

Aula 01 - Introdução à Linguagem C

Programação: Estrut. Sequencial

OPL e IB

Prof: Anilton Joaquim da Silva

anilton.ufu@outlook.com

A linguagem C

- A linguagem C foi desenvolvida no fim da década de 60;
- C++ começou na década de 70 e é uma extensão do C com diversas funcionalidades com orientação a objetos;
- A linguagem C++ é um super conjunto da linguagem C, ou seja, todo e qualquer programa em C também é um programa em C++, mesmo que o oposto não seja verdade.

Primeiro Programa

- O Algoritmo em linguagem C, abaixo, descreve para o computador os passos necessários para se escrever a mensagem “Olá Mundo!” na tela do computador.
- - `#include <stdio.h>`
 - `#include <stdlib.h>`
 - `int main()`
 - `{`
 - `printf ("Hello world!\n ");`
 - `return 0;`
 - `}`

Área de um retângulo

- A área de um retângulo pode ser facilmente calculada caso você saiba o comprimento de sua base e de sua altura. Matematicamente, seja b o comprimento da base e a o comprimento da altura. A função f equivalente à área do retângulo pode ser definida como: $f(a; b) = a * b$. Isto é, a função f tem dois parâmetros (a - altura e b - base, do retângulo) e calcula a área como sendo a multiplicação de a e b .

```
int f(int a, int b)
2 {
    return a * b;
4 }
```

Tipos primitivos da Linguagem C

- O código acima tem a limitação de só calcular a área de retângulos cujos lados tenham tamanhos inteiros.
- Para corrigir esta deficiência, vamos alterá-lo para que aceite números reais. Em computação, números reais são também chamados de números com pontos flutuantes e, em linguagem C, simplesmente de float.
- Podemos corrigir o programa simplesmente substituindo as ocorrências da palavra int por float.

```
float f(float a, float b)
2 {
    return a * b;
4 }
```

Organização do Código

- É possível perceber um padrão nos exemplos:
 - A linha definindo a função é seguida por uma linha contendo apenas um caracter { que é alinhado com o início da linha acima.
 - A última linha da função contém apenas um caracter }, alinhado com o { do início da função.
 - Todas as linhas entre o { inicial e o } final estão com alinhamento mais avançadas em relação às chaves.
 - Os caracteres { e } representam a especificação de um bloco de código (incio e fim)

Comentários

- Algo que faltou nestes exemplos e que também serve ao propósito de facilitar o entendimento do código são os chamados comentários.

- `/*` → comentários por blocos

```
.....  
.....*/
```

- `//` → comentários de linha

```
1 /*  
2  * A funcao a seguir calcula a area de um retangulo de base  
3  * base e altura altura. Os parametros e resultado da funcao  
4  * sao do tipo float.  
5  */  
6 float area_retangulo(float altura, float base)  
7 {  
8     //Calcula e retorna a area do retangulo.  
9     return altura * base;  
10 }
```

Saída de dados padrão

- Um programa está executando a saída de dados quando envia para “fora” do programa tais dados. Exemplos comuns de saída de dados são a escrita em arquivo, o envio de mensagens na rede ou, impressão ou, mais comum, a exibição de dados na tela (stdout).
- Para enviar dados para a saída do C, usamos a expressão **printf** , seguido do dado a ser impresso na tela.
- Forma geral: **printf("expressão de controle", argumentos);**
- **expressão de controle** : \n – nova linha, %c – char, %d – int, %f – float, e outros.
- **argumentos**: constantes, variáveis, expressões, e função.

Saída de dados

- exemplos.
 - `printf ("numero");`
 - `printf(" %d " , 10);`
- Imprime na tela: numero 10
 - `printf (" numero %d " , 10);`
- Imprime na tela: numero 10
 - `printf (" numero %d \n texto " , 10);`
- Imprime na tela: numero 10
 texto
 - `printf (" sen(1) \n %d " , sin(1));`
- Imprime na tela: sen(1)
 0

A função main()

