



# Algoritmos e Estrutura de Dados – I

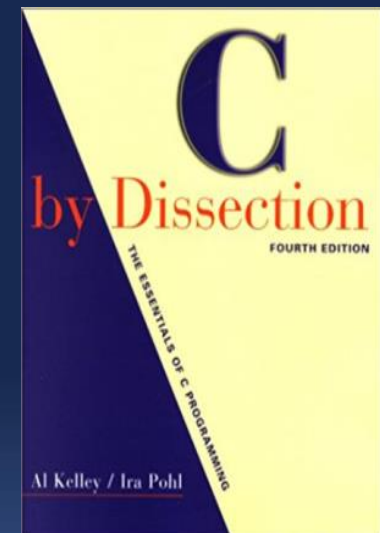
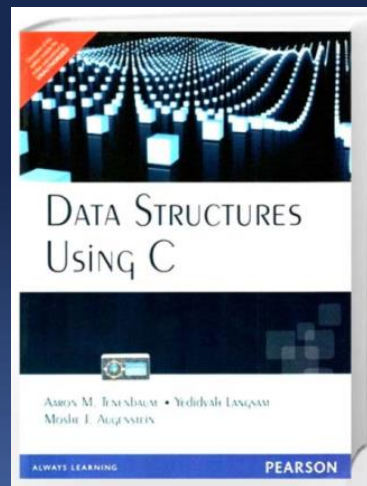
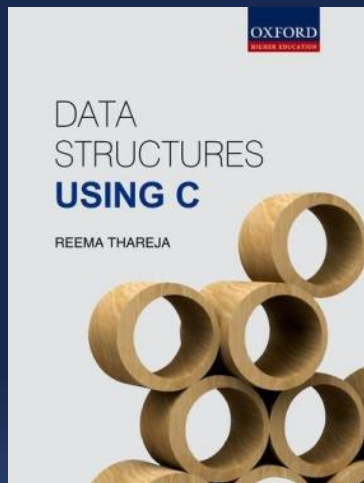
## Unidade 6 – Fluxo de Controle



Prof. Aparecido V. de Freitas  
Doutor em Engenharia  
da Computação pela EPUSP  
[aparecidovfreitas@gmail.com](mailto:aparecidovfreitas@gmail.com)

# Bibliografia

- ✓ Data Structures using C - Oxford University Press - 2014
- ✓ Data Structures Using C - A. Tenenbaum, M. Augensem, Y. Langsam, Pearson 1995
- ✓ C By Dissection - Kelley, Pohh - Third Edition - Addison Wesley



# Introdução



- ✓ Os comandos na Linguagem são normalmente executados em **sequência**;
- ✓ Esse procedimento é conhecido por **Fluxo Sequencial de Controle**;
- ✓ Porém, é muito comum nos programas haver mudanças desse fluxo sequencial;
- ✓ Essas mudanças de fluxo são causadas por comandos de decisão ou de repetição que usualmente ocorrem nos programas em tempo de execução.



# Operadores Lógicos e Relacionais

- ✓ São os operadores frequentemente empregados para se alterar o fluxo de controle dos programas;
- ✓ Esses operadores são usados em expressões que quando avaliadas retornam valores **true** ou **false**.

Relational, equality, and logical operators		
<b>Relational operators</b>	less than	<
	greater than	>
	less than or equal to	<=
	greater than or equal to	>=
<b>Equality operators</b>	equal to	==
	not equal to	!=
<b>Logical operators</b>	(unary) negation <i>not</i>	!
	logical <u>and</u>	&&
	logical <u>or</u>	

# Operadores Lógicos e Relacionais



- ✓ O operador lógico **!** é **unário**;
- ✓ Todos os demais são **binários**. Eles operam em expressões cujo resultado corresponde ao valor inteiro 0 ou ao valor inteiro 1;
- ✓ Em C, **falso** é representado pelo valor **zero** e **true** é representado por **qualquer valor diferente de zero**;
- ✓ Exemplos de valores em C que representam falso: inteiro **0**, ou valor em ponto flutuante **0.0** ou ainda o caractere nulo (**'\0'**);

Relational, equality, and logical operators		
Relational operators	less than	<
	greater than	>
	less than or equal to	<=
	greater than or equal to	>=
Equality operators	equal to	==
	not equal to	!=
Logical operators	(unary) negation <i>not</i>	!
	logical and	&&
	logical or	





# Operadores Relacionais

Values of relational expressions				
<b>a - b</b>	<b>a &lt; b</b>	<b>a &gt; b</b>	<b>a &lt;= b</b>	<b>a &gt;= b</b>
positive	0	1	0	1
zero	0	0	1	1
negative	1	0	1	0

# Programa 1

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

int main ( ) {
    setlocale(LC_ALL, "Portuguese");

    int i=1, j=2, k=3;

    printf ("A expressão => (i < (j-k)) retornou %d" , (i < (j-k)) ) ;

    if (i < (j-k))
        printf("\n\n ---- Expressão avaliada como TRUE ----\n\n");
    else
        printf("\n\n ---- Expressão avaliada como FALSE ----\n\n");

    return 0;
}
```

# Programa 1

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_1S\Alg_Est_Dados_I\Fontes_C\Pgm_6_1.exe
A expressao => (i < (j-k)) retornou 0
---- Expressao avaliada como FALSE ----

-----
Process exited after 0.7236 seconds with return value 0
Press any key to continue . . .
```



# Programa 2

```
/* Unidade 6 - Programa 2 */
#include <stdio.h>
#include <locale.h>

int main ( ) {
    setlocale(LC_ALL, "Portuguese");

    int i=1, j=2, k=3;

    printf ("A expressão => ( (-i) + (5*j) ) >= (k+1) retornou %d" , ( (-i) + (5*j) ) >= (k+1) ) ;

    if ( ( (-i) + (5*j) ) >= (k+1) )
        printf("\n\n ---- Expressão avaliada como TRUE ----\n\n");
    else
        printf("\n\n ---- Expressão avaliada como FALSE ----\n\n");

    return 0;
}
```

# Programa 2

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_2.exe
A expressao => ( (-i) + (5*j) ) >= (k+1) retornou 1
---- Expressao avaliada como TRUE ----

-----
Process exited after 0.3017 seconds with return value 0
Press any key to continue . . .
```

# Programa 3

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

int main ( ) {

    setlocale(LC_ALL, "Portuguese");

    int i = 1 , j=2 , k=3;
    double x =5.5, y = 7.7;

    printf("A expressão => (x-y) <= ((j-k) -1) retornou %d" , (x-y) <= ((j-k) -1) );

    if ( (x-y) <= ((j-k) -1) )
        printf ("\n\n --- Expressão avaliada como TRUE ----\n\n");

    else
        printf("\n\n ---- Expressão avaliada como FALSE ----\n\n");

    return 0;
}
```

# Programa 3

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_3.exe
A expressao => (x-y) <= ((j-k) -1) retornou 1
---- Expressao avaliada como TRUE ----

-----
Process exited after 0.9609 seconds with return value 0
Press any key to continue . . .
```

# Programa 4

```
/* Unidade 6 - Programa 4 */
#include <stdio.h>
#include <locale.h>

int main ( ) {
    setlocale(LC_ALL, "Portuguese");

    int i=1, j=2, k =3;
    double x=5.5, y = 7.7;

    printf ("A expressão => ( ( x + k) + 7) < (y / k)  retornou %d" , ( ( x + k) + 7) < (y / k)  ) ;

    if ( ( ( x + k) + 7) < (y / k) )
        printf("\n\n ----  Expressão avaliada como TRUE  ----\n\n");
    else
        printf("\n\n ----  Expressão avaliada como FALSE  ----\n\n");

    return 0;
}
```

# Programa 4

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_4.exe
A expressao => ( ( x + k) + 7) < (y / k)  retornou 0
---- Expressao avaliada como FALSE ----
-----
Process exited after 0.4006 seconds with return value 0
Press any key to continue . . .
```

# Operadores de Igualdade

- ✓ Os operadores de igualdade `==` e `!=` são binários e atuam nas expressões;
- ✓ Ao serem avaliados também resultam no valor `int 0` ou no valor `int 1`;
- ✓ Exemplos:

```
ch == 'A'  
count != -2  
x + y == ( 2 * k) + 10
```

