

Algoritmos e Estruturas de Dados I

Estruturas Condicionais e de Repetição (parte 1)

Pedro O.S. Vaz de Melo

Problema 1

Determine as raízes da equação $ax^2 + bx + c = 0$.

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

int main(int argc, char *argv[])
{
    int a = 2, b = 3, c = 1;
    float delta, x1, x2;

    delta = b*b - 4*a*c;
    printf("A equacao %s\n", (delta >= 0) ? "possui raizes reais" :
                                                "nao possui raizes reais");

    if (delta >= 0)
    {
        printf("As raizes sao %s\n", (delta > 0) ? "diferentes" : "iguais");
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        printf("Raiz x1 = %f\n", x1);
        printf("Raiz x2 = %f\n", x2);
    }

    system("PAUSE");
    return 0;
}
```

Processamento condicional

- Todo programa na linguagem C inicia sua execução na primeira instrução da função **main**.
- As instruções são executadas **sequencialmente**, na ordem em que aparecem no texto.
- Muitas vezes, é necessário executar um conjunto de instruções **se uma condição for verdadeira** e, **caso contrário**, um outro conjunto de instruções.
- Quando um programa executa ou deixa de executar instruções com base no valor de uma condição, o programa realiza um **processamento condicional**.

Processamento condicional

- O programa `p05.c` realiza um processamento condicional.

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

```
int main(int argc, char *argv[])
{
    int a = 2, b = 3, c = 1;
    float delta, x1, x2;
```

```
    delta = b*b - 4*a*c;
```

```
    printf("A equacao %s\n", (delta >= 0) ? "possui raizes reais" :
        "nao possui raizes reais");
```

```
    if (delta >= 0)
    {
```

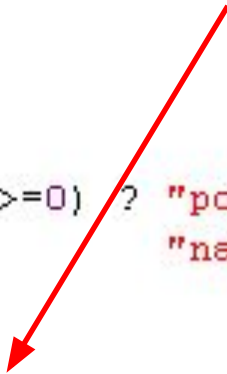
```
        printf("As raizes sao %s\n", (delta > 0) ? "diferentes" : "iguais");
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        printf("Raiz x1 = %f\n", x1);
        printf("Raiz x2 = %f\n", x2);
```

```
    }
```

```
    system("PAUSE");
    return 0;
```

```
}
```

Estas instruções serão executadas somente se `delta >= 0`.



Processamento condicional

- Para executar um processamento condicional, um programa precisa utilizar o comando **if**.
 - Todo comando **if** requer uma **condição**. O valor de uma condição pode ser **verdadeiro** ou **falso**.
 - Em C, não existe um tipo de dados específico para representar valores lógicos (**V** ou **F**).
- Qualquer valor diferente de zero é interpretado como verdadeiro, enquanto zero é falso.

Operadores relacionais

- Para escrever condições, são utilizados os **operadores relacionais** e os **operadores lógicos**.

Operador	Significado
----------	-------------

Operadores relacionais

- Para escrever condições, são utilizados os **operadores relacionais** e os **operadores lógicos**.

Operador	Significado	Condição	Valor lógico
>	Maior do que.		
<	Menor do que.		
>=	Maior do que ou igual a.		
<=	Menor do que ou igual a.		
==	Igual a.		
!=	Diferente de.		

```
int a = 3; float x = 1.5;
```

Operadores relacionais

- Para escrever condições, são utilizados os **operadores relacionais** e os **operadores lógicos**.

Operador	Significado	Condição	Valor lógico
>	Maior do que.	$(a \neq x)$	Verdadeiro.
<	Menor do que.	$(a/2.0 == x)$	Verdadeiro.
>=	Maior do que ou igual a.	$(a/2 == x)$	Falso.
<=	Menor do que ou igual a.	$(a/x < 2)$	Falso.
==	Igual a.	(a)	Verdadeiro.
!=	Diferente de.	$(a - 2*x)$	Falso.