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

float area_retangulo(float altura, float base)
{
    //Calcula e retorna a area do retangulo.
    return altura * base;
}

int main()
{
    float area;
    area = area_retangulo(2.0, 3.5);
    printf ( “ \n area do retangulo: % ” , area);

    return 0;
}
```

A função main()

- Algumas observações importantes sobre a função main:
 - A função main tem sempre um resultado do tipo inteiro e seu resultado é sempre o (return 0;);
 - Só pode haver uma Função main para cada programa;
 - Não se pode ter nomes repetidos de funções, para cada programa;
 - Finalmente, a função area_retangulo aparece antes da função main no programa. Isto deve ser verdade para todas as funções do seu programa. Isto ocorre por quê, antes de executar a função main, o computador precisa aprender sobre a existência das outras funções.

Compilação e Execução

- Para colocarmos nossos algoritmos em execução, o primeiro passo é escrevê-los, usando um editor de textos qualquer que salve arquivos em texto puro, como o notepad, vim, gedit, etc. A este arquivo com o código chamaremos código fonte ou simplesmente fonte (extensão .c).
- A sequência de passos que compõem a compilação é a seguinte:
 - **Código Fonte → Pré-processador → Fonte Expandido → Compilador → Arquivo Objeto → Ligador → Executável**
- A compilação traduz o código que você escreveu para uma linguagem inteligível ao computador, salvando-o em um arquivo chamado arquivo objeto. Por exemplo, a compilação transformaria o código “Olá Mundo!” escrito acima em algo como:

...

```
CALL write(0x1,0x400623,0xe)
```

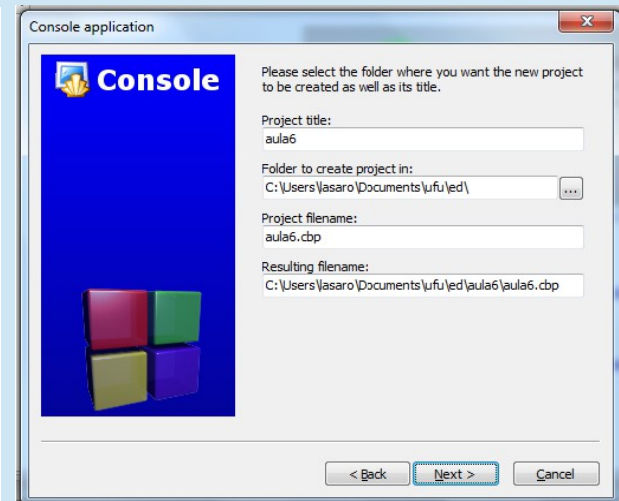
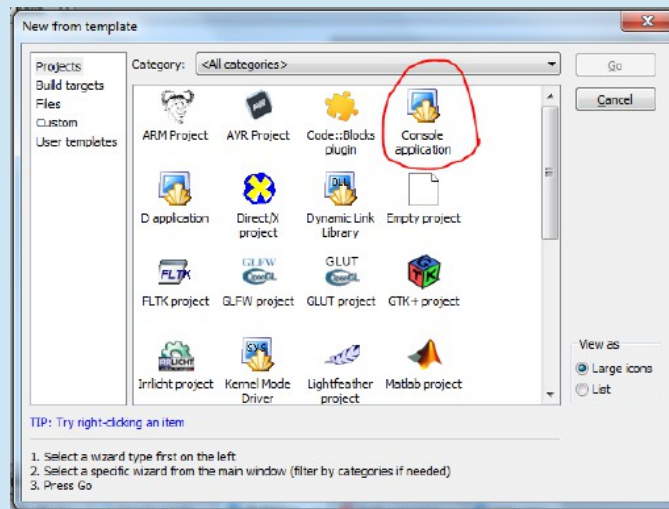
```
GIO fd 1 "Olá Mundo!"
```

