



# Algoritmos e Estrutura de Dados – I

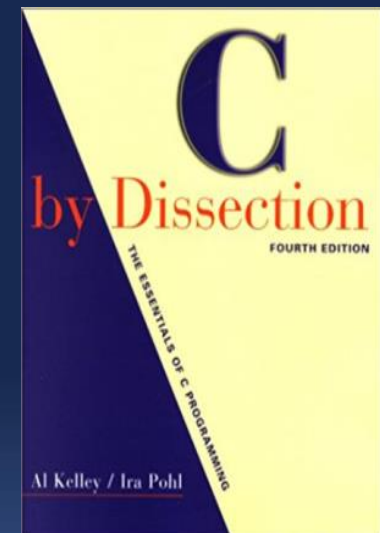
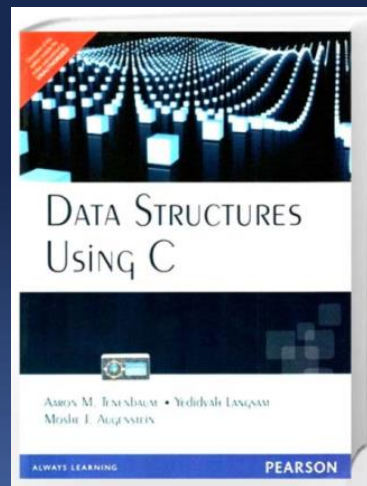
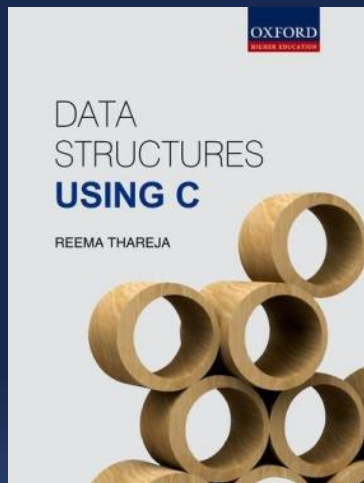
## Unidade 4 – Introdução à Linguagem C



Prof. Aparecido V. de Freitas  
Doutor em Engenharia  
da Computação pela EPUVSP  
[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, Pohl - Third Edition - Addison Wesley



# Introdução

- A linguagem C foi desenvolvida em meados de 1970 por Dennis Ritchie;
- Linguagem de uso geral;
- Empregada em diversas arquiteturas de computador;
- Usada como base para C++ e Java.



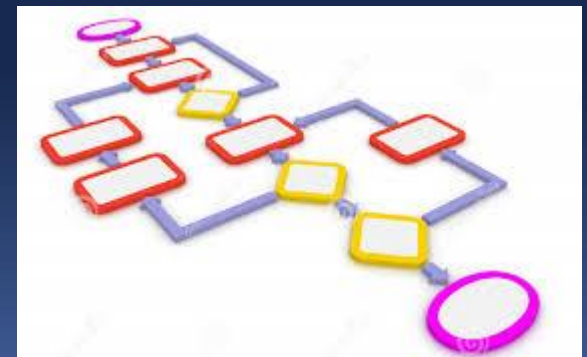
“

```
#include <stdio.h>
main()
{
    printf("hello, world\n");
}
```

Dennis M. Ritchie (1941-2011)

# Estrutura de um Programa C

- ✓ Um programa C contém **1** ou mais **funções**;
- ✓ Uma **função** é um **agrupamento** de **comandos** que executam uma tarefa bem definida;
- ✓ A função **main()** é a mais importante função de todo programa C, uma vez que todo programa em C se **inicia** a partir dessa **função**.

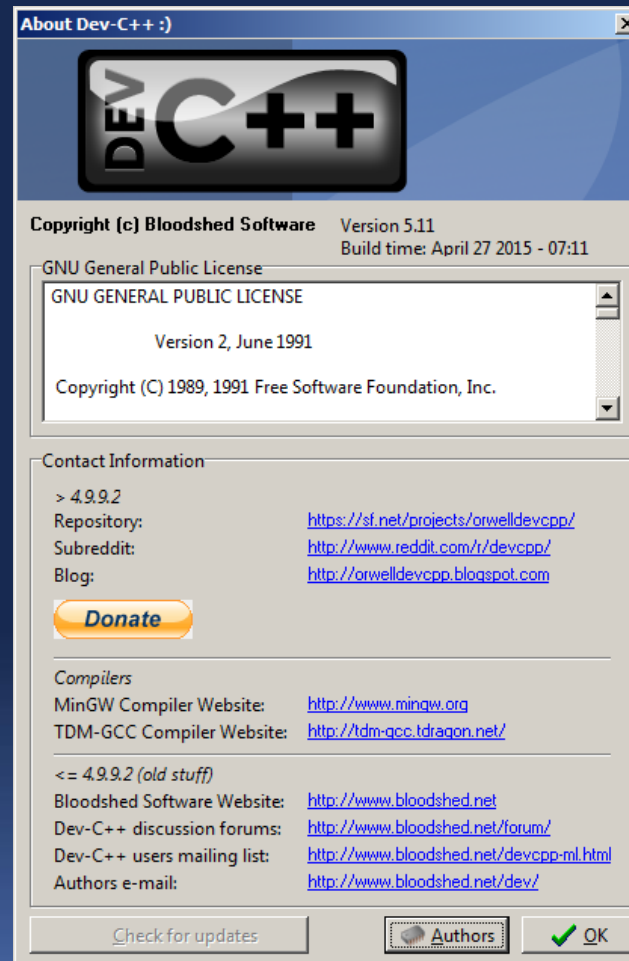


# Estrutura de um Programa C

```
main()
{
    Statement 1;
    Statement 2;
    .....
    .....
    Statement N;
}
Function1()
{
    Statement 1;
    Statement 2;
    .....
    .....
    Statement N;
}
Function2()
{
    Statement 1;
    Statement 2;
    .....
    .....
    Statement N;
}
.....
.....
FunctionN()
{
    Statement 1;
    Statement 2;
    .....
    .....
    Statement N;
}
```