```
int a = 3; float x = 1.5;
```


Operadores lógicos

- Os operadores lógicos permitem combinar várias condições em uma única expressão lógica.

Operador	Significado
----------	-------------

Operadores lógicos

- Os operadores lógicos permitem combinar várias condições em uma única expressão lógica.

Operador	Significado	Expressão	Valor Lógico
&&	Conjunção lógica (“and”)		
	Disjunção lógica (“or”)		
!	Negação lógica (“not”)		

```
int a = 3; float x = 1.5;
```

Operadores lógicos

- Os operadores lógicos permitem combinar várias condições em uma única expressão lógica.

Operador	Significado	Expressão	Valor Lógico
&&	Conjunção lógica (“and”)	$((a/2 == x) \ \&\& \ (a > 2))$	Falso.
	Disjunção lógica (“or”)	$((x \leq a) \ \&\& \ (a \geq 2*x))$	Verdadeiro.
!	Negação lógica (“not”)	$!(a/3 \leq x)$	Falso.
		$(a \ \&\& \ x)$	Verdadeiro.
		$((a - 2*x) \ \ (x < a/2))$	Falso.

```
int a = 3; float x = 1.5;
```

Operador condicional

- O operador condicional na linguagem C tem a seguinte sintaxe:

```
(condição)?resultado-se-condição-verdadeira : resultado-se-condição-falsa
```

- Os resultados podem ser de qualquer tipo (int, float, char, double) e mesmo strings.
- Exemplos:

```
(b != 0) ? a/b : 0  
(peso <= 75) ? "ok" : "deve emagrecer"
```

Operador condicional

- O operador condicional pode ser usado em atribuições.
- Exemplo:

```
float nota1 = 5.0, nota2 = 4.0;  
  
media = ((nota1 >= 3) && (nota2 >= 5)) ?  
    (nota1 + 2*nota2)/3 :  
    (nota1 + nota2)/2;
```

↓
media recebe o valor 4.5

Qual seria o valor de média se:

```
float nota1 = 5.0;  
float nota2 = 6.5;
```

Operador condicional

- No programa `p01.c`, o operador condicional é usado dentro da função `printf`.

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

int main(int argc, char *argv[])
{
    int a = 2, b = 3, c = 1;
    float delta, x1, x2;

    delta = b*b - 4*a*c;
    printf("A equacao %s\n", (delta >= 0) ? "possui raizes reais" :
                                                "nao possui raizes reais");
    if (delta >= 0)
    {
        printf("As raizes sao %s\n", (delta > 0) ? "diferentes" : "iguais");
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        printf("Raiz x1 = %f\n", x1);
        printf("Raiz x2 = %f\n", x2);
    }

    system("PAUSE");
    return 0;
}
```

Atribuição e teste de igualdade

- **Atenção!**

- Um erro comum em linguagem C é usar o operador de atribuição (=) em vez do operador relacional (==) em condições que testam igualdade.

```
int fator = 3;
if (fator == 1)
{
    printf("O fator e' unitario\n");
}
printf("fator = %d\n", fator)
```

Imprime:

fator = 3

pois:

(fator == 1) é falso!

```
int fator = 3;
if (fator = 1)
{
    printf("O fator e' unitario\n");
}
printf("fator = %d\n", fator)
```

Imprime:

O fator e' unitario

fator = 1

pois:

(fator = 1) é verdadeiro!

Problema 2

- Dada uma temperatura em graus centígrados, apresentá-la em graus Fahrenheit. A fórmula de conversão é: $F = (9 * C + 160) / 5$.

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

int main(int argc, char *argv[])
{
    float C,F;
    printf("Digite a temperatura em graus C: ");
    scanf("%f", &C);
    F = (9 * C + 160) / 5;
    printf("Esta temperatura corresponde a %.1f graus F\n", F);
    system("PAUSE");
    return 0;
}
```


Problema 2

- Nos programas anteriores, os valores das variáveis eram estabelecidos em operações de atribuição. Mas agora, qual é o valor de C?

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

int main(int argc, char *argv[])
{
    float C,F;
    printf("Digite a temperatura em graus C: ");
    scanf("%f", &C);
    F = (9 * C + 160) / 5;
    printf("Esta temperatura corresponde a %.1f graus F\n", F);
    system("PAUSE");
    return 0;
}
```

Leitura de Dados

- Uma outra forma de atribuir valores a variáveis é a **leitura de dados**. Em C, usa-se a função **scanf**.
- Assim como **printf**, a função **scanf** pode ter vários parâmetros, sendo o primeiro uma **string**.
- No caso da função **scanf**, esta string deve conter apenas **tags** (ex: “%d %f”), separadas por espaços em branco.
- Os demais parâmetros da função **scanf** devem ser endereços de variáveis.