```
RET
```

OBS: para um primeiro programa:
primeiroProg.c
primeiroProg.obj
primeiroProg.exe

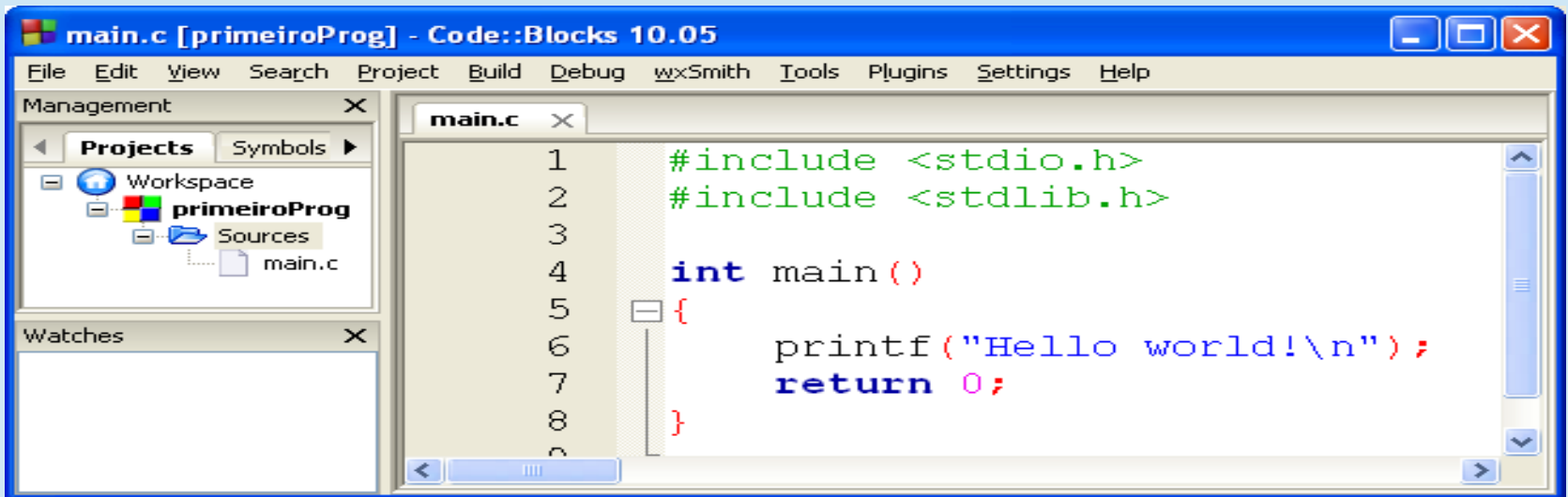
A IDE Code::Blocks

- Criando um Projeto: clique em **File** e, em seguida, **New, Project**;
- Escolha **Console Application** e então clique em **Go**;
- Escolha **C** e clique em **Next**;
- Em **Project title** escreva algo como **primeiroProg**; em **Folder to create the project in**, clique no botão com . . . e escolha uma pasta para salvar o projeto. Pode ser a pasta Meus Documentos ou uma pasta qualquer em um pen drive. Clique então **Next** e, na tela seguinte, clique em **Finish**.




A IDE Code::Blocks

- Seu projeto foi criado. Agora abra o arquivo main.c, que está na pasta sources, dando um clique duplo no nome do arquivo. Observe que o Code::Blocks criou automaticamente um programa básico.



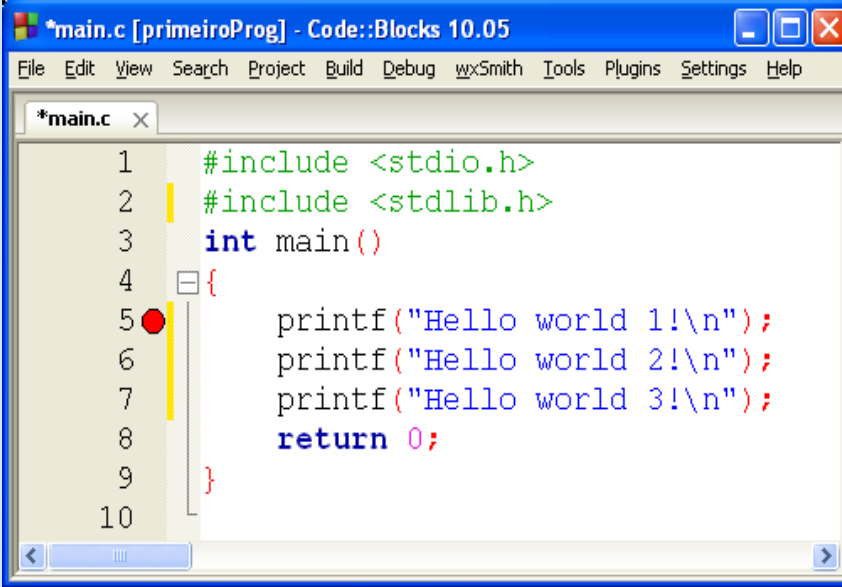
The screenshot shows the Code::Blocks IDE interface. The title bar reads "main.c [primeiroProg] - Code::Blocks 10.05". The menu bar includes File, Edit, View, Search, Project, Build, Debug, wxSmith, Tools, Plugins, Settings, and Help. On the left, the "Management" panel shows a tree view with "Workspace", "primeiroProg", "Sources", and "main.c". The main editor window displays the following C code:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      printf("Hello world!\n");
7      return 0;
8  }
```

- Clique em em build and run. 
- Parabéns, você acaba de executar seu primeiro programa.