# Primeiro programa C

## IDE Dev C++



# Primeiro programa C

## IDE Dev C++

sourceforge.net/projects/orwelldevcpp/

Trad OneDrive GDrive R T g Ms Prime aws Oracle ibm Foto libgen FStack Y F


**SOURCEFORGE**

Open Source Software Business Software Services Resources

**UMA OFERTA Especial PARA VOCÊ**

**GANHE 1 DE ISPIRAZIONE NA COMPRA**


Home / Browse / Development / Integrated Development Environments (IDE) / Dev-C++

 **Dev-C++**

A free, portable, fast and simple C/C++ IDE  
Brought to you by: [orwelldevcpp](#)

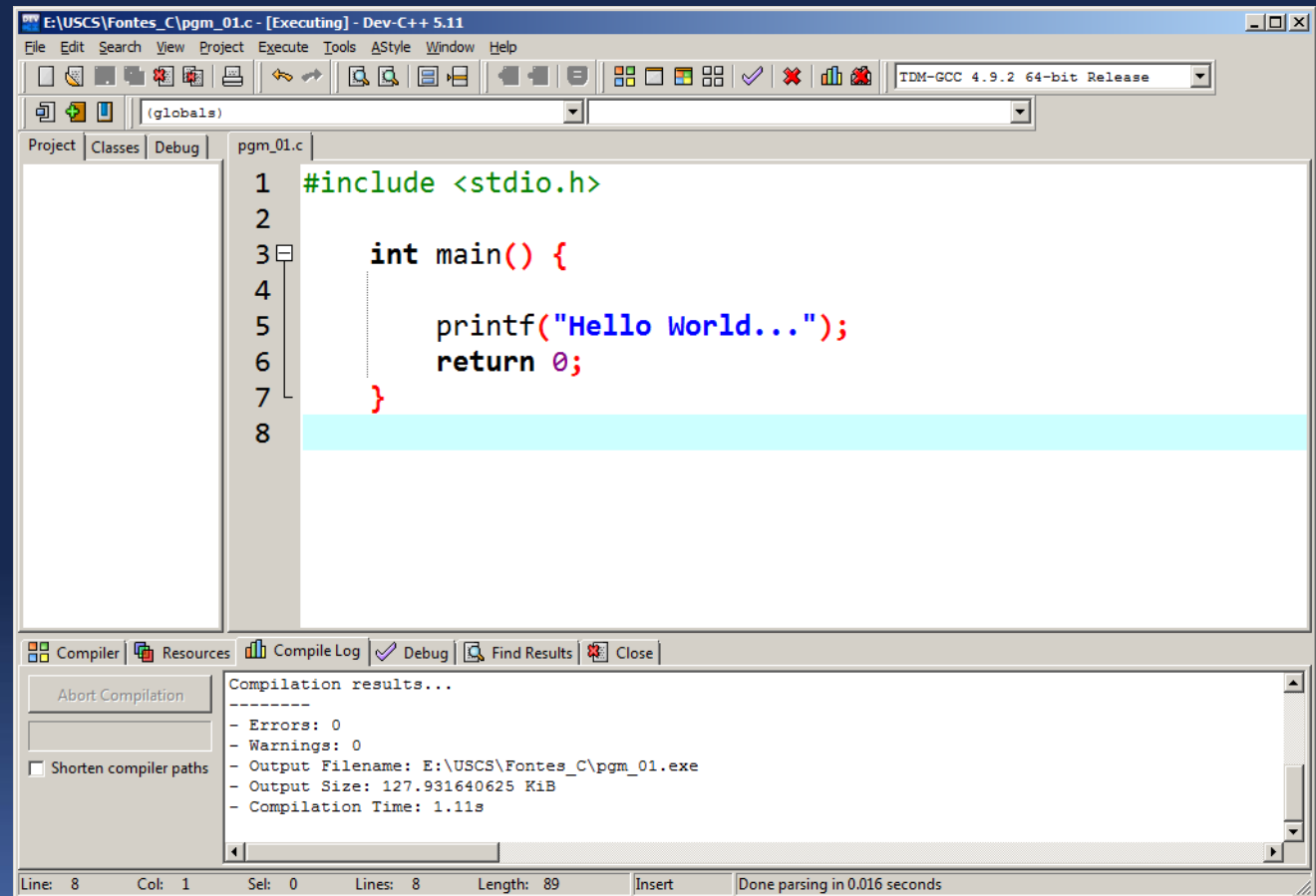
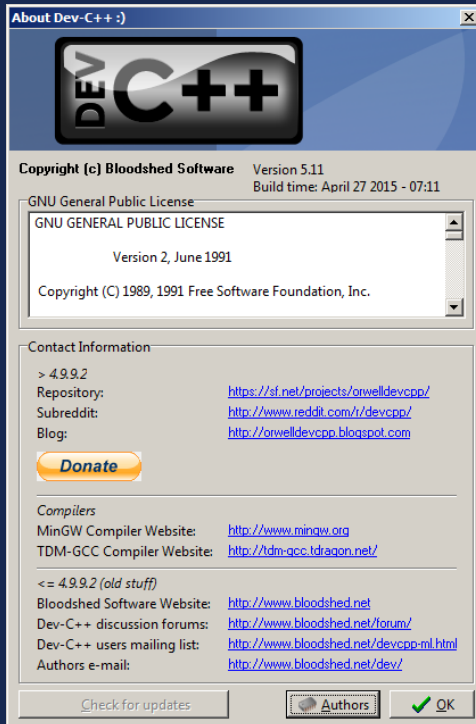
★★★★★ 141 Reviews