Values of equality expressions		
a - b	a == b	a != b
zero	1	0
nonzero	0	1



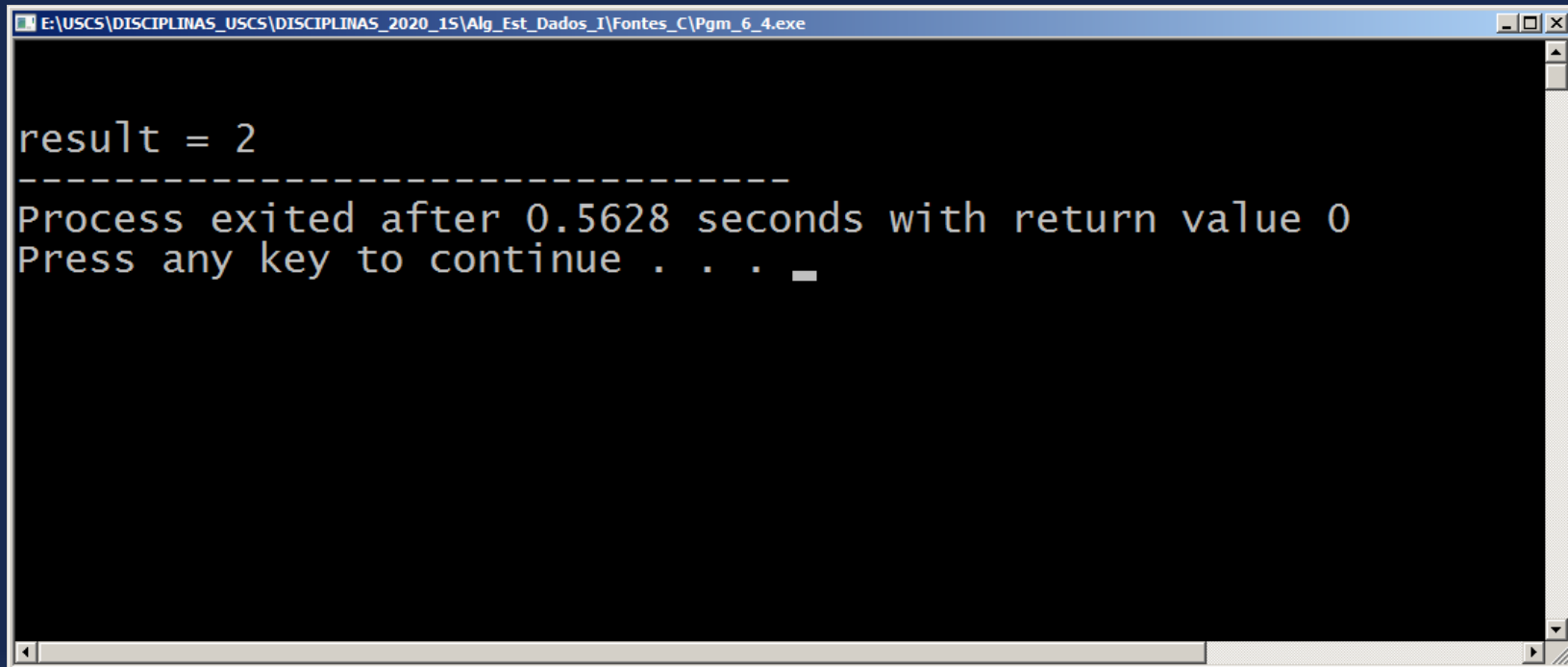
# Operadores Lógicos e Expressões

- ✓ O operador lógico **!** é unário;
- ✓ Os operadores **&&** e **||** são binários;
- ✓ Cada um destes operadores, quando aplicados em expressões resultam no valor **int 1** ou no valor **int 0**;
- ✓ Se uma expressão tem o valor **0**, sua negação resulta em **int 1**. Se a expressão tem um valor diferente de **0**, sua negação terá o valor **int 0**;

## Programa 5

```
/* Unidade 6 - Programa 5 */  
#include <stdio.h>  
#include <locale.h>  
  
int main ( ) {  
    setlocale(LC_ALL, "Portuguese");  
  
    int i=7, j=7;  
    double x=0.0, y = 999.9;  
  
    int result;  
  
    result = ! (i-j) + 1;  
  
    printf("\n\nresult = %d" , result);  
  
    return 0;  
}
```

# Programa 5



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_4.exe

result = 2
-----
Process exited after 0.5628 seconds with return value 0
Press any key to continue . . .
```

## Programa 6

```
/* Unidade 6 - Programa 6 */
#include <stdio.h>
#include <locale.h>

int main ( ) {
    setlocale(LC_ALL, "Portuguese");

    int i=7, j=7;
    double x=0.0, y = 999.9;

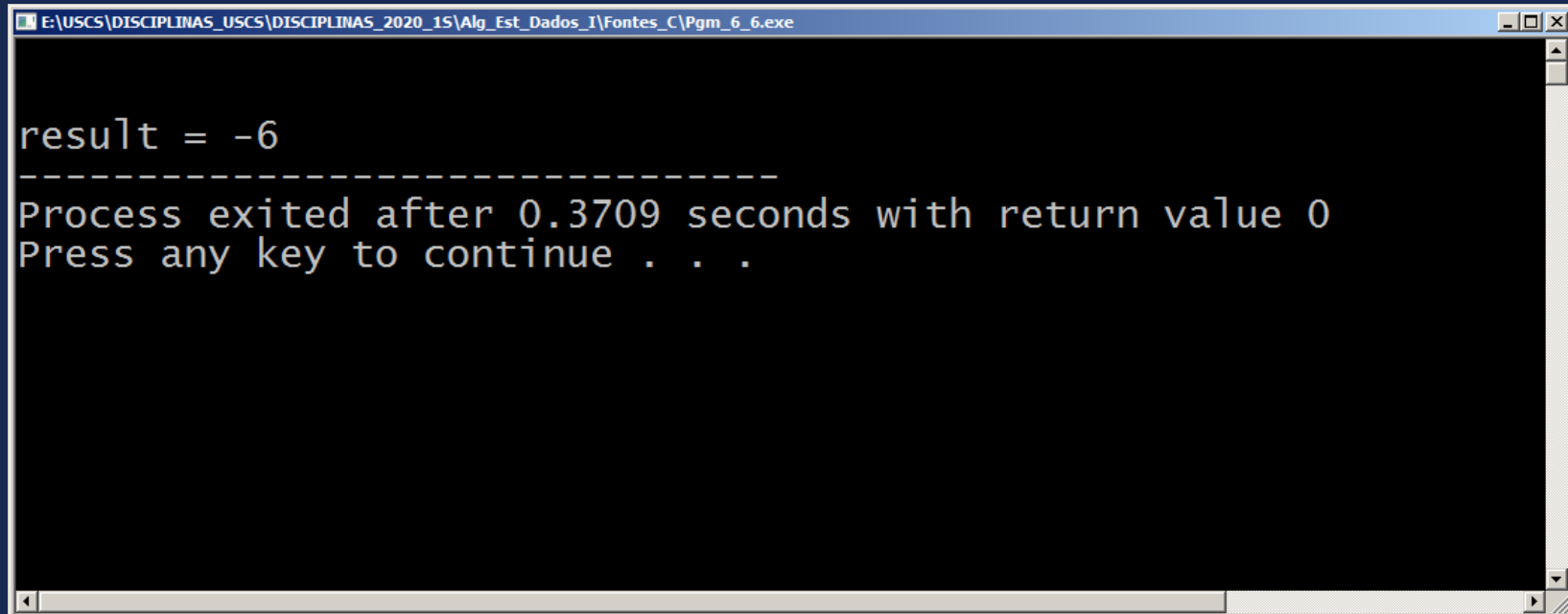
    int result;

    result = !i - j + 1;

    printf("\n\nresult = %d" , result);

    return 0;
}
```

# Programa 6



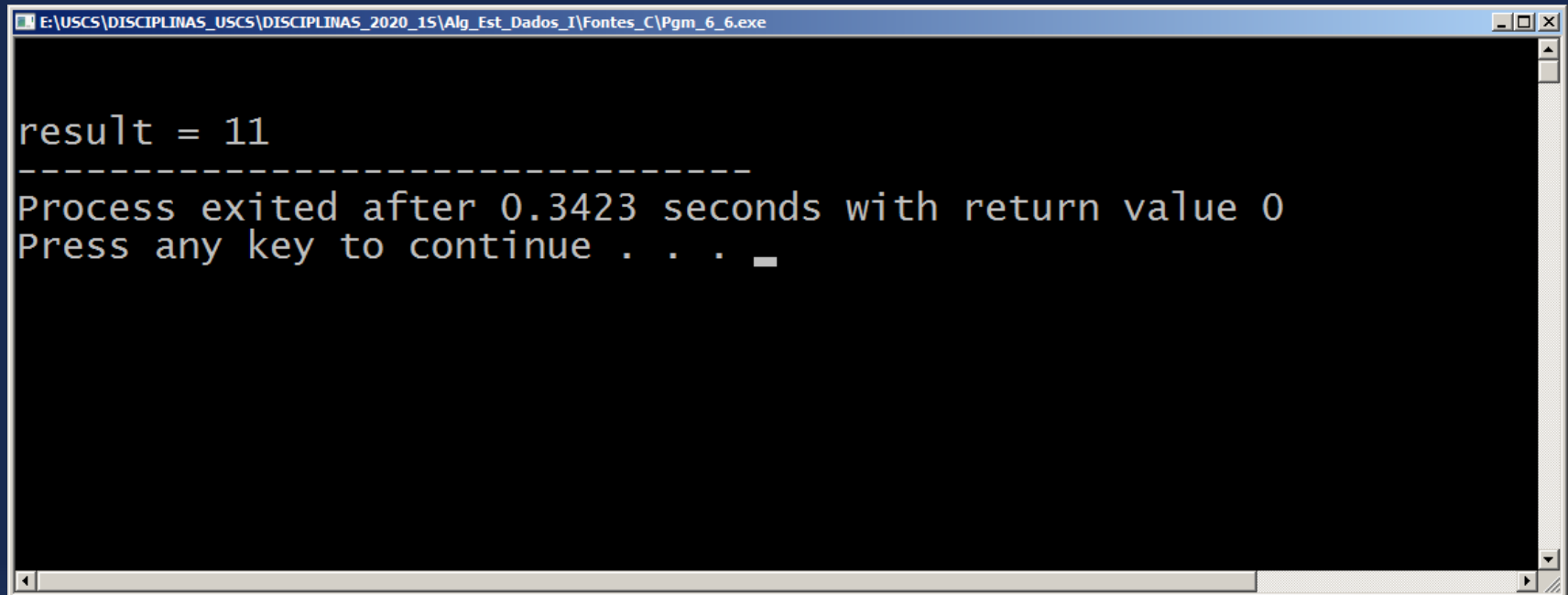
```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_6.exe

