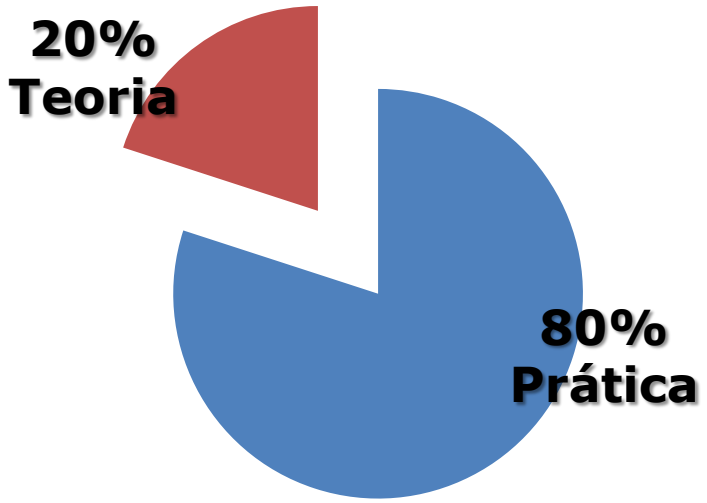
# Curso básico de Linguagem C

# Introdução

# Sobre o curso

- O curso será apresentado em 5 (cinco) encontros, totalizando 20 (vinte) horas/aula.

- Apresentará uma introdução a linguagem C.



# Objetivos

- Introduzir o aluno à sintaxe de desenvolvimento da linguagem C;
- Apresentar a estrutura básica de um programa em C;
- Apresentar regras fundamentais de boa prática de programação;
- Tornar o aluno apto a compreender e criar seus próprios códigos em C;

# Material de Apoio

- "Curso de Linguagem C" – UFMG - http://www.ead.eee.ufmg.br/cursos/C/
- "C How to program" – 5ª. Ed. – Deitel & Deitel
- "C Completo e Total" – 3ª. Ed. – Herbert Schildt

# Linguagem C

```
#include <stdio.h>

int main() {

    for { int count = 1; count <= 1000; count++ ) {

        printf("Eu aprenderei C neste curso com o Ricardo!");

    }

return 0;

}
```

# Sobre a Linguagem

- Surgiu na década de 70, criado por Dennis Ritchie.
- Uma das suas vantagens é possuir características tanto de "alto nível" quanto de "baixo nível".
- Muitos programas, ainda hoje, são desenvolvidos em C.
- O C é uma linguagem ESTRUTURADA!

# O C é "case sensitive"!

Isso quer dizer que a linguagem C, diferencia letras maiúsculas e minúsculas, tanto para nome de funções, variáveis e comandos da linguagem; ou seja, ao declararmos as seguintes variáveis:

**int Soma**, **SOMA, SoMa**, **soma;**

**Todas as variáveis acima serão diferentes para o C.**

Isto também se aplica aos comandos de sintaxe do C, como, por exemplo, o "**for**" e "**if**" que se forem escritos em maiúsculas o compilador não poderá interpretá-las corretamente.

# Estrutura do Código

Devido ao fato do C ser uma linguagem estruturada, ou seja, o compilador segue um fluxo linear de compilação, devemos respeitar uma estrutura de código. Vejamos abaixo:

1ª. Declaração das bibliotecas (cabeçalhos, headers).

2ª. Declaração das variáveis globais.

3ª. Declaração das funções extras.

4ª. Função principal do programa.

# Palavras Reservadas

- Toda linguagem de programação possuí “palavras reservadas”.

- Palavras reservadas não podem ser utilizadas a não ser por seus propósitos originais.

# Palavras Reservadas

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

# Exemplo de Código

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

Bibliotecas

```c
int soma(int a, int b) {
    return a + b;
}
```

Funções

```c
int main() {

    int a = 0, b = 0, result = 0;

    printf("Vamos somar 2 valores!\n");

    printf("Digite o primeiro valor da soma: ");
    scanf("%d", &a);
    printf("Digite o segundo valor da soma: ");
          scanf("%d", &b);

    result = soma(a, b);
    printf("O resultado da soma eh: %d", result);

    getch(); // Aguarda uma tecla para finalizar o programa

    return 0;
}
```

Função principal do programa

# Análise do Exemplo

Vamos analisar o código anterior?

1. Encontre as “etapas” da estrutura no código anterior.

1. Simule a saída na tela gerada pelo programa para as 2 entradas abaixo:

   a) Primeiro valor recebe **10** e o segundo valor recebe **3**.