**Downloads: 92,764 This**

 **Download** **Get Updates** **Share This**

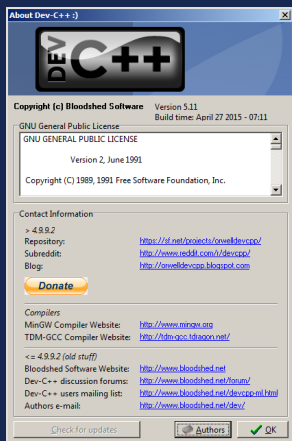
Windows | BSD

# Primeiro programa C IDE Dev C++





# Primeiro programa C IDE Dev C++



```
E:\USCS\Fontes_C\Programa_01.exe
Hello World....
-----
Process exited after 0.01734 seconds with return value 15
Press any key to continue . . .
```

# Primeiro programa C

## Compilador C – onlineGDB



The screenshot shows the onlineGDB web interface. On the left is a sidebar with navigation links: OnlineGDB beta, code.compile.run.debug.share, IDE, My Projects, Learn Programming, Programming Questions, Jobs (new), Sign Up, and Login. At the bottom of the sidebar are social media icons for Facebook and Twitter, and a button with a plus sign and '37.4K'. The main area displays a C program in a code editor. The code is as follows:

```

1  /*****
2
3      Online C Compiler.
4      Code, Compile, Run and Debug C program online.
5      Write your code in this editor and press "Run" button to compile
6
7      *****/
8
9  #include <stdio.h>
10
11 int main()
12 {
13     printf("Hello World");
14
15     return 0;
16 }
17
18

```

At the top of the code editor, there are buttons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The file name 'main.c' is shown in the top left of the editor area.

# Primeiro programa C

## Compilador C – onlineGDB

main.c

```
1  /*****
2
3      Online C Compiler.
4      Code, Compile, Run and Debug C program online.
5      Write your code in this editor and press "Run" button to compile
6
7      *****/
8
9  #include <stdio.h>
10
11  int main()
```

input

Hello World

...Program finished with exit code 0  
Press ENTER to exit console.

# Identificadores e Keywords

- Toda palavra escrita em C é um **identificador** ou uma **keyword**;
- **Identificadores** são nomes que o programador dá para elementos do programa tais como: **variáveis**, **arrays** e **funções**;
- **Identificadores** são formados por uma sequência de letras (maiúsculas ou minúsculas), numerais e underscores.



# Identificadores – Regras de Formação

- ✓ **Não** podem conter caracteres especiais, exceto **underscore** (**\_**)
- ✓ **Não** é permitido dois **underscores** em sequência;
- ✓ **Keywords** **não** podem ser utilizados como **identificadores**;
- ✓ A linguagem **C** é **case sensitive**;
- ✓ **Identificadores** devem começar com uma **letra** ou **underscore**;
- ✓ O tamanho de um identificador **não** deve ultrapassar **31 caracteres**.



# Keywords

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

# Tipos Básicos de Dados

- ✓ Um **tipo de dado** determina o **conjunto de valores** que um item pode conter e as operações que podem ser executadas com este item;
- ✓ A linguagem C possui **quatro** tipos de dados;
- ✓ O tipo **char** tem um byte de tamanho e é usado para armazenar caracteres simples;
- ✓ A linguagem C não provê qualquer tipo de dado para armazenar textos;

Data Type	Size in Bytes	Range	Use
char	1	-128 to 127	To store characters
int	2	-32768 to 32767	To store integer numbers
float	4	3.4E-38 to 3.4E+38	To store floating point numbers
double	8	1.7E-308 to 1.7E+308	To store big floating point numbers

# Um primeiro Programa

```
[*] Hello.c
1 #include <stdio.h>
2
3 int main(void) {
4
5     printf("Hello World...\n");
6     return 0;
7 }
8
```



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

- ✓ O caractere **#** indica uma **diretiva** de pré-processamento;
- ✓ A **diretiva include** inclui uma cópia do arquivo **header file stdio.h** no início do código do programa;
- ✓ Este **header file** é providenciado pelo **sistema C**;
- ✓ Os caracteres **< e >** indicam que o arquivo será copiado a partir da biblioteca padrão da **Linguagem C**;
- ✓ Esta **diretiva** foi necessária pois o programa está chamando a função **printf** cuja definição está no arquivo **stdio.h**.

```
int main(void)
```

- ✓ Todo programa em **C** tem uma função **main()**;
- ✓ Essa função determina o **início** da execução do código;
- ✓ Os **parênteses** indicam que **main** é uma função;
- ✓ A keyword **int** declara o **tipo** do dado que será retornado;
- ✓ A keyword **void** indica que a função **não** recebe argumentos.