result = -6
-----
Process exited after 0.3709 seconds with return value 0
Press any key to continue . . .
```

# Programa 7

```
/* Unidade 6 - Programa 7 */  
#include <stdio.h>  
#include <locale.h>  
  
int main ( ) {  
    setlocale(LC_ALL, "Portuguese");  
  
    int i=7, j=7;  
    double x=0.0, y = 999.9;  
  
    int result;  
  
    result = !!i - !j + 10;  
  
    printf("\n\nresult = %d" , result);  
  
    return 0;  
}
```

# Programa 7



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6.exe

result = 11
-----
Process exited after 0.3423 seconds with return value 0
Press any key to continue . . . _
```



## Programa 8

```
/* Unidade 6 - Programa 8 */
#include <stdio.h>
#include <locale.h>

int main ( ) {
    setlocale(LC_ALL, "Portuguese");

    int i=7, j=7;
    double x=0.0, y = 999.9;

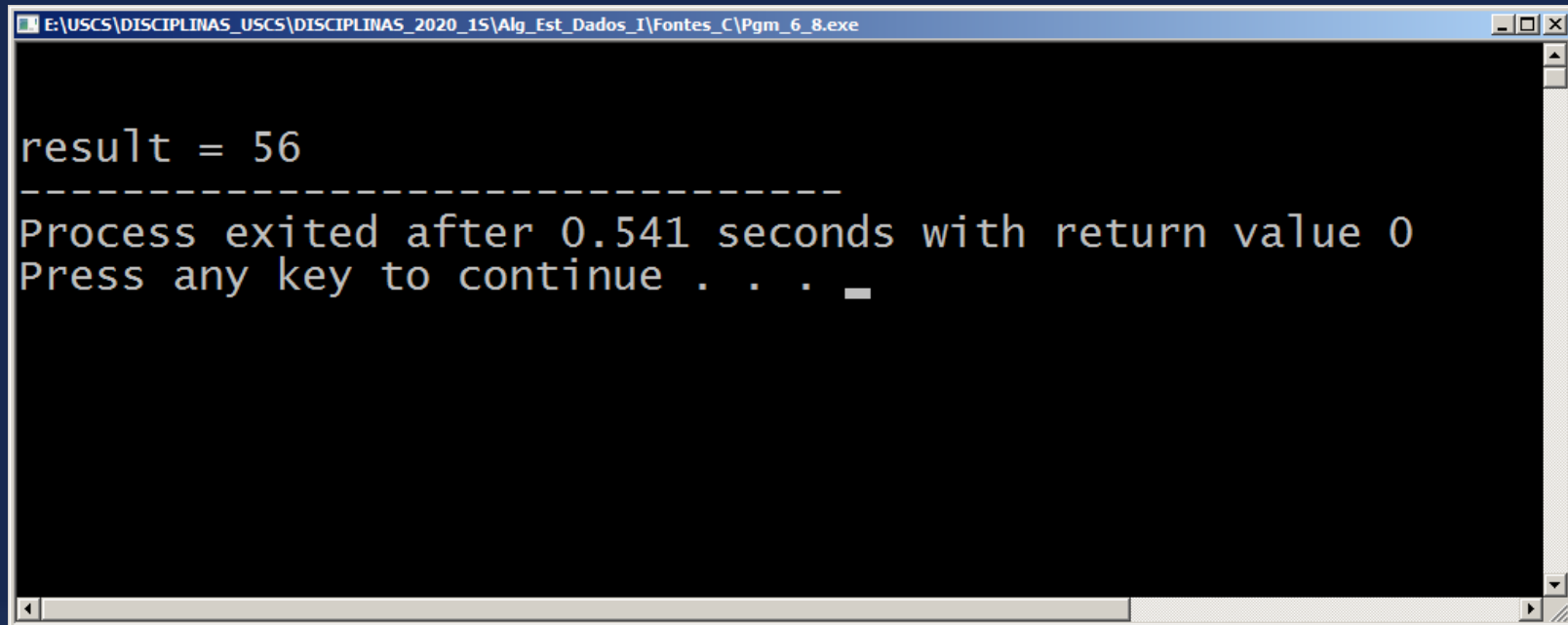
    int result;

    result = !(x + 3.3) + !!y + !!!i + 55;

    printf("\n\nresult = %d" , result);

    return 0;
}
```

# Programa 8



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_8.exe

result = 56
-----
Process exited after 0.541 seconds with return value 0
Press any key to continue . . . _
```

## Programa 9

```
/* Unidade 6 - Programa 9 */
#include <stdio.h>
#include <locale.h>

int main ( ) {
    setlocale(LC_ALL, "Portuguese");

    int i=7, j=7;
    double x=0.0, y = 999.9;

    int result;

    result = !x + 3*!!y + !(x+y) * !0;

    printf("\n\nresult = %d" , result);

    return 0;
}
```

# Programa 9

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_9.exe

result = 4
-----
Process exited after 0.2583 seconds with return value 0
Press any key to continue . . .
```

# Operadores Lógicos

- ✓ Os operador lógico **!** é unário;
- ✓ Os operadores **&&** e **||** são binários;
- ✓ Cada um destes operadores, quando aplicados em expressões resultam no valor **int 1** ou no valor **int 0**;
- ✓ Se uma expressão tem o valor **0**, sua negação resulta em **int 1**. Se a expressão tem um valor diferente de **0**, sua negação terá o valor **int 0**;