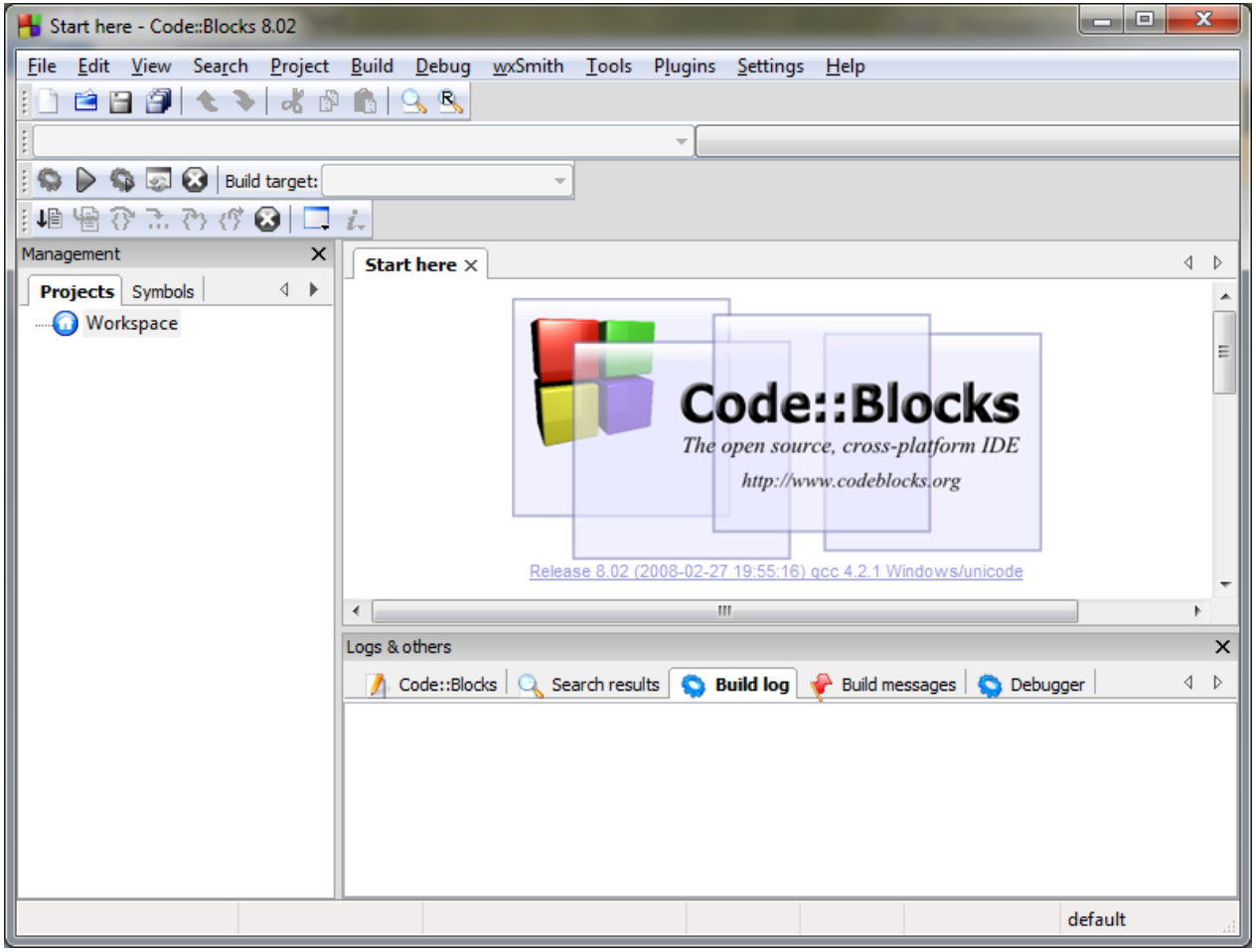
   a) Primeiro valor recebe **2.4** e o segundo valor recebe **1**.

# Codeblocks

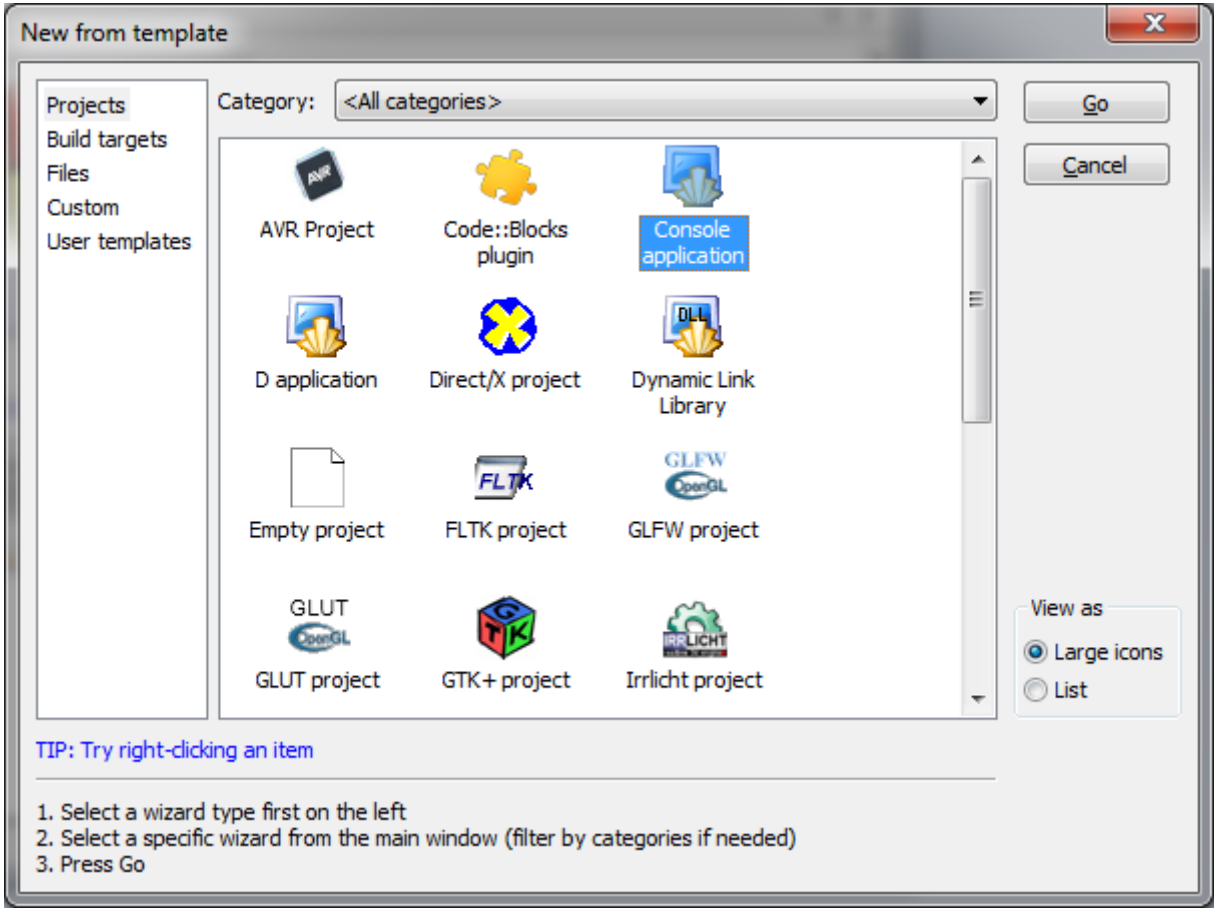
- O que é o codeblocks?
- Para que serve o codeblocks?
- **O Codeblocks NÃO É UM COMPILADOR!**
- Alguns exemplos de IDEs:

  Visual Studio, Dev-Cpp, Eclipse, entre outras...

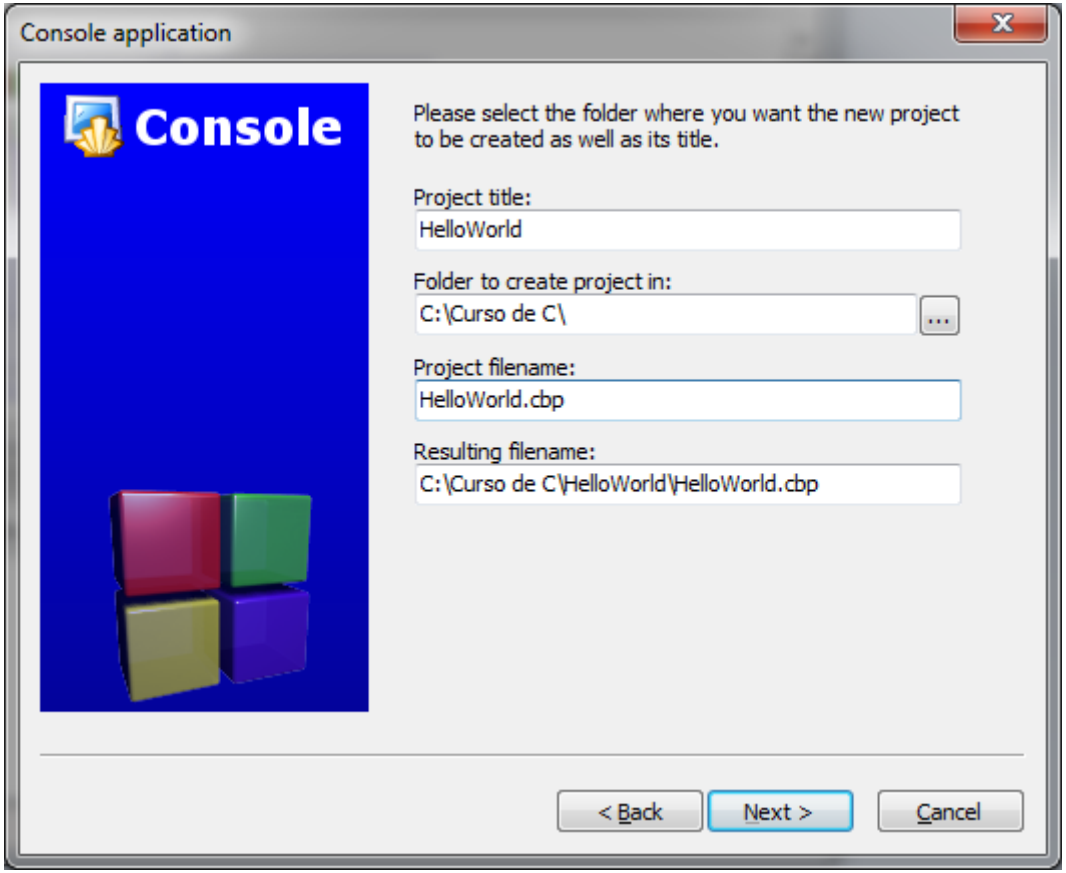
# Interface


Start here - Code::Blocks 8.02
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target:
Management
Projects Symbols
Workspace
Start here
Code::Blocks
The open source, cross-platform IDE
http://www.codeblocks.org
Release 8.02 (2008-02-27 19:55:16) gcc 4.2.1 Windows/unicode
Logs & others
Code::Blocks Search results Build log Build messages Debugger
default