- ✓ Um abre chaves indica o início do corpo de cada função;
- ✓ Um fecha chaves correspondente deve encerrar a função;
- ✓ O par { e } determina um bloco de código.

```
printf("Hello World...\n");
```

- ✓ A linguagem **C** contém uma biblioteca (library) de funções que podem ser usadas nos programas;
- ✓ A função **printf()** está exibindo um texto na **console** (tela);
- ✓ O **header** file **stdio.h** fornece informações para o compilador processar a compilação do código.

```
printf("Hello World...\n");
```

- ✓ A função `main()` está chamando a função `printf()` e passando a ela o argumento `"Hello World...\n"`;
- ✓ Em `C` um `string` é uma série de caracteres delimitados entre `" "`;
- ✓ Os caracteres `\n` representam uma informação de controle para a console indicando uma `quebra de linha` (`newline`);
- ✓ Toda declaração em `C` termina com `;`.

```
printf("Hello World...\n");
```

- ✓ A função `main()` está chamando a função `printf()` e passando a ela o argumento `"Hello World...\n"`;
- ✓ Em `C` um `string` é uma série de caracteres delimitados entre `" "`;
- ✓ Os caracteres `\n` representam uma informação de controle para a console indicando uma `quebra de linha` (`newline`);
- ✓ Toda `declaração` em `C` termina com `;`.

```
return 0;
```

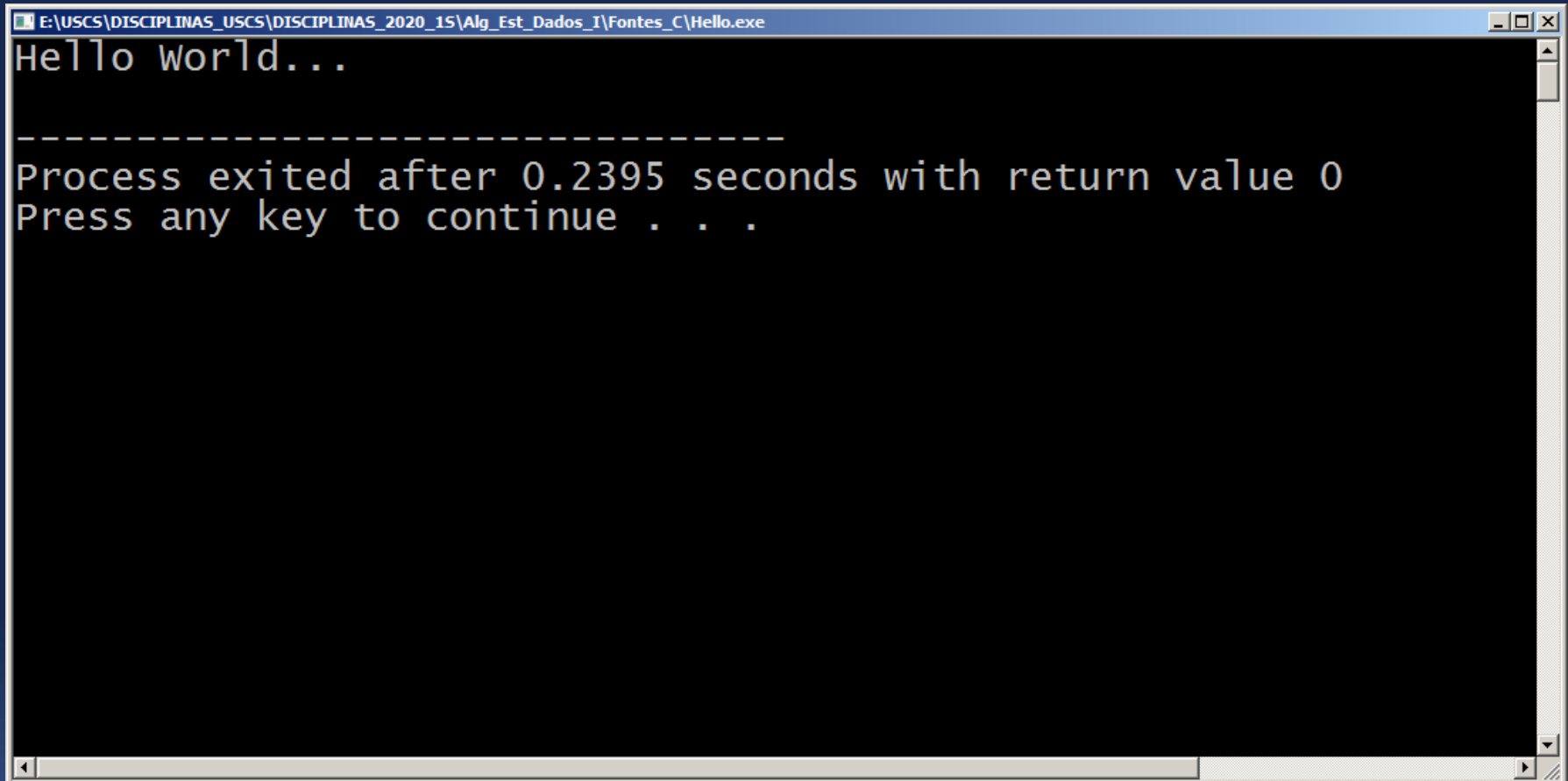
- ✓ O valor inteiro 0 é retornado pela função `main( )` para o sistema operacional;
- ✓ O valor 0 indica que o programa terminou com sucesso;
- ✓ Valores diferentes de zero retornados ao sistema operacional indicam que a função `main( )` foi executada sem sucesso.