# Programa 10

```
/* Unidade 6 - Programa 10 */
#include <stdio.h>
#include <locale.h>

int main ( ) {
    setlocale(LC_ALL, "Portuguese");

    int i = 3, j = 3, k = 3;
    double x = 0.0, y = 2.3;

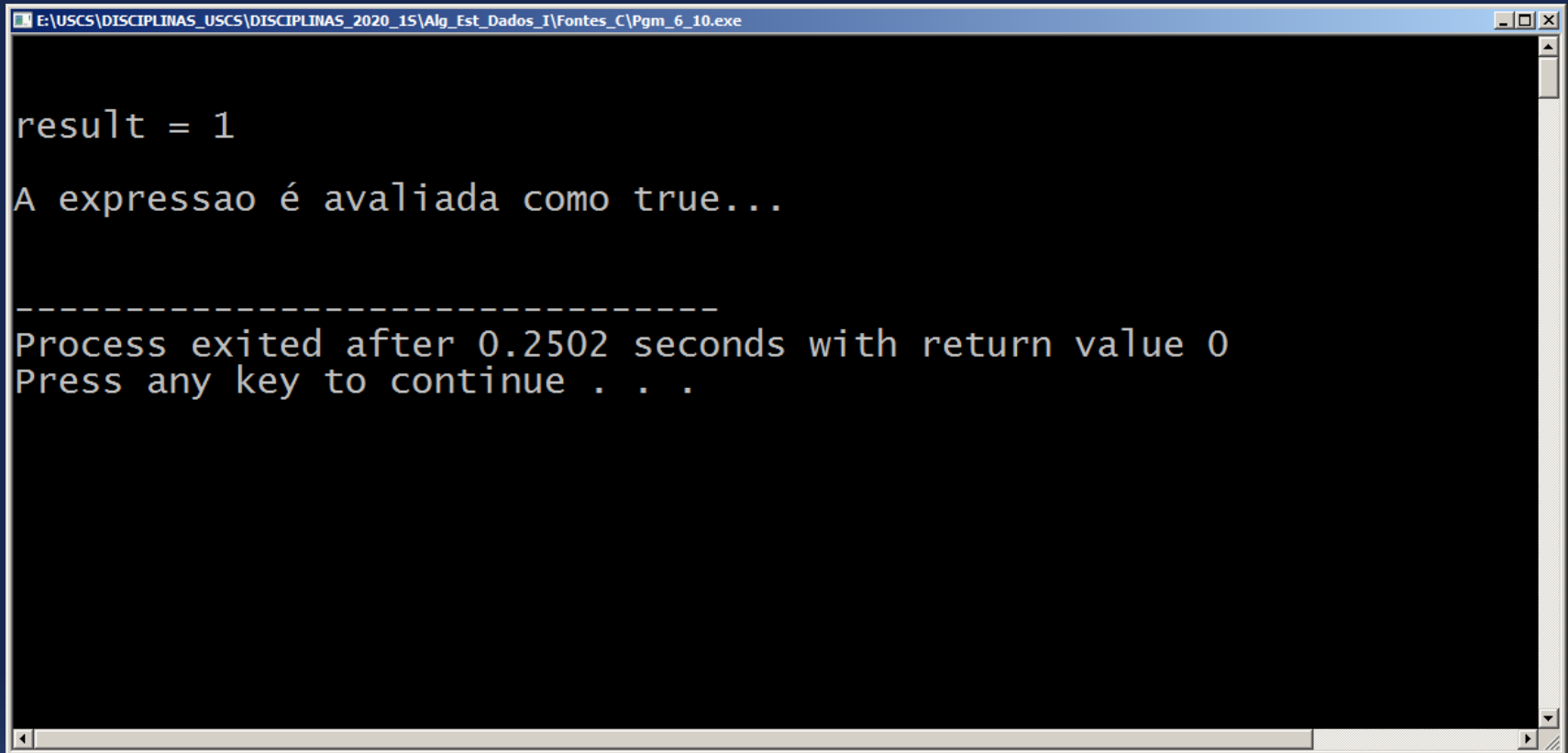
    int result = (i && j) && k;

    printf("\n\nresult = %d" , result);

    if (result)
        printf("\n\nA expressão é avaliada como true...\n\n");
    else
        printf("\n\nA expressão é avaliada como false...\n\n");

    return 0;
}
```

# Programa 10



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_10.exe

result = 1
A expressao é avaliada como true...

-----
Process exited after 0.2502 seconds with return value 0
Press any key to continue . . .
```



# Operadores Bitwise

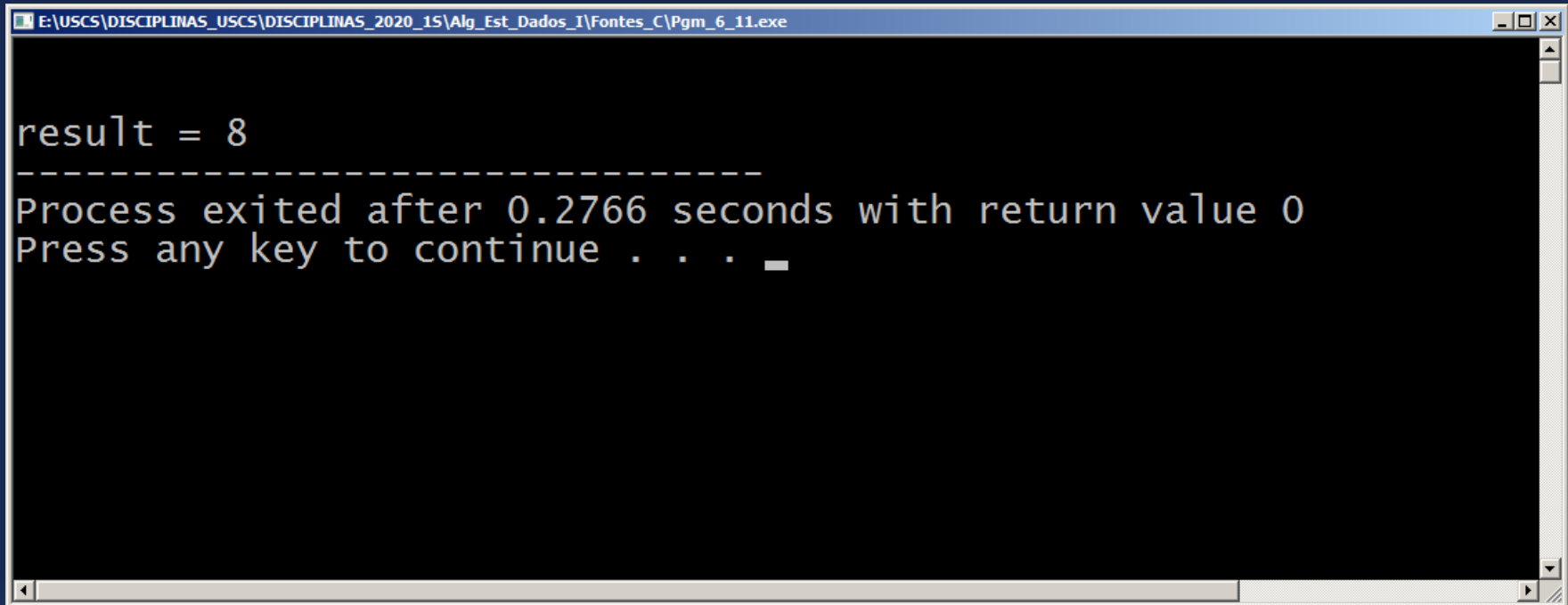
- ✓ São operadores utilizados quando se necessita executar operações a nível de **bits**, com números **inteiros**;
- ✓ Com esses operandores os operandos são considerados números binários;
- ✓ Assim, esses operadores atuam com operações **binárias**;
- ✓ Os operadores mais utilizados são:

Operador	Significado
&	Bitwise AND
	Bitwise OR
<<	Bitwise Left Shift
>>	Bitwise Right Shift
~	Bitwise Complement

# Operadores Bitwise – Programa 11

```
/* Unidade 6 - Programa 11 */  
#include <stdio.h>  
#include <locale.h>  
  
int main ( ) {  
    setlocale(LC_ALL, "Portuguese");  
  
    int i = 10, j = 12;  
  
    int result = (i & j);  
  
    printf("\n\nresult = %d" , result);  
  
    return 0;  
}
```

# Operadores Bitwise – Programa 11



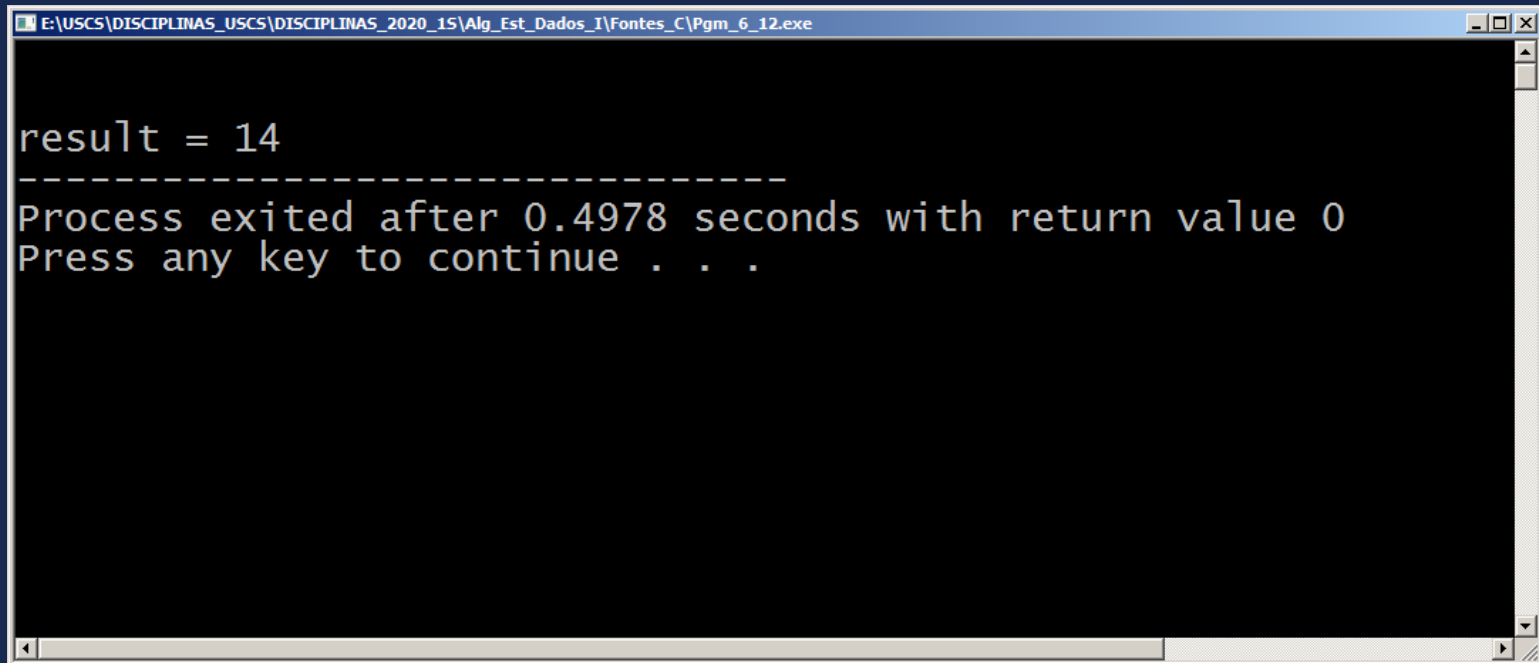
```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_11.exe

result = 8
-----
Process exited after 0.2766 seconds with return value 0
Press any key to continue . . . _
```

# Operadores Bitwise – Programa 12