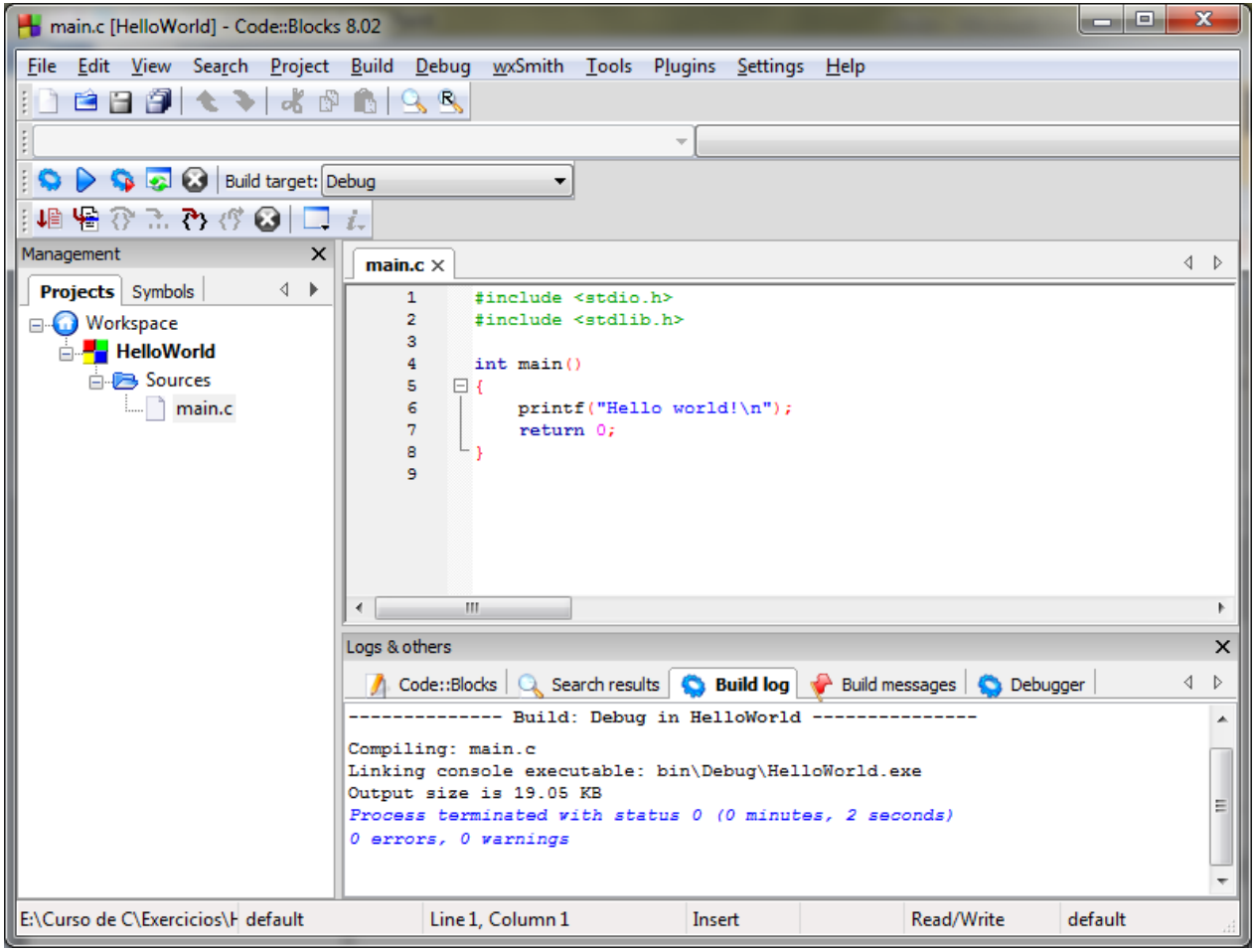
# Criando um projeto



# Criando um projeto



# Compilando o projeto



# Olá mundo

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

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Este será o corpo principal do seu programa!

**Observação**

O Codeblocks criará automaticamente o código acima. Mas não se acostume, viu?

Observe bem o código, você consegue entende-lo?

# Prática

- Vamos criar um primeiro projeto?

- Agora substitua a mensagem “Hello World” pela seguinte mensagem:

**Meu nome eh** <*SEUNOME*> **e eu estou aprendendo C!**

# Antes de programar

- Boas práticas.
- Aprender a pensar.
- Saber trabalhar em equipe.
- Ser curioso.
- Ser persistente.

# Boas práticas

Quando falamos de “boas práticas” podemos fazer referência as “regras de etiqueta”. Assim como na vida, um código de programação em qualquer linguagem necessita de normas organizacionais.


Estas normas, nem sempre sao regras, mas são “dicas” para manter seu código legível, que resultará em um código fácil de entender e manter.

# Os princípios DRY, DIE e KISS

- **DRY** ou **D**on't **R**epeat **Y**ourself (Não se repita)
- **DIE** ou **D**uplication **I**s **E**vil (Duplicação é maligna/má)
- **KISS** ou **K**eep **I**t **S**imple, **S**tupid (Mantenha isto simples, estúpido)

**Informação**

Alguns conceitos ficaram mais claros enquanto avançarmos em nosso estudo. Ao utilizarmos funções perceberemos que não precisamos repetir muitas coisas e inclusive podemos criar bibliotecas padronizadas que pouparão nosso trabalho.

# Comentários

Como já falamos de boas práticas, seria impossível não falar de "comentários em C". Comentários são blocos de texto que são ignorados pelos compiladores.

No C teremos duas formas de comentários, são elas:

```
// Comentário de uma linha

/*
 Bloco de comentário
*/
```

# Identação

Observe o bloco de código abaixo:

```
int main(){int check=1;if(check){printf("Hello
world!\n");}else{printf(“Goodbye world!\n");return 0;}
```

O código acima não está identado. Note como está complicado de ler, apesar de ser um código extremamente simples.