- ✓ O caracter } indica **fechamento** de bloco e, dessa forma, o **encerramento** da definição da função `main( )`;



# Executando o código



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\Hello.exe
Hello world...
-----
Process exited after 0.2395 seconds with return value 0
Press any key to continue . . .
```

## Reescrevendo o código

```
#include <stdio.h>

int main(void) {

    printf("Hello, ");
    printf(" World...\n");
    return 0;
}
```

- ✓ Observe que o **string** usado como argumento do primeiro **printf** terminou com um caractere em branco e, portanto, nesse primeiro **printf** não haverá salto de linha.

# Nomes

- ✓ Um **nome** ou um **identificador** é uma cadeia de caracteres usada para se identificar alguma **entidade** em um programa;
- ✓ Na Linguagem **ANSI C**, o tamanho de um nome deve ser no máximo de **31 caracteres**;
- ✓ Nomes devem iniciar com **letras** ou **\_** (**underline**) e podem ser seguidos por **letra**, **\_**, ou **dígitos**.

# Nomes – Palavras Reservadas

- ✓ Palavra-reservada: palavra que não pode ser usada como um nome definido pelo programador;
- ✓ Exemplos: **if** , **for** , **while** , **do** , **int** , **break** , etc....

# Variáveis

- ✓ Variável é um **abstração** de uma célula de memória;
- ✓ Surgiram durante a **mudança** das linguagens de programação de **baixo** nível para alto **nível**;
- ✓ Pode ser caracterizada por alguns atributos:
  - ❖ **Nome**: identificador;
  - ❖ **Endereço**: localização da memória a ela associada;
  - ❖ **Tipo**: intervalo de valores possíveis e operações
  - ❖ **Valor**: o que está armazenado na variável em um determinado instante;
  - ❖ **Tempo de vida**: tempo no qual a memória permanece alocada à variável;
  - ❖ **Escopo**: partes do programa onde a variável é acessível.

# Tipos de dados

- ✓ Um **tipo de dado** define uma **coleção** de **dados** e um conjunto de **operações** pré-definidas sobre esses dados;
- ✓ A linguagem **C** fornece uma coleção apropriada de tipos de dados;
- ✓ Entender o **sistema de tipos** de uma linguagem de programação é um dos aspectos mais importantes para entender a sua **semântica**.

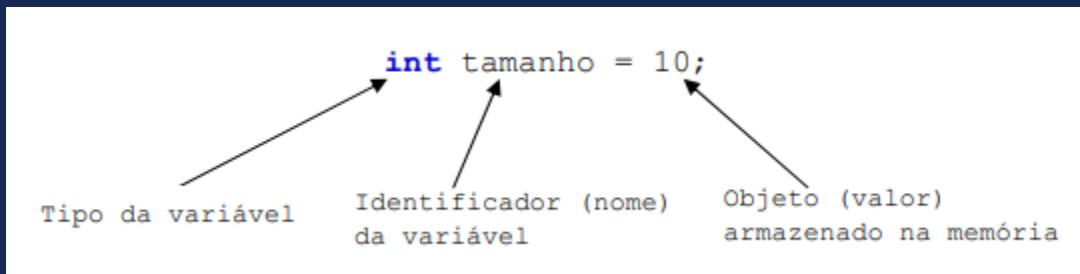
# Tipos de Dados

- Todas as variáveis em C tem um **tipo**;
- Cada tipo define os valores que a variável pode armazenar;
- Cada tipo ocupa uma certa quantidade de memória.

Tipo	Valores Válidos
char	letras e símbolos: 'a', 'b', 'H', '\n', '*', '1', '0'
int	de -32767 até 32767 (apenas números inteiros)
float	de $-3.4 \times 10^{38}$ até $+3.4 \times 10^{38}$ com até 6 dígitos de precisão
double	de $-1.7 \times 10^{308}$ até $+1.7 \times 10^{308}$ com até 10 dígitos de precisão

# Declaração de Variáveis

- Todas as variáveis **tem que ser declaradas *antes*** de serem usadas;
- Não há uma inicialização implícita na declaração





# Declaração de Variáveis

```
// Exemplo de programa em C

#include <stdio.h>    // Arquivo de cabeçalho (header)
void main()
{
    int contador;           // declarações simples
    float PrecoDoQuilo;
    double TaxaDeCambio;
    char LetraDigitada;
    int IdadeManoel, IdadeJoao, IdadeMaria; // Pode colocar mais de uma variável na
                                           // na mesma linha

    double TaxaDoDolar,
           TaxaDoMarco,
           TaxaDoPeso,      // Também pode trocar de linha no meio
           TaxaDoFranco;