```
/* Unidade 6 - Programa 12 */  
#include <stdio.h>  
#include <locale.h>  
  
int main ( ) {  
    setlocale(LC_ALL, "Portuguese");  
  
    int i = 10, j = 12;  
  
    int result = (i | j);  
  
    printf("\n\nresult = %d" , result);  
  
    return 0;  
}
```

# Operadores Bitwise – Programa 12



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_12.exe

result = 14
-----
Process exited after 0.4978 seconds with return value 0
Press any key to continue . . .
```

## Operadores Bitwise – Exemplo shift

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

int main ( ) {

    setlocale(LC_ALL, "Portuguese");

    int i = 20;

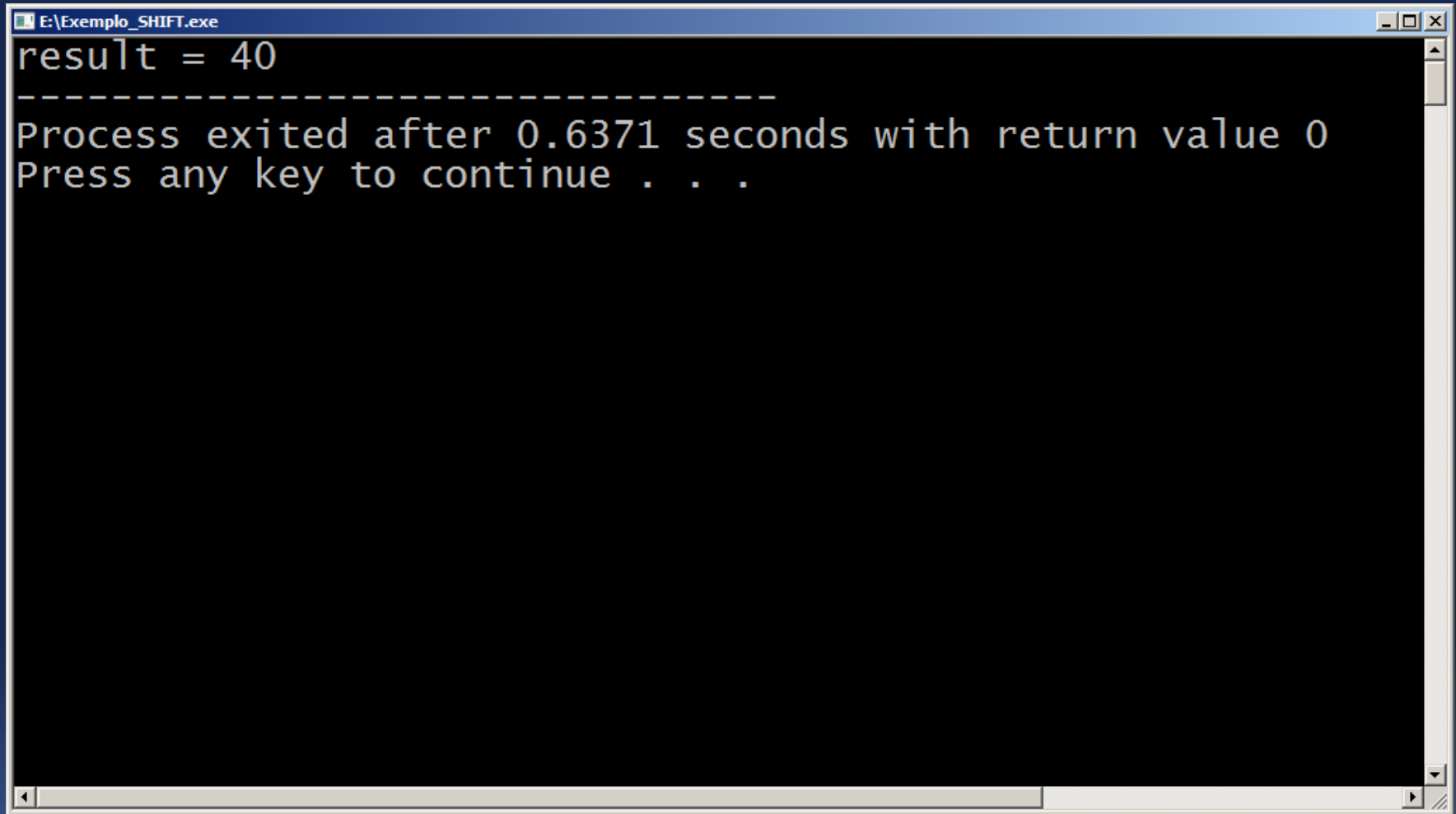
    int result;

    result = i << 1;

    printf ("result = %d", result);

    return 0;
}
```

# Operadores Bitwise – Exemplo shift



```
E:\Exemplo_SHIFT.exe
result = 40
-----
Process exited after 0.6371 seconds with return value 0
Press any key to continue . . .
```



## Operadores Bitwise – Exemplo shift

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

int main ( ) {

    setlocale(LC_ALL, "Portuguese");

    int i = 20;

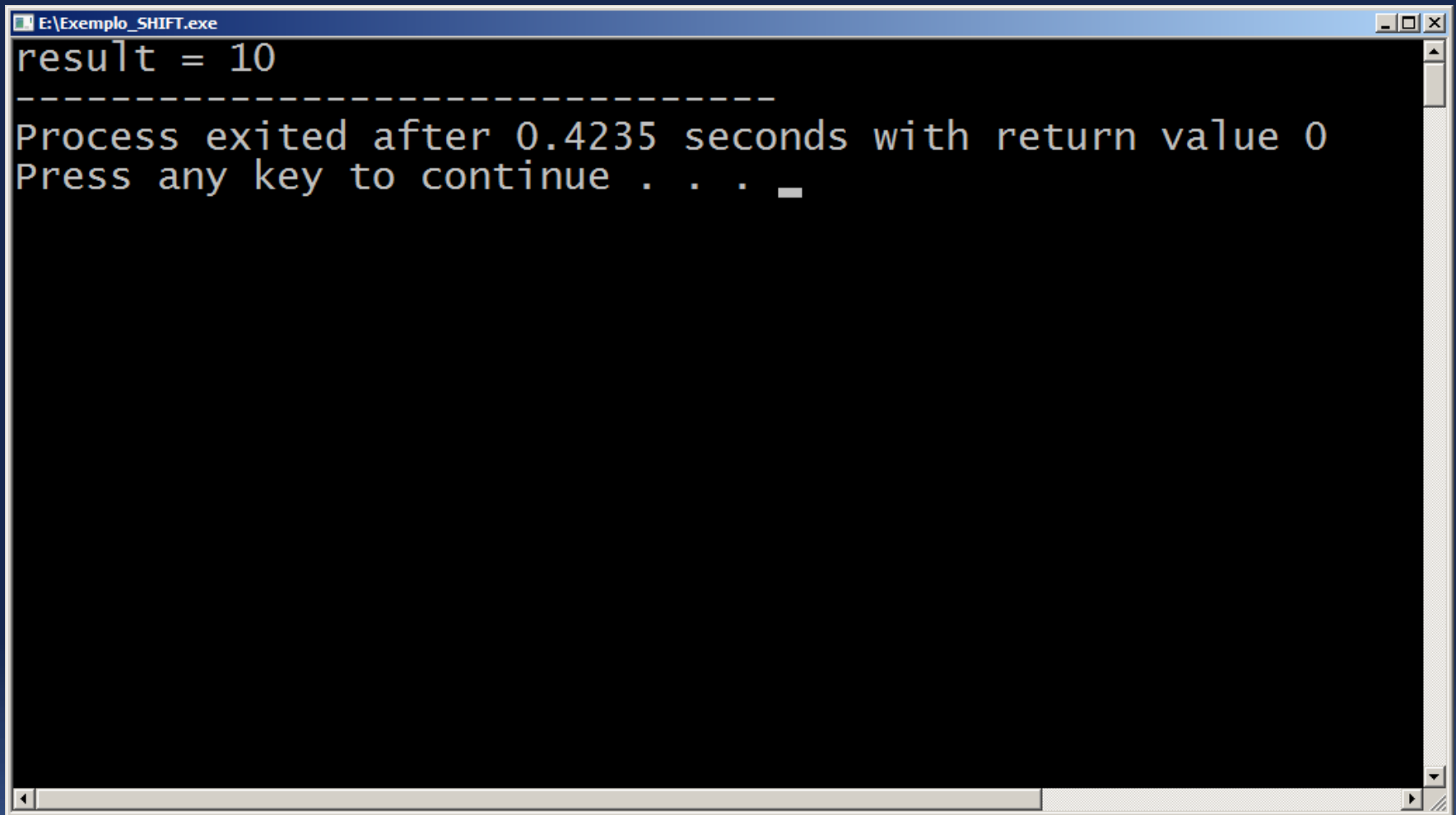
    int result;

    result = i >> 1;

    printf ("result = %d", result);

    return 0;
}
```