Leitura de Dados

- O que acontece quando o computador executa uma **instrução de leitura de dados**? Exemplo:

```
scanf ("%f", &C) ;
```

- A execução do programa é **interrompida**. O computador espera que o usuário digite algum valor e **pressione a tecla Enter**.
- Após pressionar **Enter**, o computador retoma a execução do programa e armazena o(s) valor(es) digitado(s) no(s) endereço(s) fornecido(s) na função **scanf**.

Leitura de Dados

- Após pressionar **Enter**, o computador retoma a execução do programa e armazena o(s) valor(es) digitado(s) no(s) endereço(s) fornecido(s) na função **scanf**.

```
scanf ("%f", &C) ;
```

Endereço de Memória	Variável Alocada	Valor
10FA0001	C	
10FA0002	aux	
10FA0003		
10FA0004		

Leitura de Dados

- Após pressionar **Enter**, o computador retoma a execução do programa e armazena o(s) valor(es) digitado(s) no(s) endereço(s) fornecido(s) na função **scanf**.

```
scanf ("%f", &C) ;
```



exemplo: 20 +
ENTER

Endereço de Memória	Variável Alocada	Valor
10FA0001	C	
10FA0002	aux	
10FA0003		
10FA0004		

Leitura de Dados

- Após pressionar **Enter**, o computador retoma a execução do programa e armazena o(s) valor(es) digitado(s) no(s) endereço(s) fornecido(s) na função **scanf**.

```
scanf ("%f", &C) ;
```



exemplo: 20 +
ENTER

Endereço de Memória	Variável Alocada	Valor
10FA0001	C	20.0
10FA0002	aux	
10FA0003		
10FA0004		

Leitura de Dados

- O que difere a **leitura de dados** da **operação de atribuição**?

- Na **operação de atribuição**, o valor a ser atribuído é definido antes da execução do programa, enquanto numa **operação de leitura de dados**, o valor atribuído é definido durante a execução.

- Em programação, diz-se que coisas são **estáticas** quando ocorrem antes do programa executar e **dinâmicas** quando ocorrem durante a execução.

```
C = 32;
```

```
scanf ("%f", &C)
```

→ Valor de C é estabelecido **estaticamente**.

→ Valor de C é estabelecido **dinamicamente**.

Leitura de Dados

- Na leitura de dados, o valor digitado pelo usuário deve ser do mesmo tipo que a variável.
- Com a leitura de dados, a execução de um programa pode ser realizada para valores diferentes das variáveis.
- Porém, se o valor da variável é estabelecido de forma estática, para cada valor da variável, é necessário compilar o programa novamente.

Problema 3

- Dadas as idades (tipo `int`) e os pesos (tipo `float`) de duas pessoas, exibir quem é a pessoa mais velha e a sua idade e quem é a pessoa mais leve e o seu peso.

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

int main(int argc, char *argv[])
{
    int idade1, idade2, maior_idade;
    int mais_velho, mais_leve;
    float peso1, peso2, menor_peso;
    printf("Digite a idade e o peso da pessoa 1: ");
    scanf("%d %f", &idade1, &peso1);
    printf("Digite a idade e o peso da pessoa 2: ");
    scanf("%d %f", &idade2, &peso2);
```

Problema 3

```
if (idade1 > idade2)
{
    maior_idade = idade1;
    mais_velho = 1;
}
else
{
    maior_idade = idade2;
    mais_velho = 2;
}
if (peso1 < peso2)
{
    menor_peso = peso1;
    mais_leve = 1;
}
else
{
    menor_peso = peso2;
    mais_leve = 2;
}
printf("Maior idade = %d (da pessoa %d)\n", maior_idade, mais_velho);
printf("Menor peso = %.1f (da pessoa %d)\n", menor_peso, mais_leve);
system("PAUSE");
return 0;
}
```

Comando `if-else`

- Todo comando `if` requer uma condição que pode ser verdadeira ou falsa.
- Caso a `condição seja verdadeira`, o comando `if` executa um conjunto de instruções, podendo deixar de executar um outro conjunto alternativo.
- Quando existe um conjunto de instruções a ser executado, caso o valor da condição seja falso, utiliza-se o comando `if-else`.

Comando if-else

- Exemplo:

```
if (delta >=0)
{
    x1 = (-b + sqrt(delta)) / (2*a);
    x2 = (-b - sqrt(delta)) / (2*a);
}
else
{
    printf("Sem raízes reais.");
}
```

- Um conjunto de instruções começa com o símbolo { e termina com o símbolo }. Caso, o conjunto contenha apenas uma instrução, as chaves são opcionais.

Comando **if-else**

- Caso, o conjunto contenha apenas uma instrução, as chaves são opcionais.