    .....
}
```

# Operações Aritméticas

- ✓ Principais operadores aritméticos:

+ - \* / %

- ✓ Estes operadores são binários;
- ✓ Em C, uma expressão **inteira** **dividida** por outra expressão **inteira** resulta em um valor **inteiro**. A parte decimal é descartada nesse caso;
- ✓ Assim,  $7/2$  tem o valor 3,  $18/4$  tem o valor 4;
- ✓ **Divisão por zero não é permitida (Operação ilegal).**

# Operações de Atribuição

- ✓ Comandos de **atribuição** são definidos pelo sinal de **=**;
- ✓ O valor da avaliação da expressão à direita é atribuído à variável definida no lado esquerdo;
- ✓ Exemplo:

```
a = 10 * b;
```
- ✓ O valor da variável **b** é multiplicado por **10** e o resultado é atribuído à variável **a**;
- ✓ O símbolo **\*** está representando a operação de multiplicação.

## Exercício 1 – Pgm\_01.c

- ✓ Declarar uma variável **A** com o valor **10**;
- ✓ Declarar uma variável **B** com o valor **20**;
- ✓ Declarar uma variável **C** que recebe a soma de **A** e **B**;
- ✓ Imprimir na Console o valor da variável **C**.

## Exercício 1 - Pgm\_01.c

```
PGM_01.c
1  #include <stdio.h>
2
3  int main ( ) {
4
5      int  A = 10;
6      int  B = 20;
7      int  C = A + B;
8
9      printf("C = %d", C);
10
11     return 0;
12 }
```

## Exercício 1 - Pgm\_01.c

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\PGM_01.exe
C = 30
-----
Process exited after 0.3405 seconds with return value 0
Press any key to continue . . . _
```

# Função printf()

- ✓ A função printf() admite mais de um argumento;
- ✓ Por exemplo: `printf("A = %d", A);`
- ✓ No exemplo acima, o primeiro argumento é "A = %d";
- ✓ O segundo argumento é A;
- ✓ **Primeiro** argumento é sempre um **string**. No exemplo, o string contém **%d** que é uma especificação de conversão, também chamada de formato. Nesse caso, a variável definida no segundo argumento será convertida para decimal inteiro

## Exercício 2 – Pgm\_O1.c

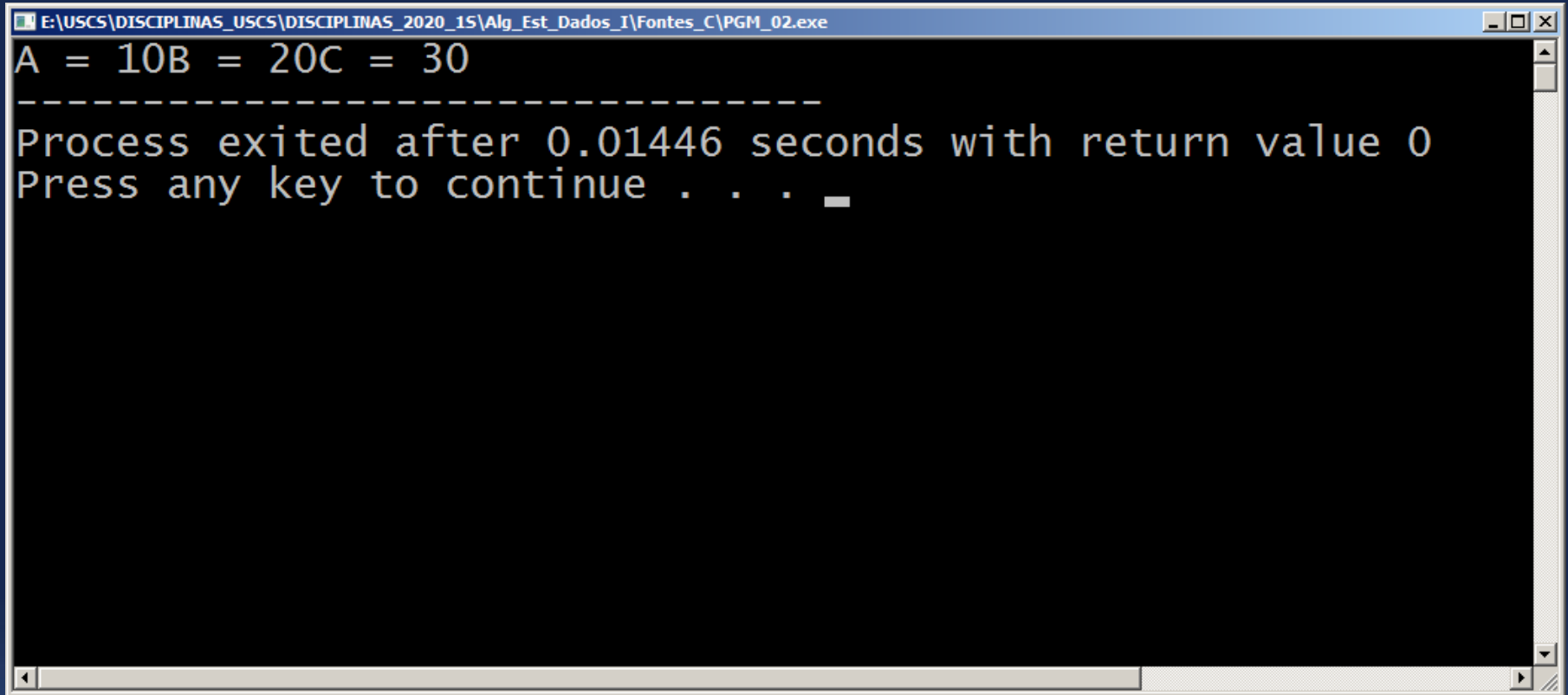
- ✓ Declarar uma variável inteira **A** com o valor **10**;
- ✓ Declarar uma variável inteira **B** com o valor 20;
- ✓ Declarar uma variável inteira **C** que recebe a soma de **A** e **B**;
- ✓ Imprimir na Console o valor das variáveis **A**, **B** e **C**.