Depuração

- Todo programa é comum encontrar erros (bugs) de codificação e de lógica. Uma das formas de achar os bugs do seu programa é fazer com que o computador execute seu programa passo a passo, isto é, linha a linha, e acompanhar esta execução verificando se o programa faz o que você espera.



```
*main.c [primeiroProg] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help

*main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main()
4  {
5  printf("Hello world 1!\n");
6  printf("Hello world 2!\n");
7  printf("Hello world 3!\n");
8  return 0;
9  }
10
```

- Para depurar, clique ao lado direito do número 5 (quinta linha do programa), até que uma bolinha vermelha apareça, como na figura. A bolinha vermelha é, na verdade, um sinal de pare, e diz ao computador que deve, ao executar seu programa, parar ali.
- Clique no menu **Debug** e então em **Start** ou, alternativamente, pressione a tecla **F8** ().
- Observe que a execução parou onde você esperava.
- Agora, clique em **Debug** e **Next Line** ou aperte **F7** () , no teclado, sucessivamente para ver o que acontece. Observe que cada linha é executada passo a passo.

Declaração de Variáveis

- Na linguagem C, toda variável deve ser declarada (isto é, criada) no início do corpo da função que a contem. A declaração de uma variável tem pelo menos duas partes:
 - Tipo: tipo de dado, ou seja, se é um número, ou uma palavra, ou uma caractere, etc;
 - Nome: usado para referenciar a variável quando se precisa ler, usar em uma equação ou escrever a mesma;
- Algumas regras simples devem ser seguidas na hora de se nomear uma variável:
 - o nome só pode conter os caracteres [a-z], [A-Z], [0-9] e o “_”;
 - o nome não pode começar com números.

Declaração de Variáveis

- Tipos básicos:
 - **int** - representando um número inteiro (16 bits – 2^{15}). ex. 3, 4, -5;
 - **float** - representando um número real, com casas decimais separadas por ponto “.” (32 bits – $3.4E^{38}$). ex. 3.1416, 0.75, -1.2;
 - **char** - representando um caractere (1 byte): letra, dígito, ... identificado por apóstrofes. Exemplo ‘5’, ‘a’, ‘Z’, ‘.’, ‘e’, ‘-’, ‘#’.
- Modificadores:
 - **unsigned** (unsigned int (16 bits: 0 – 2^{16}))
 - **long** (long int (32 bits: -2^{31} a $(2^{31}) - 1$))
- Exemplo:

São exemplos de declarações de variáveis válidas:

```
1 int nota1, nota2;  
2 float media;  
3 char _caractere;
```

São exemplos de declarações inválidas:

```
1 int 1nota, 2nota;  
2 float #media;  
3 char nome completo;
```

Atribuição e uso de variáveis

<pre>1 int inteiro1, inteiro2; float real; 3 inteiro1 = 0; 5 inteiro2 = 10; real = 10.0;</pre>	<pre>int inteiro1 = 0, inteiro2 = 10; float real = 10.0;</pre>
--	--

Parâmetros são variáveis:

```
float area_retangulo(float altura, float base)
{
    //Calcula e retorna a area do retangulo.
    return altura * base;
}
int main()
{
    float area;
    area = area_retangulo(2.0, 3.5);
    printf ( " \narea do retangulo: %f " , area);

    return 0;
}
```

Entrada de dados padrão

- De forma semelhante ao printf, há um comando para leitura denominado scanf.
- Forma geral: `scanf("expressão de controle", argumentos);`
- **expressão de controle:** %c – char, %d – int, %f – float, ...
- **argumentos:** lista de variáveis (endereços: &variável)
- Este comando permite ler valores digitados (teclado = stdin) pelo usuário atribuindo as variáveis argumentos.