Identar um código nada mais é que separar os códigos em blocos através de tabulação.

# Tipos de Dados

- A linguagem C possuí 5 (cinco) tipos básicos de dados: **char**, **int**, **float**, **void** e **double**.

- Para cada tipo de dado existem modificadores de tipo, estes são 4 (quatro): **signed**, **unsigned, long** e **short**.

- Lembre-se, para o **float** nenhum modificado pode ser aplicado; assim como para o **double** podemos aplicar apenas o **long**.

# Tipos de Dados

| Tipo | Número de bits | Formato de leitura com scanf | Intervalo | |
|---|---|---|---|---|
| | | | Inicio | Fim |
| char | 8 | %c | -128 | 127 |
| unsigned char | 8 | %c | 0 | 255 |
| signed char | 8 | %c | -128 | 127 |
| int | 16 | %i | -32.768 | 32.767 |
| unsigned int | 16 | %u | 0 | 65.535 |
| signed int | 16 | %i | -32.768 | 32.767 |
| short int | 16 | %hi | -32.768 | 32.767 |
| unsigned short int | 16 | %hu | 0 | 65.535 |
| signed short int | 16 | %hi | -32.768 | 32.767 |
| long int | 32 | %li | -2.147.483.648 | 2.147.483.647 |
| signed long int | 32 | %li | -2.147.483.648 | 2.147.483.647 |
| unsigned long int | 32 | %lu | 0 | 4.294.967.295 |
| float | 32 | %f | 3,4E-38 | 3,4E+38 |
| double | 64 | %lf | 1,7E-308 | 1,7E+308 |
| long double | 80 | %Lf | 3,4E-4932 | 3,4E+4932 |

# Tipos de Dados

- Declaração de variável:

```
tipo_da_variavel nome_da_variavel = valor_inicial_da_variavel;
```

- Declaração de variáveis de um mesmo tipo:

```
tipo_da_variavel nome_var1 = valor1, nome_var2 = valor2;
```

**Boas práticas!**
Ao nomear uma variável seja obejtivo, use nomes faceis de entender e se necessário faça um comentáro acima da variável explicando sua utilidade. Em nomes compostos separe-os utilizando *underline*.

# Prática

No seu programa "Hello World", criado anteriormente, vamos fazer algumas modificações:

- Crie duas variáveis do tipo **int** chamadas **num1** e **num2** armazenem consecutivamente os valores **10** e **3**
- Crie uma variável do tipo **float** chamada **result** que armazene o valor **0.0**
- Imprima esses valores da seguinte forma:

```
printf("Os valores sao: %i, %i e %f", num1, num2, result);
```

# Constantes

- São valores fixos mantidos pelo computador.
- As constantes podem ser classificadas em 4 (quatro):
  - Constantes básicas
  - Constantes hexadecimais e octais
  - Constantes de strings
  - Constantes de barra invertida

# Constantes básicas

| Tipo de Dado | Exemplos de Constantes |
|---|---|
| char | ‘b’ ‘\n’ ‘\0’ |
| int | 2 320000 – 130 |
| long int | 100000 -467 |
| short int | 100 -30 |
| unsigned int | 50000 35678 |
| float | 0.0 23.7 -12.3e-10 |
| double | 12546354334.0 -0.0000034236556 |

# Constantes hexadecimais e octais

| Tipo de Dado | Exemplos de Constantes |
|---|---|
| Constante Hexadecimal (8 bits) | 0xEF |
| Constante Hexadecimal (16 bits) | 0x12A4 |
| Constante Octal (12 bits) | 03212 |
| Constante Octal (24 bits) | 034215432 |

Nunca escreva 013 pensando que o C compilará isto como sendo 13.
Em C os valores 013 e 13 são diferentes!

# Constantes de strings

Neste caso cabe apenas uma observação, um alerta, sabemos que **"João"** é uma constante string. Isto implica, por exemplo, no fato de que **'t'** é diferente de **"t"**, pois **'t'** é um **char** enquanto **"t"** é uma string com 2 (dois) **char**s onde o primeiro é **'t'** e o segundo é **'\0'**.

Uma **string** é um conjunto de **char**s com um terminador **'\0'** ao final.

# Constantes de barra invertida

| Código | Significado |
|---|---|
| \b | Retrocesso (“back”) |
| \f | Alimentação de formulário (“form feed”) |
| \n | Nova linha (“new line”) |
| \t | Tabulação horizontal (“tab”) |
| \” | Aspas dupla |
| \’ | Aspas simples |
| \0 | Nulo (0 em decimal) – Terminador |
| \\ | Barra invertida |
| \v | Tabulação vertical |
| \a | Sinal sonoro (“beep”) |
| \N | Constante octal (N é o valor da constante) |
| \xN | Constante hexadecimal (N é o valor da constante) |

# Operadores

- Realizam funções aritméticas e lógicas.
- Possuem, como na matemática, regras de precedência.
- Podem ser classificados em 3 (três) categorias.