## Exercício 2 – Pgm\_O2.c

```
PGM_O2.c
1  #include <stdio.h>
2
3  int main ( ) {
4
5      int  A = 10;
6      int  B = 20;
7      int  C = A + B;
8
9
10     printf("A = %d", A);
11     printf("B = %d", B);
12     printf("C = %d", C);
13
14     return 0;
15 }
```

## Exercício 2 - Pgm\_02.c



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\PGM_02.exe
A = 10B = 20C = 30
-----
Process exited after 0.01446 seconds with return value 0
Press any key to continue . . . _
```

## Exercício 2 – Revisitado – Pgm\_O2.c

```
PGM_02.c
1  #include <stdio.h>
2
3  int main ( ) {
4
5      int  A = 10;
6      int  B = 20;
7      int  C = A + B;
8
9
10     printf("A = %d\n", A);
11     printf("B = %d\n", B);
12     printf("C = %d\n", C);
13
14     return 0;
15 }
```

## Exercício 2 – Revisitado – Pgm\_02.c

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\PGM_02.exe
A = 10
B = 20
C = 30

-----
Process exited after 0.4698 seconds with return value 0
Press any key to continue . . .
```

# Formato Geral de um programa C

diretivas de pré-processamento

```
int  main (void) {
```

declarações ...

comandos...

```
}
```

# Tipo char

- ✓ A keyword **char** significa "character";
- ✓ Variáveis e constantes do **tipo** char são utilizadas para manipular **caracteres**;
- ✓ Constantes do tipo **char** são definidas entre ' ', como 'A' e '1' e '+';

## Exercício 3 – Pgm\_03.c

- ✓ Declarar uma variável **A** com o valor 'a';
- ✓ Imprimir na Console o valor da variável **A**.

## Exercício 3 - Pgm\_O3.c

```
#include <stdio.h>

int main ( ) {

    char  A = 'a';

    printf("A = %c\n", A);

    return 0;
}
```



✓ Observe que na função **printf** usou-se o string de controle **%c** para indicar que será impresso um caractere.



## Exercício 3 - Pgm\_03.c

### Console de execução



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\PGM_03.exe
A = a
-----
Process exited after 0.02933 seconds with return value 0
Press any key to continue . . .
```

# Tipos ponto flutuante

- ✓ Representam **números** com **casas decimais** (formato ponto flutuante);
- ✓ Ao se representar números reais, o padrão é **double**;
- ✓ Assim, **1.055** e **0.0023** são números **double**;
- ✓ Constantes do tipo **float** são representadas anexando-se um sufixo **F** ao final do número, como **5.76F**.

## Exercício 4 – Pgm\_04.C

- ✓ Declarar uma variável **X** com o valor **1.0**; (**double**)
- ✓ Declarar uma variável **Y** com o valor **7.0**; (**double**)
- ✓ Declarar uma variável **Z** que recebe soma de **X** e **Y**;
- ✓ **Imprimir** na Console o valor da variável **Z**.

## Exercício 4 – Pgm\_04.C

```
#include <stdio.h>

int main ( ) {

    double X = 1.0;

    double Y = 7.0;

    double Z = X + Y;

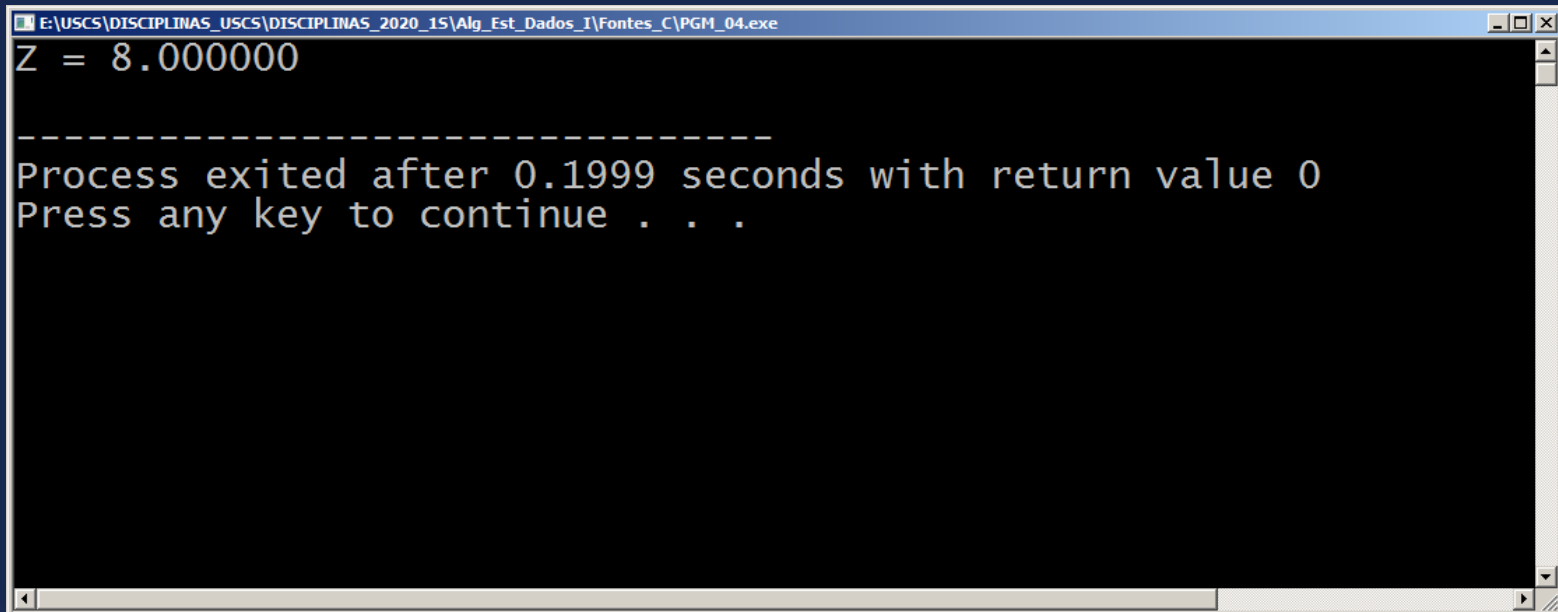
    printf("A = %f\n", Z);