Entrada e Saída de dados

- Exemplos
- `char letra;`
- `int idade;`
- `float altura;`
- `printf ("Informe a letra inicial de seu nome e sua idade, e altura: ");`
- `// a seguir eh feita a leitura`
- `scanf (" %c %d %f " , &letra, &idade, &altura);`
- `printf (" \n A letra eh %c " , letra);`
- `printf (" , sua idade eh %d, e sua altura eh %f \n " , idade, altura);`

Formatação de impressão

- Em algumas ocasiões há necessidade de formatar a saída para, por exemplo, garantir que os dados fiquem alinhados, imprimir uma tabela, ou simplesmente por estética.

```
int main() {  
    printf( " \n%2d ", 350);  
    printf( " \n%4d ", 350);  
    printf( " \n%04d ", 21);  
    printf( " \n%06d ", 21);  
    printf( " \n%6.4d ", 21);  
    printf( " \n%6.0d ", 21);  
    return 0;  
}
```

Formatação de impressão

- Para formatação de números reais (float e double), o exemplo a seguir mostra alguns comandos para formatação:

```
int main() {  
    printf("\n%4.2f ", 3456.78);  
  
    printf("\n%3.1f ", 3456.78);  
  
    printf ("\n%10.3f ", 3456.78);  
  
    printf("\n%10.2f %10.2f %10.2f ", 8.0, 15.3, 584.13);  
    printf("\n%10.2f %10.2f %10.2f ", 834.0, 1500.55, 4890.21);  
    printf("\n %-10.2f  %-10.2f  %-10.2f ", 8.0, 15.3, 584.13);  
    printf("\n %-10.2f %-10.2f %-10.2f ", 834.0, 1500.55, 4890.21);  
  
    return 0;  
  
}
```

Operadores

- Aritméticos:

- = (atribuição), + (soma), - (subtração), * (multiplicação), / (divisão) e % (resto da divisão de int)

OBS: $a += b \rightarrow a = a + b$; $x *= y \rightarrow x = x * y$; $i++; \rightarrow i = i + 1$; $i--; \rightarrow i = i - 1$;

- Relacionais:

- == (teste de igualdade), != (diferente), > (maior que), < (menor que), >= (maior ou igual) e <= (menor ou igual)

- Lógicos:

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

- Funções

- abs(X): obtém o valor absoluto de X;
- sqrt(X): calcula a raiz quadrada de X; pow(X, Y): calcula potencia - X elevado a Y (X^Y)
- log(X): calcula o logaritmo de X;
- mod(X, Y): obtém o resto da divisão de X por Y;
- trunca(X): obtém a parte inteira de X;
- round(X): arredonda o valor de X;
- sin(X): calcula o valor do seno de X;
- cos(X): calcula o valor do cosseno de X; tan(X): calcula o valor da tangente de X

Escopo de Variáveis

```
float area_retangulo(float altura, float base)
{
    //Calcula e retorna a area do retangulo.
    return altura * base;
}
int main()
{
    float area,
        a,
        b;
    printf("Qual a altura do retangulo? ");
    scanf("%f", &a);
    printf("Qual a base do retangulo? ");
    scanf("%f", &b);
    area = area_retangulo(a, b);
    printf ( " \n area do retangulo: %f" , area);
    return o;
}
```

Posso chamar as variáveis
float a, b; de **float altura, base;**?

Esta mudança afeta alguma coisa na
função **area_retangulo**?

Estas mudanças não afetaram a execução do programa. Isto acontece por que as variáveis tem **escopos bem definidos** em C. A **variável altura** da função **main** não é a mesma **variável/parâmetro altura** da função **area_retangulo**; cada uma só existe dentro do corpo da função em que foi declarada. Quando a função **area_retangulo** é invocada passando-se como parâmetro a variável **altura** da função **main**, o **valor desta variável é copiado para o parâmetro altura** da função invocada.

OBS: definição de constantes

```
#define PI 3.141559
```

Deve ser escrito no começo do programa depois dos includes e antes do código.

Lendo e imprimindo Strings

- Declarando uma string:
`char nome[30];` → vetor de caracteres
- Lendo uma string:
`scanf (“ %s” , nome);` → lê somente uma única palavra
`gets (nome);` → lê uma frase com até 30 caracteres
- Imprimindo uma string:
`printf (“ %s “ , nome);`
ou
`puts (nome);`
→ imprime nome, palavra ou frase