# Operadores Bitwise – Exemplo shift



```
E:\Exemplo_SHIFT.exe
result = 10
-----
Process exited after 0.4235 seconds with return value 0
Press any key to continue . . . _
```

# Operadores Bitwise – Exemplo – Complemento

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

int main ( ) {

    setlocale(LC_ALL, "Portuguese");

    int i = 1;

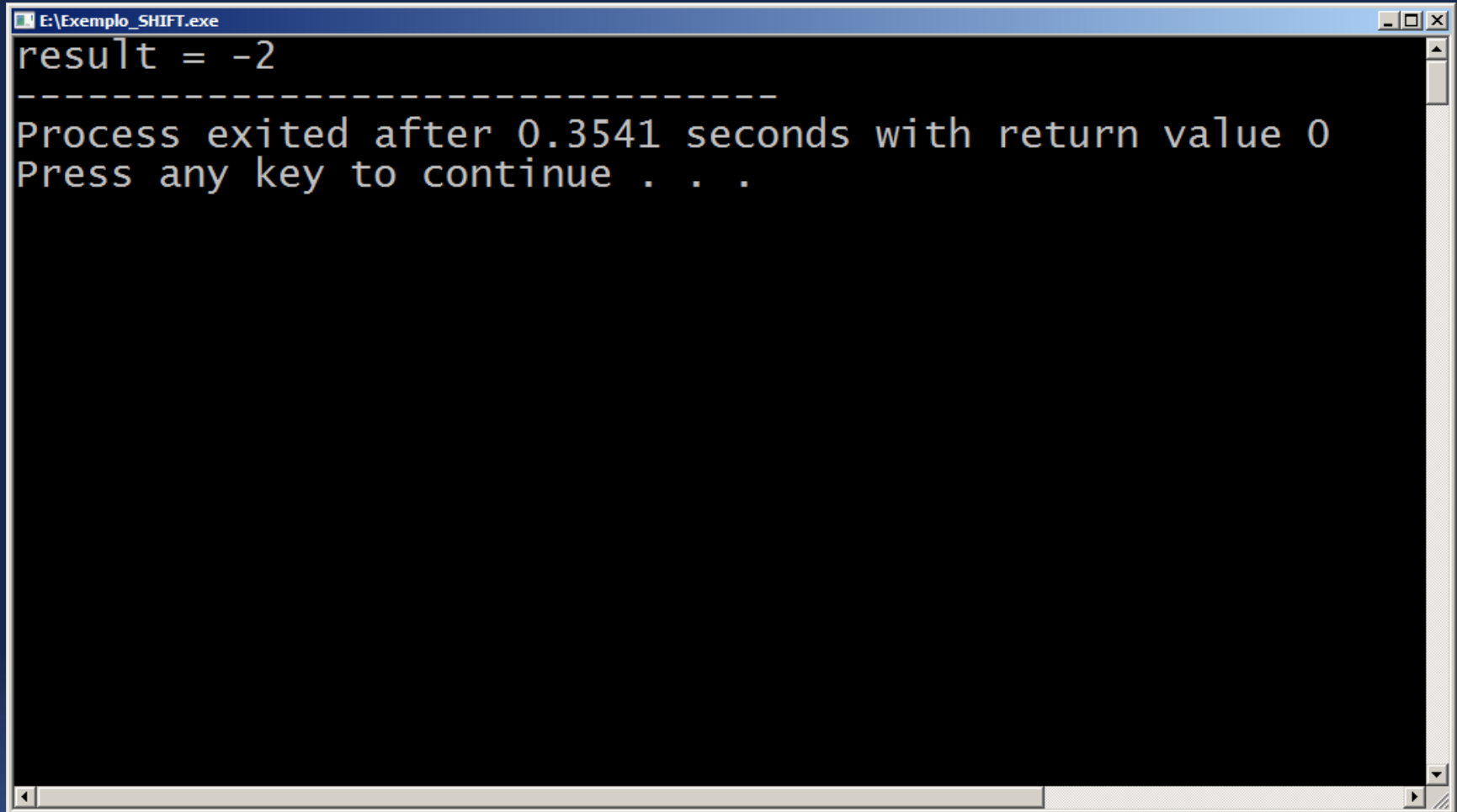
    int result;

    result = ~i;

    printf ("result = %d", result);

    return 0;
}
```

# Operadores Bitwise – Exemplo – Complemento



```
E:\Exemplo_SHIFT.exe
result = -2
-----
Process exited after 0.3541 seconds with return value 0
Press any key to continue . . .
```

# Comando Composto

- ✓ Um **comando composto** corresponde a um conjunto de declarações delimitadas por { e } (**Bloco**);
- ✓ O **bloco** é tratado como uma **unidade** de comandos de execução;
- ✓ Onde for sintaticamente correto colocar um comando também será colocar um **comando composto**.
- ✓ É correto assim, escrever-se:

```
{  
    a = 1;  
    {  
        b = 2;  
        c = 3;  
    }  
}
```

# Comando Vazio

- ✓ Escrito por um ";"
- ✓ Uma expressão seguida por um ";" é chamada expressão comando;
- ✓ O comando **vazio** é um caso particular de uma expressão comando.

# Comando Vazio – Programa 13

```
/* Unidade 6 - Programa 13 */
#include <stdio.h>
#include <locale.h>

int main ( ) {
    setlocale(LC_ALL, "Portuguese");

    int i = 20;

    int j;

    j = i; ; ;

    ;

    ;;;

    { ; ; ; }

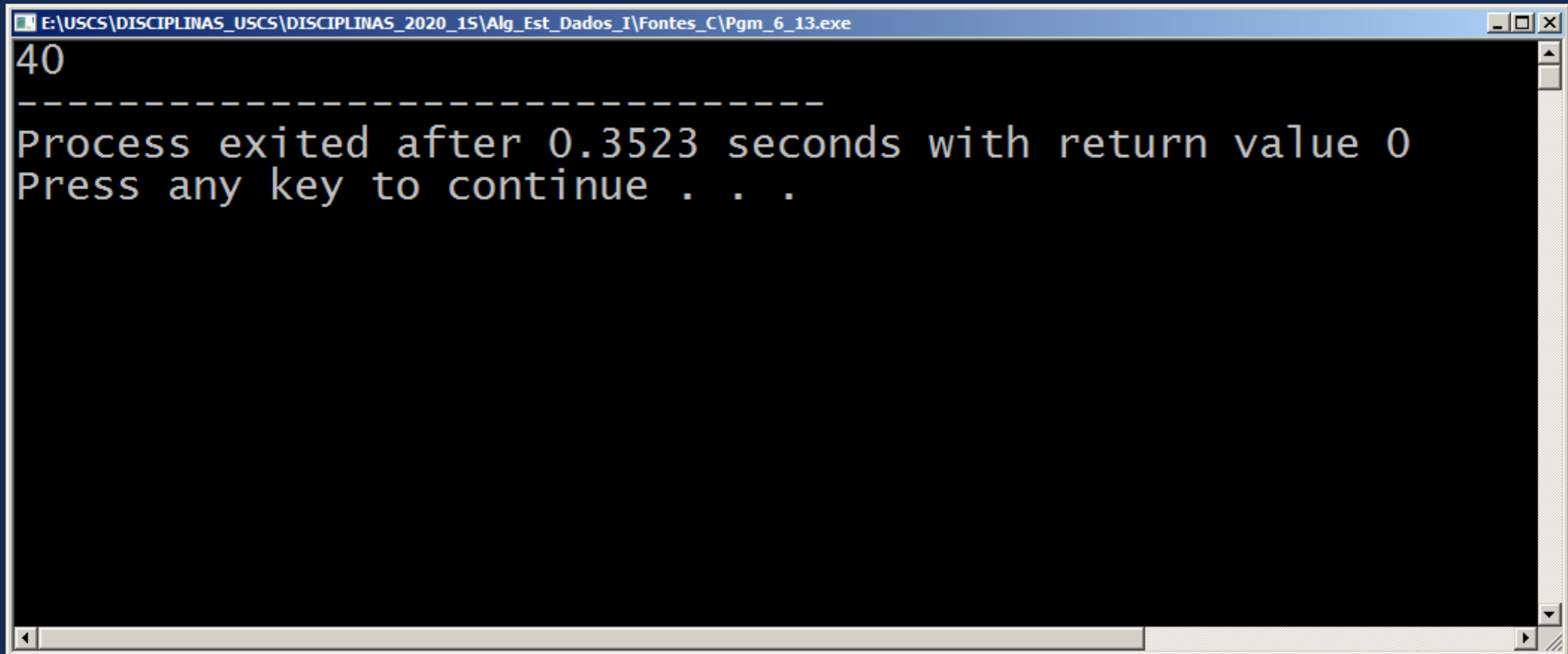
    ( i = j) ;