```
if (delta >=0)
{
    x1 = (-b + sqrt(delta)) / (2*a);
    x2 = (-b - sqrt(delta)) / (2*a);
}
else
    printf("Sem raízes reais.");
```

Comando if-else

- Qualquer instrução pode fazer parte de um conjunto de instruções, inclusive um comando **if** ou um comando **if-else**.

```
if (delta >=0)
{
    x1 = (-b + sqrt(delta)) / (2*a);
    if (delta == 0)
        x2 = x1;
    else
        x2 = (-b - sqrt(delta)) / (2*a);
}
else
{
    printf("Sem raízes reais.");
}
```

Comando **if-else**

- Qualquer instrução pode fazer parte de um conjunto de instruções, inclusive um comando **if** ou um comando **if-else**.

```
if (delta >= 0)
{
    x1 = (-b + sqrt(delta)) / (2*a);
    if (delta == 0)
        x2 = x1;
    else
        x2 = (-b - sqrt(delta)) / (2*a);
}
else
{
    printf("Sem raízes reais.");
}
```

Por que não foram usadas as chaves { } nesse comando?

A importância dos recuos (indentação do código)

- Programas mais complexos são mais difíceis de ler e compreender.
- Uma forma de melhorar a legibilidade do programa é usar **recuos**.
- Os recuos devem ser usados sempre após o símbolo **{**, sendo as instruções recuadas à direita.
- O símbolo **}** deve estar alinhado ao abre-chaves correspondente.

A importância dos recuos (indentação do código)

- Exemplo:

```
if (nota >= 6)
    if (nota_anterior < nota)
        printf("Você está melhorando.");
    else
        printf("Você precisa estudar mais!");
else
    printf("Sem estudo é difícil ser aprovado.");
```

Recuos não resolvem ambiguidades

Exemplo:

```
if (nota >= 7)
    if (nota_anterior < nota)
        printf("Você está melhorando.");
else
    printf("Sem estudo é difícil ser aprovado.");
```

De quem é o **else** acima?

O compilador sempre associa um **else** ao “**if** anterior mais próximo que ainda não possui um **else**.”

Como associar o **else** à instrução **if (nota >= 7)**?

Recuos não resolvem ambiguidades

- Exemplo:

```
if (nota >= 7)
{
    if (nota_anterior < nota)
        printf("Você está melhorando.");
}
else
    printf("Sem estudo é difícil ser aprovado.");
```

- Neste caso, as chaves, em vez de opcionais, serão obrigatórias, pois apenas os recuos não resolvem.

Problema 4

- Dado o valor da variável N , determine a soma dos números inteiros de 1 a N .

Problema 4

- Deseja-se calcular o valor de: $1 + 2 + 3 + \dots + N$.
- **Observação:** não sabemos, *a priori*, quantos termos serão somados, pois o valor de **N** é estabelecido dinamicamente.
- Para se calcular esta soma, utiliza-se o comando **while**.

- O comando **while** permite que um conjunto de instruções seja executado tantas vezes quantas forem necessárias, **enquanto** uma condição for verdadeira.

Problema 4

- Dado o valor da variável **N**, determine a soma dos números inteiros de **1** a **N**.

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

int main(int argc, char *argv[])
{
    int N;
    int i, s;

    printf("Digite o valor de N: ");
    scanf("%d", &N);
    s = 0;
    i = 1;
    while (i <= N)
    {
        s = s + i;
        i++;
    }
    printf("Soma = %d\n", s);
    system("PAUSE");
    return 0;
}
```

O comando **while**

- Quando um programa executa um conjunto de instruções repetidas vezes, diz-se que o programa está realizando um **processamento iterativo**.
- Cada execução do conjunto de instruções denomina-se uma **iteração**. Exemplo de uso do comando **while**:

```
s = 0;  
i = 1;  
while (i <= 3)  
{  
    s = s + i;  
    i++;  
}
```

Instante	s	i	(i <= 3)
inicial	0	1	V

O comando **while**

- Quando um programa executa um conjunto de instruções repetidas vezes, diz-se que o programa está realizando um **processamento iterativo**.
- Cada execução do conjunto de instruções denomina-se uma **iteração**. Exemplo de uso do comando **while**:

```
s = 0;  
i = 1;  
while (i <= 3)  
{  
    s = s + i;  
    i++;  
}
```

