px = 0;                    // atribui 0 à variável x
```

Exercício 1 - Qual valor impresso?

```
main ( )
{
        int x,y,*px;
        x = 3;
        px = &x;
        y = *px + 5;
        printf("y = %d \n", y);
}
```

Exercício 2 - Qual valor impresso?

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

Exercício 3 - Qual valor impresso?

```
main ( )
{
        int x, *px, *py;
```

```
        x = 9;
        px = &x;
        py = &x;
        printf (" %d, %d, %d \n", x, *px, *py);
}
```

9.1 Expressões com apontadores

- operações aritméticas (* tem maior prioridade que operações aritméticas)

```
int x, z, *px;

x = 2;
px = &x;
z = *px + 1;       /* z recebe 2 + 1 */
```

- atribuições

```
int *px, *py, y;

py = &y;
px = py;
```

- comparações

com operadores = =, !=, >, <

- incremento, decremento dos endereços

```
int x, *px;

px = &x;
(*px)++;          // equivale a x++
```

9.2 Apontadores e Vetores

Declara vetor com 5 inteiros armazenados de forma contígua na memória:

```
int a[5];
```

Declara um apontador para inteiro:

```
int *pa;
```

Apontador pa recebe endereço do primeiro elemento do vetor a:

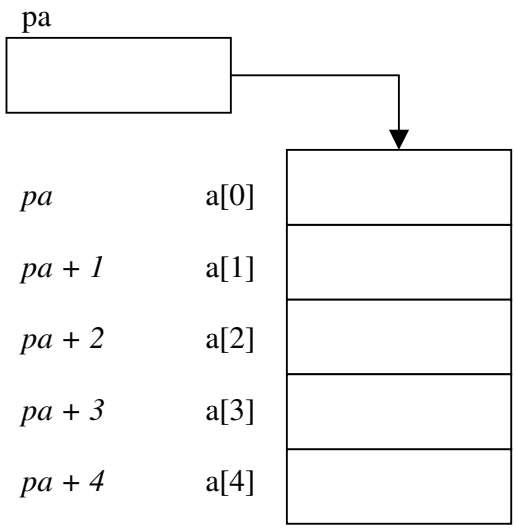
```
pa = &a[0];
```

O comando abaixo também atribui o endereço do primeiro elemento do vetor a para pa:

```
pa = a;
```

O comando abaixo atribui a[0] para variável inteira x:

```
int x = *pa;        /* equivale a x = a[0] */
```



São equivalentes:

```
a[i]    e   *(a + i)
&a[i]   e   a + i
pa[i]   e   *(pa + i)
```

Exemplo:

```
// Copia elementos do vetor a em vetor b

main ( )
{
        int a[10], b[10];
        int *p, *q;

        p = a; q = b;
        for (i = 0; i < 10; i++)
        {
                *q = *p;
                q++;
                p++;
        }
}
```

10 – Alocação Dinâmica

Funções em <stdlib.h>:

- void *malloc(n)
  retorna ponteiro para um bloco de memória de n bytes não inicializado. Retorna NULL se não houver espaço suficiente.

- void free(void *p)
  desaloca espaço apontado por p. p deve ser ponteiro para bloco previamente alocado por malloc.

Exemplo 1:

```
// aloca espaço para 40 inteiros, inicializa inteiros com valores 0 a 39.

main ( )
{
        int *p, t;
        p = (int *) malloc (40 * sizeof (int));
        if(!p)
                printf("memória insuficiente \n");
        else
        {
                for (t = 0; t < 40; t++)
                        *(p + t) = t;
                for (t = 0; t < 40; t++)
                        printf("%d ", *(p + t));
                free(p)
        }
}
```

Exemplo 2:

```
// acha o maior de n números

main ( )
{
        int i, n;
        float max, *p;
        printf("quantidade de números: ");
        scanf("%d", &n);
        p = (float *) malloc (n * sizeof(float));
        if (!p)
        {
                printf("sem memória ");
                exit(1);
        }
        printf("digite números: ");
```

```
        for(i = 0; i < n; i++)
                scanf("%f", (p + i));
        max = *p;
        for (i = 1; i < n; i++)
                if(*(p+i) > max)
                        max = *(p + 1);
        printf(" main = %f \n ", max);
        free (p);
}
```

Exemplo 3:

```
// soma de 2 matrizes n x m

include <stdlib.h>

void soma( int *a, int *b, int *c, int n, int m)
{
        int i;

        for (i=0; i<n*m;i++)
                *c++ = *a++ + *b++;
}

void imprime(int *a, int n, int m)
{
        int i,j;

        for (i=0;i<n;i++) {
                for (j=0;j<m;j++)
                        printf("%d",*(a+i*m+j));
                printf("\n");
        }
}

main( )
{
        int *a, *b, *c, i, j, m, n;
        scanf("%d, %d", &n, &m);
        a = (int *) malloc (n*m*sizeof(int));
        b = (int *) malloc (n*m*sizeof(int));
        c = (int *) malloc (n*m*sizeof(int));

        for (i = 0; i < n; i++)
                for (j = 0, j < n, j++)
                {
                        scanf((a + i * m + j))
                        scanf((b + i * m + j))
                }
        soma(a, b, c, n, m);
        imprime(c, n, m);
```

}

11 – Tipo char e Cadeias

11.1 Tipo char

Variável que pode guardar um caráter ASCII (uma letra, um dígito, sinais de pontuação, caracteres de controle).

Declaração

```
char c;
```

Atribuição

```
c = 'a';
```

Funções de Entrada/Saída em <stdio.h>

- getchar() – para leitura de um caráter do teclado
- putchar(c) – para mostrar um caráter na tela

Funções para manipular caracteres em <ctype.h>

- isalpha(c)
- isdigit(c)
- isspace(c)
- toupper(c)
- tolower(c)

Exemplo

```
// lê um caráter e verifica se é vogal

main()
{
        char c;
        c = getchar();
        c = toupper(c );
        if (c=='A' || c=='E' || c=='I' || c=='O' || c=='U') printf("vogal\n");
}
```

11.2 Cadeias

Guardam uma coleção de caracteres contiguos na memória. O último caráter da cadeia é '\0' para indicar o fim da cadeia.

Para armazenar essa coleção pode ser usada uma variável estática:

```
char cadeia[30];
```

Ou ainda um apontador para uma variável dinâmica:

```
char *pcadeia;
pcadeia = (char ) malloc (30*sizeof(char));
```

Constantes cadeia são listas de caracteres entre aspas. '\0' é colocado pelo compilador. Essas constantes são armazenadas em uma tabela pelo compilador e o apontador contém endereço para essa tabela.

Exemplo:

```
char *s;
s = "hello";     // s aponta para endereço de hello na tabela criada pelo compilador


char s[100];
s = "hello";     // erro de compilação
```

Exemplo de inicialização

Aloca 100 bytes contíguos na memória e inicializa com cadeia "hello". Também coloca '\0' na sexta posição.

```
char s[100] = "hello";
```

Aloca 6 bytes contíguos na memória e inicializa com cadeia "hello".

```
char s[] = "hello;
```

Funções para Entrada/Saída de cadeias em <stdio.h>

1 - gets(cadeia)

Leitura do dispositivo padrão dos caracteres até encontrar enter. Substitui enter por '\0'.

Exemplo:

```
main( )
{
        char str[80];
        gets(str);
        printf("%s", str);
}
```

2 – puts(cadeia)

Escreve cadeia na saída padrão.

Exemplo:

```
puts("mensagem");
```

Funções para manipular cadeias em <string.h>

1 – strcpy(destino, origem)
Copia origem em destino, coloca '\0' em destino, retorna destino.

Exemplo:

```
main( )
{
        char str[80];
        strcpy(str, "ola");
        puts (str);        // escreve ola na tela
}
```

2 – strcat(cadeia1, cadeia2)
Concatena cadeia1 e cadeia2 em cadeia1.
Não verifica tamanho.

Exemplo:

```
main()
{
        char cadeia[30];
        strcpy(cadeia, "Ana");
        strcat(cadeia1, "Maria");
        puts(cadeia);              // escreva AnaMaria na tela
}
```

3 – strcmp(cadeia1, cadeia2)
devolve 0 se iguais
< 0 se cadeia1<cadeia2
> 0 se cadeia1>cadeia2

4 – strlen(cadeia)
retorna tamanho de cadeia (sem contar '\0')

5 – strchr(cadeia,c)
retorna apontador para primeira ocorrência do caráter c em cadeia ou NULL se não existir.

6 – strpbrk(cadeia,ct)
retorna apontador para primeira ocorrência de qualquer caracter em ct ou nulo se não existir.

12 – Tipo Struct

É uma coleção de uma ou mais variáveis possivelmente de tipos diferentes agrupadas sob um único nome. Agrupa variáveis relacionadas a uma entidade como, por exemplo, empregado, aluno.

Declaração

```
struct ponto{
        int x;
        int y;
}
struct ponto pt;
```

ou

```
struct ponto{
        int x;
        int y;
}
struct ponto pt = {20,10};          //inicializa variáveis da estrutura
```

ou

```
struct ponto{
        int x;
        int y;
} pt;
```

ou

```
typedef struct
{
        int x;
        int y;
} tponto;
ponto pt;
```

Operador '.'

Usado para referenciar campos de uma estrutura. Exemplo:

```
pt.x
pt.y
```

Estruturas e funções

```
struct ponto{
        int x;
```

```
        int y;
}

struct ponto makepoint (int x, int y)          /* retorna valor tipo estrutura */
{
        struct ponto temp;
        temp.x = x;
        temp.y = y;
        return(temp);
}

struct ponto addpoint (struct ponto p1, struct ponto p2)
{
        p1.x += p2.x;
        p1.y += p2.y;
        return (p1);                           /* retorna soma de 2 pontos */
}
```

Uma estrutura pode ser definida dinamicamente. Exemplo:

```
struct ponto *pp;
pp = (struct ponto *) malloc(sizeof(struct ponto);
```

Para referenciar campos através de apontadores pode-se usar:

```
(*pp).x = 0;
```

ou

```
pp->y = 0;
```

Uma estrutura muito grande deve ser passada para uma função por referência. Exemplo:

```
void makepoint (ponto *pt)
{
        pt->x = 0;
        pt->y = 0
}
```

Vetores de estruturas

Exemplo de declaração:

```
struct filme{
        char titulo[30], diretor[30];
        int ano;
}vfilme[100];
```

Trecho de programa que lê a coleção de filmes e dados relacionados:

```
for (i=0;i<100;i++) {
        printf("entre com titulo:"); gets(vfilme[i].titulo);
        printf("entre com diretor:"); gets(vfilme[i].diretor);
        printf("entre com ano:"); scanf("%d", &vfilme[i].ano);
}
```

13 - Recursão

Um programa é recursivo quando invoca a si mesmo. A recursão é apropriada principalmente quando o problema a resolver ou a estrutura de dados a ser processada são definidos recursivamente.

Um programa recursivo P consiste de um teste de uma condição que, se verdadeira, o problema pode ser resolvido sem outras chamadas recursivas. Se a condição for falsa existe um bloco de comandos que incluem uma ou mais chamadas recursivas.

Exemplo de problema recursivo:

Fatorial definido como:
a) 0! = 1
b) n! = n (n-1)!

Função recursiva para cálculo de fatorial:

```
long long int fatorial(int n)
{
  if(n==0) return 1;
  return n*fatorial(n-1);
}
```

Outros Exemplos:

Função que calcula a somatória dos inteiros de 1 a n:

```
int soma(int n)
{
        if (n==1) return 1;
      return n + soma(n-1);
}
```

Função que calcula o número de dígitos de um número inteiro positivo:

```
int digitos(int n)
{
        if (n<10) return 1;
        return 1 + digitos(n/10);
}
```

Função que escreve a representação binária de um número inteiro:

```
void binario(int n)
```

```
{
        if (n<2) printf("%1d",n);
        else {
                binario(n/2);
                printf("%1d",n%2);
        }
}
```

14 - Arquivos

Um arquivo é a unidade de armazenamento de informações em memória secundária como disco.

As funções para entrada/saída de arquivos encontram-se na biblioteca <stdio.h>.

Declaração de ponteiro para arquivo:

FILE *f;

Declara um apontador f para uma estrutura com informações para um arquivo (localização do buffer para E/S, posição para no buffer, se arquivo foi aberto para leitura ou escrita, etc).

Abertura de arquivo:

fopen(char *nome, char * modo)

Argumentos: nome do arquivo e modo (“r” para leitura, “w” para escrita, “a” para escrita no final do arquivo, “r+” para leitura e escrita, “w+” para abrir novo arquivo para leitura e escrita).
Retorna: apontador para estrutura com informações do arquivo ou NULL se houver erro.

Entrada/Saída de caracteres:

fgetc(FILE *f)

Argumentos: apontador para arquivo.
Retorna: próximo caráter ou EOF se fim de arquivo.

fputc(int c, FILE *f)

Argumentos: caráter a ser escrito, apontador para arquivo.
Retorna: caráter escrito ou EOF em caso de erro.

Entrada/Saída de cadeias:

fgets(char *linha, int max, FILE *f)

Argumentos: cadeia linha onde será colocada a próxima linha lida do arquivo f. Inclui ‘\n’ e ‘\0’ no final da cadeia. Lê no máximo max-1 caracteres.

fputs(char *linha, FILE *f)

Argumentos: escreve cadeia linha no arquivo f.
Retorna: EOF em caso de erro.

Entrada/Saída formatada:

fscanf(FILE *f, formatos, variáveis)

Lê variáveis nos formatos especificados do arquivo f.
Retorna EOF se fim de arquivo ou erro, ou número de itens lido no caso de sucesso.

fprintf(FILE *f., formatos, variáveis)

Escreve variáveis nos formatos especificados no arquivo f.
Retorna negativo se erro.

Término de uso:

fclose(f)

Desfaz a conexão entre o apontador f e o arquivo externo. Libera apontador que pode ser utilizado para outro arquivo.

Exemplo:

```
// programa que lê arquivo entrada.txt e copia no arquivo saida.txt

main() {

        FILE *e,*f;
        char c;

        e=fopen("entrada.txt","r");
        s=fopen("saida.txt","w");
        while((c=fgetc(e))!=EOF) fputc(c,s);
        fclose(e);
        fclose(s);
}
```

Exercícios

1. Ler um arquivo e contar quantos caracteres tem o arquivo, quantos são letras e quantas linhas tem o arquivo.
2. Verificar se dois arquivos têm o mesmo conteúdo.
3. Dada uma cadeia de entrada, contar quantas vezes essa cadeia aparece em um arquivo.

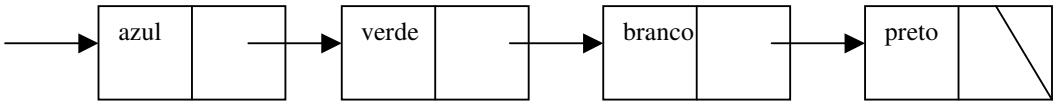
15- Listas Ligadas

Uma lista é uma seqüência de 0 ou mais elementos de um determinado tipo.

Representação:

- $a_0, a_1, a_2, \ldots, a_n$
- $n \geq 0$ e a_i têm mesmo tipo
- n é o tamanho da lista
- elemento a_i está na posição i
- elementos são ordenados (a_i precede a_{i+1})
- cada elemento tem apontador para elemento seguinte

Exemplo de uma coleção de cores armazenadas em uma lista ligada:



Operações:

- insere(x,p.L): insere x na posição p da lista L movendo elementos a partir de p para uma posição adiante na lista.
- busca(x,L): retorna posição de x na lista L.
- recupera(p,L): retorna elemento na posição p da lista L.
- remove(p,L): remove elemento na posição p da lista L.
- imprime(L): imprime elementos de L na ordem de ocorrência.

15.1 Listas Simples - Representação com apontadores

Declaração

```
struct no {
        char info[30];
        struct no *prox;
};
```

Função para inserir um elemento no início da lista

```
/* insere no início da lista */
struct no *insere(char *x, struct no *ini)
{
        struct no * aux;
        aux = (struct no *) malloc(sizeof(struct no));
        strcpy(aux->info,x);
        aux->prox=ini;
        return(aux);
}
```

Função para buscar um elemento na lista

```
/* busca x, retorna apontador da primeira ocorrência ou NULL caso não encontre*/
struct no * busca (char *x, struct no *ini)
{
        while (ini)
                if(strcmp(ini->info,x)) ini=ini->prox;
                else return(ini);
        return(NULL);
}
```

15.2 Listas Simples com nó cabeça

Função para iniciar uma lista

```
struct no *inicia()
{
        struct no *inicio= (struct no *)malloc(sizeof(struct no));
        inicio->prox=NULL;
        return(inicio);
}
```

Função para inserir um elemento no inicio da lista

```
void insere(struct no *inicio, char *x)
{
        struct no *temp=(struct no*)malloc(sizeof(struct no));
        strcpy(temp->info,x);
        temp->prox=inicio->prox;
        inicio->prox=temp;
}
```

15.3 Exercícios

1. Função para imprimir campo info dos elementos da lista.
2. Função para contar número de nós na lista.
3. Função para contar ocorrências do elemento x na lista.
4. Função para verificar se 2 listas são iguais (retorna 1 ou 0).
5. Função para unir 2 listas (retorna apontador para lista unida).
6. Função que insere elemento em lista ordenada deixando a lista ordenada.
7. Função para remover um elemento de uma lista ordenada.

15.4 Listas Simples – elementos da lista estão ordenados

Função para inserir em lista ordenada com nó cabeça

```
void insereordenado(struct no *inicio, char *x)
```

```
{
        struct no *novo,*atual,*pred;
        atual=inicio->prox;
        pred=inicio;
        while (atual)
                if (strcmp(x,atual->info)>0){
                        pred=atual;
                        atual=atual->prox;
                }
                else break;
        novo= (struct no*)malloc(sizeof(struct no));
        strcpy(novo->info,x);
        novo->prox=pred->prox;
        pred->prox=novo;
}
```

Função para remover de lista ordenada com nó cabeça

```
void removeordenado(struct no *inicio, char *x)
{
        struct no *atual,*pred;
        int i;
        atual=inicio->prox;
        pred=inicio;
        while (atual)
                if (i=strcmp(x,atual->info)>0){
                        pred=atual;
                        atual=atual->prox;
                }
                else if (i) return();
                     else {
                        pred->prox=atual->prox;
                        free(atual);
                        return();
                        }
}
```