# Aritméticos e de Atribuição

| Operador | Ação |
|---|---|
| + | Soma (inteiro e ponto flutuante) |
| - | Subtração ou troca de sinal (inteiro e ponto flutuante) |
| * | Multiplicação (inteiro e ponto flutuante) |
| / | Divisão (inteiro e ponto flutuante) |
| % | Resto da divisão (inteiros) |
| ++ | Incremento (inteiros e ponto flutuante) |
| -- | Decremento (inteiro e ponto flutuante) |

# Exemplo

```
int a = 17, b = 3;
int x, y;
float z = 17., z1, z2;
x = a / b;
y = a % b;
z1 = z / b;
z2 = a / b;
a++;
b--;
```

A execução deste bloco de código, resultaria em:

| | |
|---|---|
| x = 5 | y = 2 |
| z1 = 5.666666 | z2 = 5.0 |
| a = 18 | b = 2 |

# Prática

Vamos fazer algumas operações com os nossos valores?

- Some o valor de **num1** e **num2**, armazenando o resultado em **result** e em seguida imprima o resultado;
- Divida o valor de **num1** por **num2**, armazenando o resultado em **result** e em seguida imprima o resultado;
- Multiplique o valor de **num1** por **num2**, armazenando o resultado em **result** e em seguida imprima o resultado.

# Expressões

- São combinações de variáveis, constantes e operadores.

- Devemos levar em consideração a tabela de precedência ao montá-las.

**Exemplos de expressões:**

```
Anos = Dias / 365.25;
i = i + 3;
c = a * b + b / e;
c = a * (b + d) / e;
```

# Expressões

**Importante! Conversões de tipos de expressão.**

- Quando executamos expressões em tipos de dados diferentes, o compilador verifica se as conversões são possíveis, se não forem possíveis ele mostrará um erro. Se as conversões forem possíveis ele as fará, segundo as seguintes regras:

1. Todos os **char**s e **short**s **int**s são convertidos para **int**s. Todos os **float**s são convertidos para **double**s.

2. Para pares de operandos de tipos diferentes: se um deles é **long double** o outro é convertido para **long double**; se um deles é **double** o outro é convertido para **double**; se um é **long** o outro é convertido para **long**; se um é **unsigned** o outro é convertido para **unsigned**.

# Expressões

| Expressão Original | Expressão Equivalente |
|---|---|
| x = x + k; | x += k; |
| x = x – k; | x -= k; |
| x = x * k; | x *= k; |
| x = x / k; | x /= k; |

Existem várias outras abreviações. Seja curioso e pesquise!

# Racionais e Lógicos

| Operador | Ação |
|---|---|
| > | Maior do que |
| >= | Maior ou igual a |
| < | Menor do que |
| <= | Menor ou igual a |
| == | Igual a |
| != | Diferente de |
| && | AND ( E ) |
| \|\| | OR ( OU ) |
| ! | NOT ( NÃO ) |

O C também possuí uma classe de operadores lógicos chamados de "Operadores bit-a-bit", que permitem que você trabalhe diretamente na representação binária de um valor.

# Tabela Verdade

| p | q | p && q | p \|\| q |
|---|---|---|---|
| verdadeiro | verdadeiro | verdadeiro | verdadeiro |
| verdadeiro | falso | falso | verdadeiro |
| falso | verdadeiro | falso | verdadeiro |
| falso | falso | falso | falso |

# Tabela de Precedência

| Maior precedência |
| --- |
| () [] -> |
| ! ~ ++ -- . -(unário) |
| (cast) *(unário) |
| &(unário) sizeof |
| * / % |
| + - |
| << >> |
| <<= >>= |
| == != |
| & |
| ^ |
| \| |
| && |
| \|\| |
| ? |
| = += -= *= /= |
| , |
| Menor precedência |

# Introdução a Entrada e Saída de dados

- Sempre que solicitarmos alguma informação ao usuário, teremos uma entrada de dados.

- Sempre que exibirmos algo ao usuário, seja uma informação processada ou não, teremos uma saída.

- Existem várias formas de entrada e saída de dados no C, estudaremos as mais comuns.

# Entrada

Sempre que falamos de entradas de dados, devemos considerar que essas entradas podem ocorrer de diversas formas, as mais comuns:

- Dados via teclado
- Dados recebidos através de scanners (leitores de código de barra)

Trabalharemos aqui apenas com dados recebidos através do teclado, para isso precisamos conhecer as funções básicas de entrada que o C nos fornece.

# *getch()*

- É parte da biblioteca **conio.h**
- Utilizado para receber um único caractere
- Esta é uma função exclusiva para Windows

**Formato:**

```
variável_de_recebimento = getch();
```

# *getch()*

## Exemplo:

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

int main() {

   char Ch;
   Ch = getch();

   printf("Voce pressionou a tecla: %c", Ch);

   return 0;
}
```

# *scanf()*

- É parte da **stdio.h**
- Utilizado para receber **string**s.
- É multiplataforma

**Formato:**

```
scanf(string_de_controle, lista_de_argumentos);
```

# *scanf()*

**Exemplo:**

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

int main() {

   char Ch;
   scanf("%c", &Ch);

   printf("Voce pressionou a tecla: %c", Ch);

   return 0;
}
```

# Saída

Quando falamos de saída de dados, devemos considerar que as saídas de dados podem ocorrer de várias formas, as mais comuns são:

- Através do monitor
- Através da impressora

Trabalharemos aqui apenas com a exibição de mensagens na tela.