    (i == j) ; ; ; ;

    printf("%d", i + j) ;

    return 0;
}
```

# Comando Vazio



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Pgm_6_13.exe
40
-----
Process exited after 0.3523 seconds with return value 0
Press any key to continue . . .
```



# Comando if

```
if (expr)
    statement
```

```
if (expr)
    statement1
else
    statement2
```

# Comando while

```
while (expr)  
    statement  
next statement
```

# while

- ✓ **Repetição** de uma ação é uma das razões pelas quais confiamos em computadores;
- ✓ Quando há grandes quantidades de dados a serem processados, é muito conveniente usarmos mecanismos de controle que repetidamente executam instruções específicas de nosso código;
- ✓ O comando while primeiramente avalia a expressão a ele passada. Se for **diferente de zero (true)** o comando ou bloco de comandos é executado e o controle volta para o início do comando **while**.
- ✓ O processo de repetição é mantido até que a condição se torne igual a **zero (false)**.

# while

```
while (condição)  
    comando;
```

```
while (condição) {  
    comando1;  
    comando2  
    comando3;  
}
```

```
Exemplo:  
while (N != 0) {  
    scanf ("%d",&N);  
    if (N > MAIOR)  
        MAIOR = N;  
}
```

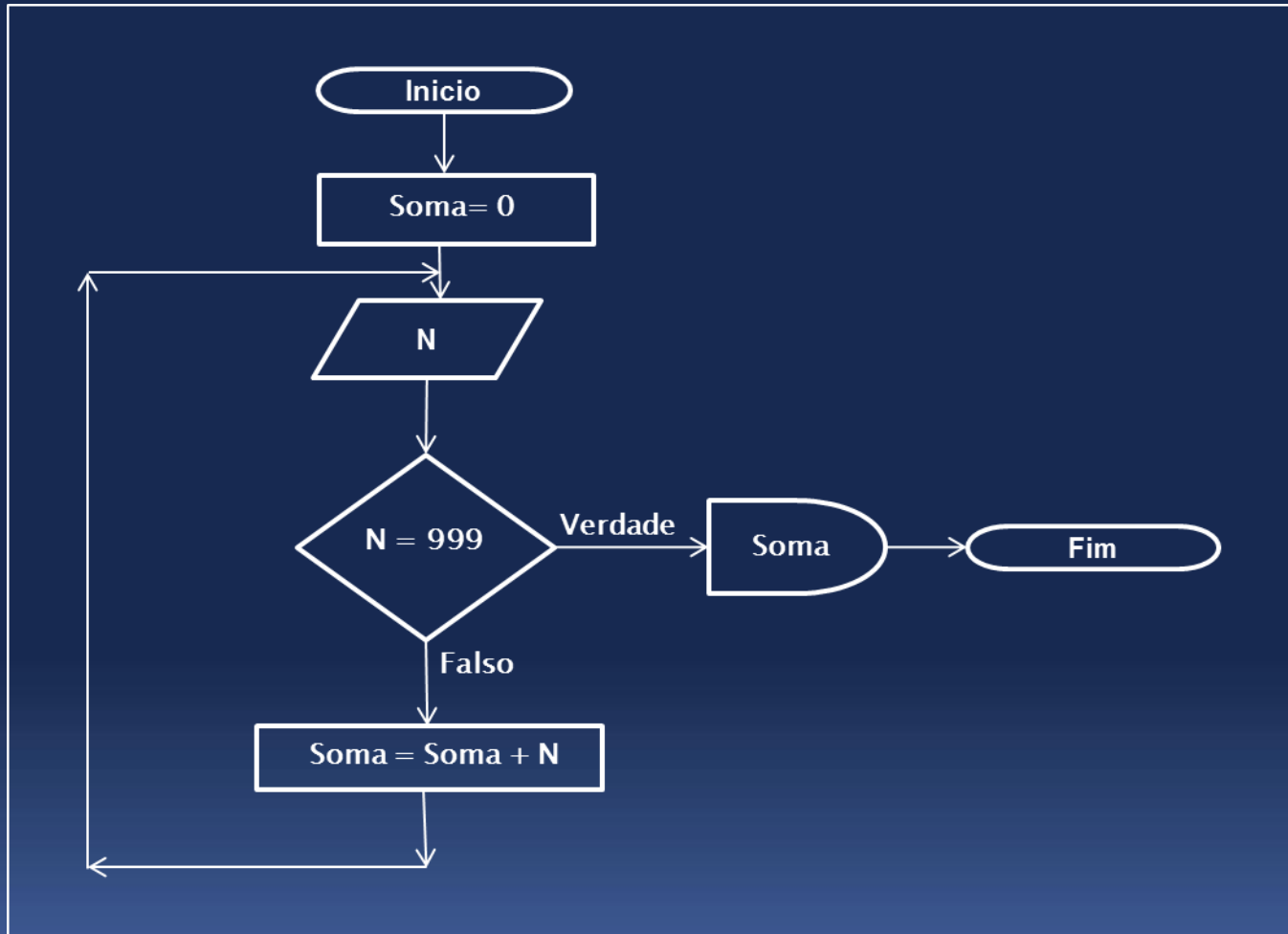
# Programa 14

Codificar um programa, com a **Linguagem C**, para a leitura de uma lista de valores numéricos inteiros.

Caso o valor entrado seja **999** o algoritmo deverá encerrar e exibir em tela o valor da **soma** dos valores entrados anteriormente.

Enquanto o usuário **não** digitar **999**, o algoritmo deverá acumular cada valor entrado e exibir a soma dos valores entrados.

# Programa 14



```
#include <stdio.h>
#include <locale.h>
#define true 1

//Unidade 6 - Programa 14

int main ( ) {
    setlocale(LC_ALL, "Portuguese");
    int soma = 0, n = 0;

    while(true) {
        printf ("\nInforme um valor inteiro qualquer ou 999 para encerrar! ");
        scanf("%d", &n);
        if (n == 999)
            break;
        else
            soma = soma + n;
    }

    printf("Soma = %d", soma) ;
    return 0;
}
```