Instante	s	i	(i <= 3)
inicial	0	1	V
1ª Iteração	0 + 1 = 1	1 + 1 = 2	V

O comando **while**

- Quando um programa executa um conjunto de instruções repetidas vezes, diz-se que o programa está realizando um **processamento iterativo**.
- Cada execução do conjunto de instruções denomina-se uma **iteração**. Exemplo de uso do comando **while**:

```
s = 0;  
i = 1;  
while (i <= 3)  
{  
    s = s + i;  
    i++;  
}
```

Instante	s	i	(i <= 3)
inicial	0	1	V
1ª Iteração	0 + 1 = 1	1 + 1 = 2	V
2ª Iteração	1 + 2 = 3	2 + 1 = 3	V

O comando **while**

- Quando um programa executa um conjunto de instruções repetidas vezes, diz-se que o programa está realizando um **processamento iterativo**.
- Cada execução do conjunto de instruções denomina-se uma **iteração**. Exemplo de uso do comando **while**:

```
s = 0;  
i = 1;  
while (i <= 3)  
{  
    s = s + i;  
    i++;  
}
```

Instante	s	i	(i <= 3)
inicial	0	1	V
1ª Iteração	0 + 1 = 1	1 + 1 = 2	V
2ª Iteração	1 + 2 = 3	2 + 1 = 3	V
3ª Iteração	3 + 3 = 6	3 + 1 = 4	F

O comando **while**

```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d",&N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

Endereço	Variável	Conteúdo
10FA0001		
10FA0002		
10FA0003		
10FA0004		

O comando **while**



```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d", &N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

Endereço	Variável	Conteúdo
10FA0001	N	
10FA0002	s	0
10FA0003	i	1
10FA0004		

O comando **while**



```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d",&N) ;  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	0
10FA0003	i	1
10FA0004		

O comando **while**



```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d", &N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

(i <= N):
TRUE

Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	0
10FA0003	i	1
10FA0004		

O comando **while**



```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d",&N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	1
10FA0003	i	2
10FA0004		

O comando **while**



```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d", &N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

(i <= N):
TRUE

Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	1
10FA0003	i	2
10FA0004		

O comando **while**



```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d",&N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	3
10FA0003	i	3
10FA0004		

O comando **while**



```
int N;
int s = 0, i = 1;
printf("Digite o valor de N: ")
scanf("%d", &N);
while (i <= N)
{
    s = s + i;
    i++;
}
printf("\nSoma = %d", s);
```

(i <= N):
TRUE

Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	3
10FA0003	i	3
10FA0004		

O comando **while**



```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d",&N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	6
10FA0003	i	4
10FA0004		

O comando **while**



```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d", &N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```

(i <= N):
FALSE

Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	6
10FA0003	i	4
10FA0004		

O comando **while**

```
int N;  
int s = 0, i = 1;  
printf("Digite o valor de N: ")  
scanf("%d",&N);  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}  
printf("\nSoma = %d", s);
```



Endereço	Variável	Conteúdo
10FA0001	N	3
10FA0002	s	6
10FA0003	i	4
10FA0004		

O comando **while**

- A solução do problema 4 sempre irá terminar, pois **i** será maior do que **N** em algum momento.
- Porém, pode a execução de um programa com processamento iterativo não terminar? Observe:

```
s = 0;  
i = 0;  
while (i < 3)  
{  
    i--;  
    s = s + i;  
}
```

Este **laço** ou **loop**
nunca irá terminar!
(Erro de lógica)

```
s = 0;  
i = 0;  
while (i > -3)  
{  
    i--;  
    s = s + i;  
}
```

Maneira
correta

O comando **while**

- **Atenção!**
- Em alguns casos, o loop infinito pode ser desejável.
 - Exemplo: um programa que monitora um reator nuclear deve estar sempre em execução.
- Neste caso, pode-se escrever:

```
while (1)
{
    monitora_reator();
}
```

Comando **do-while**

Outra forma de repetir um conjunto de instruções é com o comando **do-while**.

```
s = 0;  
i = 1;  
do  
{  
    s = s + i;  
    i++;  
}  
while (i <= N);
```

Comando
do-while

```
s = 0;  
i = 1;  
while (i <= N)  
{  
    s = s + i;  
    i++;  
}
```

Comando
while

Veja que no comando **while**, a condição é testada antes da execução das instruções, ao contrário do comando **do-while**. O que acontece para **N=0**?