# *printf()*

- É parte da **stdio.h**
- Utilizado pare imprimir na tela uma mensagem

**Formato:**

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

# *printf()*

## Exemplo:

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

int main() {

   int nota = 10;

   printf("O aluno tirou nota %d!", nota);

   return 0;
}
```

# Estruturas de Controle de Fluxo

- São responsáveis por controlar o fluxo do programa.
- Testam condições.
- Algumas são conhecidas como “loops”.

**Boas práticas!**
Lembre-se de *identar* seu código, isto facilita a leitura do mesmo, principalmente em estruturas de repetição.

# if-else

A estrutura **if-else** é utilizada para tomada de decisões, quando uma condição é válida ou não.

**Formato:**

```
if ( condicao ) {
   bloco_de_comando
}  else {
   bloco_de_comando
}
```

# if-else

**Exemplo:**

```
int a = 1, b = 2;

if ( (a + b) == 3 ) {
   printf("O resultado eh 3");
}  else {
   printf("O resultado nao eh 3");
}
```

# switch

O **switch** também é utilizado para tomada de decisões, porém cria um código mais limpo. Com ele você pode testar uma variável em relação a diversos valores pré-estabelecidos.

## Formato:

```
switch ( variável ) {
   case constante_1:
        bloco_de_comando
        break;
   default:
        bloco_de_comando
        break;
}
```

# switch

**Exemplo:**

```
scanf("%d", &num);
switch ( num ) {
   case 1:
        printf("Voce digitou 1");
        break;
   case 2:
        printf("Voce digitou 2");
        break;
   default:
        printf("Voce digitou %d", num);
        break;
}
```

# while

O **while** é uma estrutura de repetição, utilizada para criar os chamados "*loops*" *de um programa.* O código dentro do bloco repetirá enquanto a condição não for verdadeira.

**Formato:**

```
while ( condição ) {
   bloco_de_comando
}
```

# while

**Exemplo:**

```
while ( 1 == 1 ) {
   printf("Meu querido loop infinito\n");
}
```

# for

Assim como o **while** o **for** é utilizado para criar estruturas de repetição.

**Formato:**

```
for ( inicializacao; condição; incremento ) {
   bloco_de_comando
}
```

# for

**Exemplo:**

```
int count;

for (count = 1; count <= 10; count++ ) {
   printf("%d ", count);
}
```

# Strings e Matrizes

- São provavelmente os elementos mais importantes de um programa.

- Derivam de tipos básicos de variáveis.

# Strings

- São vetores de **char**s.
- Seu último elemento é o **'\0'**.

**Formato:**

```
char nome_da_string[tamanho]
```

# Strings

## Exemplo:

```
char nome[100];

printf("Digite o nome do aluno: ");
gets(nome);
```

**Dicas!**
Na linguagem C, **string**s sempre estão dentro de "aspas duplas" e podem possuir mais de um caractere, enquanto **char**s estão sempre entre aspas simples e contém apenas um elemento. Lembre-se bem disto!

# *Manipulando Strings*

Quando trabalhos com strings, sentimos a necessidade de manipulá-las. Para isto o C possuí alguns comandos específicos, são estes:

- strcpy
- strcat
- strlen
- strcmp

# strcpy()

- Utilizada para copiar um string-origem para um string-destino.

- Faz parte da biblioteca **string.h**

**Formato:**

```
strcpy(string_destino, string_origem);
```

# strcpy()

**Exemplo:**

```
char str1[100], str2[100], str3[100];

printf("Digite uma string: ");
gets(str1);

strcpy(str2, str1);
strcpy(str3, "Voce digitou: ");

printf("%s%s.\n", str3, str2);
```

# strcat()

- Anexa a string-origem ao fim da string-destino, sem alterar a string-origem.

- Faz parte da biblioteca **string.h**

**Formato:**

```
strcat(string_destino, string_origem);
```

# strcat()

**Exemplo:**

```
char str1[100], str2[100];

printf("Digite uma string: ");
gets(str1);

strcat(str2, "Voce digitou: ");
strcat(str2, str1);

printf("%s.\n", str2);
```

# strlen()

- Retorna o comprimento da **string**.
- O terminador não é contado.

**Formato:**

```
strlen(string);
```

# strlen()

**Exemplo:**

```
int size;
char nome[100];

printf("Digite o seu nome: ");
gets(nome);

size = strlen(nome);

printf("%i.\n", size);
```

# strcmp()

- Utilizado para comparar uma **string**.
- Retorna 0 se as **string**s forem iguais.

**Formato:**

```
strcmp(string1, string2);
```

# strcmp()

**Exemplo:**

```
char str1[100], str2[100];

printf("Digite a primeira string: ");
gets(str1);
printf("Digite a segunda string: ");
gets(str2);

if( strcmp(str1, str2) ) {
   printf("\nAs strings sao diferentes.");
} else {
   printf("\nAs strings sao iguais!");
}
```

# Matrizes

- Também conhecidos como vetores.
- Podem possuir várias “dimensões”.
- São estruturas de dados muito utilizadas.
- O primeiro “index” de uma matriz é sempre 0.

**Formato:**

```
tipo_de_dado nome_da_string[tamanho]
```

# Matrizes

**Exemplo:**

```
int num[10];
int i = 0;

printf("Digite 10 numeros: ");

for(i = 0; i <= 10; i++ ) {
   scanf("%i", &num[i]);
}

for(i = 0; i <= 10; i++) {
   printf("%i \n", num[i]);
}
```

# *Matrizes multidimencionais*

- São matrizes que possuem mais de uma "dimensão".

**Formato:**