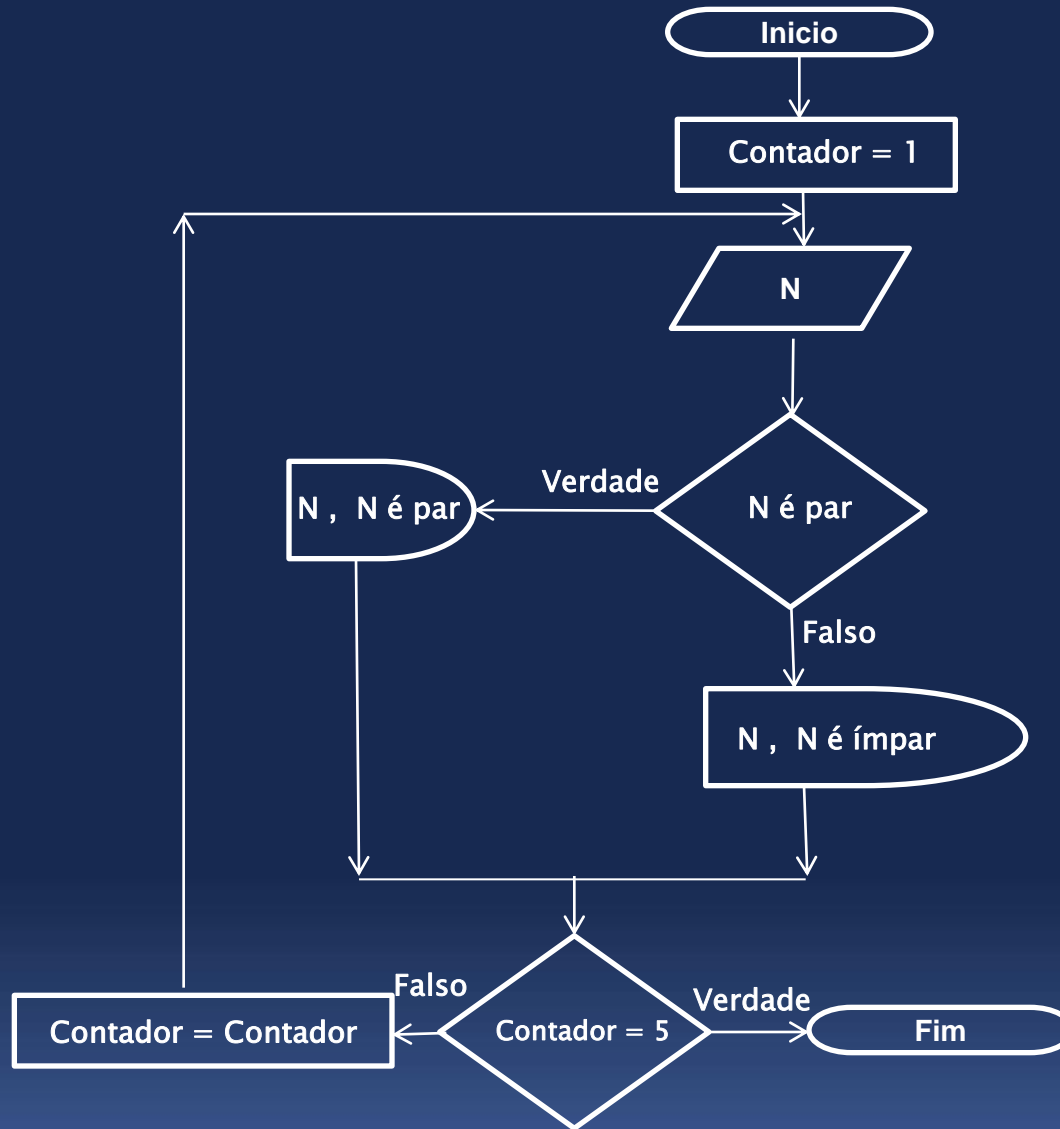
# Programa 15

**Codificar um programa, escrito na Linguagem C, para a leitura de uma lista de 5 valores numéricos inteiros.**

**Para cada valor entrado, o algoritmo deverá exibir uma mensagem informando o valor entrado e se o número é par ou ímpar.**



# Programa 15



# Programa 15



```
#include <stdio.h>
#include <locale.h>
#define true 1

//Unidade 6 - Programa 15

int main ( ) {
    setlocale(LC_ALL, "Portuguese");
    int contador = 1, n;

    while(true) {

        printf ("\nInforme um valor inteiro qualquer: ");
        scanf("%d", &n);

        if (n%2 == 0)
            printf ("%d, par\n", n);
        else
            printf ("%d, ímpar\n", n);

        if (contador == 5)
            break;

        contador++;

    }
    return 0;
}
```



# Comando for

```
for (expr1; expr2; expr3)  
    statement  
next statement
```

É equivalente a:

```
expr1;  
while (expr2) {  
    statement  
    expr3;  
}  
next statement
```

# Comando for

- ✓ A **primeira** expressão é **avaliada**. Tipicamente a primeira expressão é empregada para **inicializar** o looping;
- ✓ A **segunda** expressão é **avaliada**. Se o resultado dessa avaliação for diferente de zero (**true**) o comando ou bloco de comandos associado ao for é executado. Se for **zero**, o looping é **encerrado**;
- ✓ Após a execução do bloco de comandos associado ao for, a **terceira** expressão é avaliada e o controle volta para o início do looping.

# Comando for

```
for (var=valor inicial; condição; incremento)  
    comando;
```

```
for (var=valor inicial; condição; incremento) {  
    comando1;  
    comando2  
    comando3;  
}
```

Exemplo:

```
for (cont=3; cont<=11; cont++)  
    printf ("%d",cont);
```

# Exemplo – for

```
#include<stdio.h>

//Unidade 6 - Programa 16

int main() {

    int i, n = 5, result = 1, a = 2;

    for (i = 1 ; i <= n ; i++) {

        result = result * i;

        a = a + i;

    }

    printf("result = %d", result);
    printf("\na = %d", a);

    return 0;

}
```

# Comando do while

```
do {  
  
    statement ...  
}  
while (expressao);  
  
next statement;
```

- ✓ Equivalente ao comando **while**, porém o bloco sempre é executado pelo menos uma vez, pois a avaliação da expressão sempre é feita após a execução do bloco na primeira iteração do looping.

# Comando do while

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

```
do {  
    comando1;  
    comando2  
    comando3;  
} while (condição);
```

```
Exemplo:  
cont=0;  
do {  
    cont = cont + 1;  
    printf("%d\n",cont);  
} while (cont < 10);
```



# Comando do while

```
#include <stdio.h>

//Unidade 6 - Programa 17

int main ( ) {

    int a = 1;

    do {

        printf("\nUSCS");

        a = a + 1;

    } while (a < 5 );

}
```

# Comando switch – Exemplo

```
switch (val) {  
  case 1:  
    ++a_cnt;  
    break;  
  case 2:  
  case 3:  
    ++b_cnt;  
    break;  
  default:  
    ++other_cnt;  
}
```

- ✓ A variável **val** é avaliada. As condições estabelecidas nas cláusulas **case** são checadas e se atendidas os comandos a elas associados são executados;
- ✓ A checagem continua exceto quando um **break** for encontrado.

# Comando switch – Exemplo



```
#include <stdio.h>

int main( ) {

    int n;
    printf("Entre com um inteiro qualquer: ");

    scanf ("%d", &n) ;

    switch (n) {

        case 1:
            ++n;
            printf("USCS\n");
            printf("%d" , n);
            break;

        case 2:
            n = n + 10;
            printf("%d" , n);
            break;

        default:
            printf("\nSai pelo default....");

    }

}
```



# Arrays



- Trata-se de automatizar a declaração de um grande número de dados de um mesmo tipo simples. As variáveis assim declaradas são acessadas por meio de um índice de tipo **int**.

- **Declaração:**

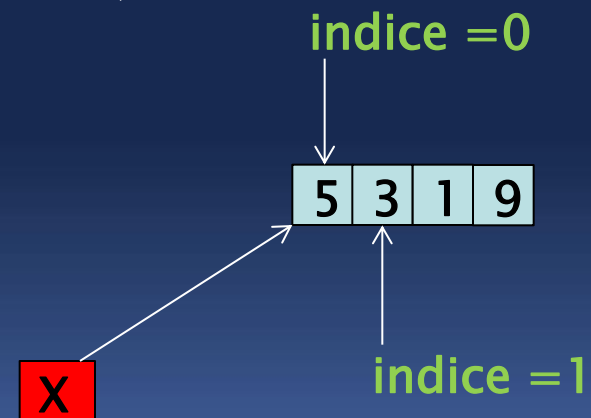
- ✓ `int v[100];`
- ✓ Índice da primeira posição = 0;
- ✓ Índice da última posição = 99;

- **Atribuição:**

- ✓ `v [9] = 87;`

- **Acesso a um valor do array:**

- ✓ `a = v[9];`



# Sem Arrays



```
#include <stdio.h>
int main()
{
    int contador=0;
    float  media, valor1, valor2, valor3, valor4, valor5;

    //scanf ("%f %f %f %f %f",
    //      &n1, &n2, &n3, &n4, &n5);

    printf ("\nEntre com o primeiro valor: ");
    scanf ("%f", &valor1);

    printf ("\nEntre com o segundo valor: ");
    scanf ("%f", &valor2);

    printf ("\nEntre com o terceiro valor: ");
    scanf ("%f", &valor3);

    printf ("\nEntre com o quarto valor: ");
    scanf ("%f", &valor4);

    printf ("\nEntre com o quinto valor: ");
    scanf ("%f", &valor5);

    media = (valor1+ valor2 + valor3 + valor4 + valor5)/5;

    if (valor1>media) contador++;
    if (valor2>media) contador++;
    if (valor3>media) contador++;
    if (valor4>media) contador++;
    if (valor5>media) contador++;

    printf ("\n\nMedia = %f  Total de valores acima da media: %d", media, contador);
    return 0;
}
```



# Com Arrays

```
#include <stdio.h>
int main() {

    printf("\n\n==== Inicio do Programa ==== \n\n");

    int i, contador=0;
    float soma=0, media;
    float v[5];

    for (i=0;i<5;i++) {
        printf("\nEntre com o valor: ");
        scanf ("%f", &v[i]);
        soma = soma + v[i];
    }

    media = soma/i;

    for (i=0;i<5;i++) {

        if (v[i]>media) contador++;

    }
    printf ("\n\nMedia: %f \n\nValores acima da media:  %d\n", media, contador);

    printf("\n\n==== Fim de Programa ==== \n\n");
}
```