    return 0;
}
```



✓ Observe que na função **printf** usou-se a especificação de conversão **%f** para indicar que será impresso valor em ponto flutuante.

# Console de Execução



```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\PGM_04.exe
Z = 8.000000
-----
Process exited after 0.1999 seconds with return value 0
Press any key to continue . . .
```

## Exercício 5 – Pgm\_05.C

- ✓ Declarar uma variável **X** com o valor **5.0**; (**float**)
- ✓ Declarar uma variável **Y** com o valor **7.005**; (**float**)
- ✓ Declarar uma variável **Z** que recebe soma de **X** e **Y**;
- ✓ Imprimir na Console o valor da variável **Z**.

## Exercício 5 – Pgm\_05.C

```
#include <stdio.h>

int main ( ) {

    float X = 5.0F;

    float Y = 7.005F;

    float Z = X + Y;

    printf("Z = %f\n", Z);

    return 0;
}
```

# Diretivas de Pré-processamento

- ✓ Antes do compilador C compilar nosso programa, o **pré-compilador** trabalha, **modificando** o nosso código fonte de acordo com as **diretivas** definidas pelo programador;
- ✓ Por exemplo, arquivos podem ser **incluídos** em nosso fonte, ou determinados strings de caracteres podem ser **modificados** para outros valores;
- ✓ As linhas de código que instruem o pré-compilador são chamadas **diretivas de pré-processamento**, e são iniciadas pelo caractere **#**.



# Diretivas de Pré-processamento

✓ Exemplos:

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

```
#include "arq.txt"
```

```
#define LIMITE 100
```

```
#define PI 3.1416
```

## Exercício 6 – Pgm\_06.C

- ✓ Declarar uma variável **R** que representa o valor do **Raio** de um círculo. Considerar que **R** tenha o valor **10.0**;
- ✓ Declarar, por meio de uma **diretiva de pré-processamento**, o valor da constante **PI = 3.141519**; (**#define PI 3.141519**)
- ✓ Declarar uma variável **A** que irá armazenar o valor da **área** do círculo.
- ✓ Imprimir na console o valor de **A**.

## Exercício 6 – Pgm\_06.C

```
#include <stdio.h>
#define PI 3.141519

int main ( ) {

    double R = 10.0;

    double A = PI * R * R;

    printf("Area = %f\n", A);

    return 0;
}
```

# Exercício 6 – Pgm\_06.C

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\PGM_06.exe
Area = 314.151900

-----
Process exited after 2.623 seconds with return value 0
Press any key to continue . . .
```

# Função printf()

- ✓ Usada, como vimos, para **imprimir saída** para a console;
- ✓ Faz parte da **biblioteca padrão** da Linguagem C;
- ✓ A sintaxe da função é:

```
printf(string_de_controle, lista_de_argumentos);
```

- ✓ A **string de controle** mostra **não** apenas os caracteres a serem impressos, mas também as **variáveis** e suas respectivas **posições**;
- ✓ A **formatação** das variáveis no texto é definida por caracteres de controle, com o uso da notação **%**.

## Função printf() – Caracteres de Controle

Caractere de Controle	Significado
%d	Inteiro
%f	Ponto Flutuante
%c	Caractere
%s	String

# Exercício 7 – Pgm\_07.C

```
#include <stdio.h>

int main ( ) {

    int A = 10;
    float B = 5.8;
    double C = 5.97;
    char X = 'x';

    printf ("USCS - Computacao\n\n");
    printf ("%d%c%f%s",40,'a',3.45," Hello World...\n\n");
    printf ("Um caractere %c e um inteiro %d\n\n",X,A);
    printf ("Um float %f e um double %f\n\n",B,C);

    return 0;
}
```

# Exercício 7 – Pgm\_07.C

```
E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Alg_Est_Dados_I\Fontes_C\PGM_07.exe
USCS - Computacao
40a3.450000 Hello world...
Um caractere x e um inteiro 10
Um float 5.800000 e um double 5.970000

-----
Process exited after 0.4678 seconds with return value 0
Press any key to continue . . .
```