```
tipo_de_dado nome_da_var[tamanho_1][tamanho_2]...[tamanho_N];
```

# Matrizes multidimensionais

**Exemplo:**

```
int mtrx[20][10];
int count = 1, i = 0, j = 0;

for( i = 0; i < 20; i++ ) {
   for( j = 0; j < 10; j++ ) {

        mtrx[i][j] = count;
        printf("mtrx[%i][%i] = %i \n", i, j, count);
        count++;
   }
}
printf("O valor de mtrx[3][5] = %i", mtrx[3][5]);
```

# *Matrizes de Strings*

- São matrizes bidimensionais.

- Podem ser chamadas de "lista de **string**s indexadas".

**Formato:**

```
char nome_da_variavel[num_de_strings][comprimento_da_string];
```

# Matrizes de Strings

**Exemplo:**

```
char frutas[3][10] = { "Pera", "Uva", "Laranja" };

printf("Voce gosta de comer %s.\n", frutas[2]);
```

Qual a mensagem de saída do exemplo acima?

```
Voce gosta de comer Laranja.
```

# *Matrizes dinâmicas*

- São matrizes que não possuem um tamanho específico.

**Exemplo:**

```
int mess[] = { "Linguagem C: Flexibilidade e Poder!" };
int mtrx[][2] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

**Dicas!**
Tome cuidado com as matrizes dinâmicas, são extremamente úteis, porém seu consumo de memória é grande e pode causar alguns erros graves caso não utilizadas corretamente.

# Funções

- São “comandos” da linguagem C, nativos ou não.
- Podem ser criados pelo usuário.
- Executam uma tarefa especifica.
- São utilizadas para separar o código e evitar repetições.

**Boas práticas!**
Fala-se no mundo da programação que se uma função não retornar algo ao seu final, ela não tem motivo de existir. Isto é uma “**FALSA VERDADE”**, ou seja, algo não aplicável.

# Estrutura de uma função

- Uma função pode ou não retornar um valor no seu final.
- Uma função pode receber ou não parâmetros.

**Formato:**

```
tipo_de_retorno nome_da_funcao(parametros) {
   corpo_da_funcao
}
```

# *Tipo e Retorno*

- Especifica o tipo de dados que a função retornará.
- Para retornar um valor utilizando **return** ao final da função seguido da valor a ser retornado.
- Os tipos mais utilizados são:
  - void
  - int
  - char

# Parâmetros

- São os valores recebidos pela funcão.
- Uma função pode não receber nenhum parâmetro.
- Uma função pode receber vários parâmetros.

**Boas práticas!**
Apesar de uma função ser capaz de receber inúmeros parâmetros, a "etiqueta" diz que para uma função ser facilmente compreendida deverá ter ao máximo 3 (três) parâmetros.

# Exemplo

```
int soma(int a, int b) {
   return a + b;
}
```

Esta função realizará a soma de dois parâmetros do tipo inteiro, e retornará o resultado no mesmo formato.

**Ideia:**

```
int resultado = soma(10, 3);
```

O valor de ***resultado*** será *13.*

# Prática

No exemplo anterior acompanhamos e entendemos como implementar uma função de soma ao nosso programa, agora vamos criar as seguintes funções:

- Subtração
- Multiplicação
- Divisão

E vamos aplicá-las em nosso programa, solicitando ao usuário a entrada de dois valores e a operação a ser realizada.

# Visibilidade de variáveis

Quando usamos funções, sentimos a necessidade de que, em alguns casos, as variáveis sejam "visualizadas" por todas as funções, tal como também teremos algumas variáveis que serão particularmente propriedades de uma função.

Chamamos esta diferença entre as variáveis de *níveis de visualização*, estes podem ser em C, desta forma classificamos as variáveis em 2 (dois) tipos:

- Variáveis Locais
- Variáveis Globais

# *Variáveis locais*

- É declarada dentro de uma função.
- Tem seu valor apenas na função onde foi declarada.
- Nenhuma outra função pode altera-la.

**Exemplo:**

```
int main() {

   char nome[100];
   return 0;
}
```

# *Variáveis globais*

- É declarada no corpo do programa. Fora de funções.
- Pode ser alterada por qualquer função do programa.

**Exemplo:**

```
int valor;

int main() {
   valor = 100;
   printf("O valor eh: %d", valor);
   return 0;
}
```

# Bibliotecas

- São inseridas através do comando **#include <>;**
- São assim chamadas por conterem funções úteis ao programador.
- Você pode criar suas próprias bibliotecas para simplificar seu trabalho.
- Também são chamadas de **headers**, **libs**, **cabeçalhos** ou **arquivos-cabeçalho**.
- Sempre terminam com a extensão **.h**

# Prática

Vamos criar nossa primeira biblioteca, ela deverá ser capaz de realizar as seguintes funções:

- Retornar o valor da soma de dois parâmetros
- Retornar o valor da subtração de dois parâmetros
- Retornar o valor da multiplicação de dois parâmetros
- Retornar o valor da divisão de dois parâmetros

# Conclusão

Muito bem, agora já sabemos conhecemos um pouco sobre a linguagem C, que tal colocarmos tudo que aprendemos em prática?

Afinal, apenas a prática nos levará a perfeição; E se surgir alguma dúvida, lembre que o Google é seu amigo.

Obrigado a todos pela paciência!

Ricardo Lüders
e-mail: xangelbr@gmail.com
Site: http://www.luders.com.br/