Aula 3: Construindo um Programa em C

Adriano Veloso

Algoritmos e Estruturas de Dados I - DCC/UFMG

Estilo e Leitura

Agora que vocês já conhece a linguagem C, vamos introduzir conceitos tais como: estilo, leitura e usabilidade. Vamos ver hoje:

- Os elementos que contribuem para o estilo de um programa em C
- Formas de se incluir código pré-escrito em um programa
- Definição de constantes a serem usadas no programa

Adotando um estilo

Devemos adotar um estilo de escrita quando programamos?

Adotando um estilo

Devemos adotar um estilo de escrita quando programamos?

SIM! O estilo escolhido vai depender de uma série de fatores:

- Se você for um estudante: o estilo vai facilitar que seu professor entenda seu código

Adotando um estilo

Devemos adotar um estilo de escrita quando programamos?

SIM! O estilo escolhido vai depender de uma série de fatores:

- Se você for um estudante: o estilo vai facilitar que seu professor entenda seu código
- Se você já trabalha: você terá que adotar o estilo imposto pela empresa

Adotando um estilo

Devemos adotar um estilo de escrita quando programamos?
SIM! O estilo escolhido vai depender de uma série de fatores:

- Se você for um estudante: o estilo vai facilitar que seu professor entenda seu código
- Se você já trabalha: você terá que adotar o estilo imposto pela empresa

A razão para adotarmos um estilo de programação é a facilidade de leitura

- Serão vários programas e vários programadores
- Se todos usarem o mesmo estilo, todos irão ler e entender melhor os programas

O estilo universal

Programa sem estilo algum:

```
void main()      {int a; a=20; printf("Imprimindo o valor de a:
%d", a);}
```

O estilo universal

Programa sem estilo algum:

```
void main()      {int a; a=20; printf("Imprimindo o valor de a:
%d", a);}
```

Estilo universal:

```
void main() {
        int a;
        a=20;
        printf("Imprimindo o valor de a: %d", a);
}
```

O estilo universal

Programa sem estilo algum:

```
void main()      {int a; a=20; printf("Imprimindo o valor de a:
%d", a);}
```

Estilo universal:

```
void main() {
        int a;
        a=20;
        printf("Imprimindo o valor de a: %d", a);
}
```

Todo bloco de comandos devem ser indentados

Documentação

- Um programa bem feito não é apenas um conjunto de instruções

Documentação

- Um programa bem feito não é apenas um conjunto de instruções
- A documentação envolve todo texto que venha a descrever o que o programa faz

Documentação

- Um programa bem feito não é apenas um conjunto de instruções
- A documentação envolve todo texto que venha a descrever o que o programa faz
- A documentação não deve ser feita após o programa estar pronto, mas sim à medida em que ele vai sendo desenvolvido

Documentação

- Um programa bem feito não é apenas um conjunto de instruções
- A documentação envolve todo texto que venha a descrever o que o programa faz
- A documentação não deve ser feita após o programa estar pronto, mas sim à medida em que ele vai sendo desenvolvido
- A documentação também deve conter o modo de uso do programa, bem como as simplificações assumidas pelo programador

Comentários

- Texto embutido no código-fonte, que ajuda no entendimento do programa

Comentários

- Texto embutido no código-fonte, que ajuda no entendimento do programa
- Regra geral: Sempre comentar trechos do programa que não são óbvias. Nunca comentar trechos do programa que sejam óbvios.

Comentários

- Texto embutido no código-fonte, que ajuda no entendimento do programa
- Regra geral: Sempre comentar trechos do programa que não são óbvias. Nunca comentar trechos do programa que sejam óbvios.
- Duas formas de inserir comentários:
  1. //linha comentada
  2. /*trecho comentado*/

Funções

Uma função é um conjunto de instruções que realizam uma operação

- Geralmente uma operação bem mais complicada do que aquela que pode ser realizada por uma única instrução

Funções

Uma função é um conjunto de instruções que realizam uma operação

- Geralmente uma operação bem mais complicada do que aquela que pode ser realizada por uma única instrução
- Uma função pode ser pré-estabelecida pelo próprio compilador ou criada pelo programador

Funções

Uma função é um conjunto de instruções que realizam uma operação

- Geralmente uma operação bem mais complicada do que aquela que pode ser realizada por uma única instrução
- Uma função pode ser pré-estabelecida pelo próprio compilador ou criada pelo programador
- Após ser criada, uma função pode ser encarada como um comando criado pelo programador

Funções

Uma função é um conjunto de instruções que realizam uma operação

- Geralmente uma operação bem mais complicada do que aquela que pode ser realizada por uma única instrução
- Uma função pode ser pré-estabelecida pelo próprio compilador ou criada pelo programador
- Após ser criada, uma função pode ser encarada como um comando criado pelo programador
- Uma função pode referenciar outras funções já criadas pelo programador

Funções

- O propósito de uma função deve ser bem definido. Uma função deve fazer uma e apenas uma operação

Funções

- O propósito de uma função deve ser bem definido. Uma função deve fazer uma e apenas uma operação
- Toda função tem um nome. Esse nome é utilizado para acionar ou chamar a função
  - printf(“printf eh uma funcao”);

Funções

- O propósito de uma função deve ser bem definido. Uma função deve fazer uma e apenas uma operação
- Toda função tem um nome. Esse nome é utilizado para acionar ou chamar a função
  - printf(“printf eh uma funcao”);
- O nome da função é sempre seguido de parenteses. Os parâmetros da função aparecem entre esses parenteses
  - printf(“printf eh uma funcao”);

Funções

- O propósito de uma função deve ser bem definido. Uma função deve fazer uma e apenas uma operação
- Toda função tem um nome. Esse nome é utilizado para acionar ou chamar a função
  - printf(“printf eh uma funcao”);
- O nome da função é sempre seguido de parenteses. Os parâmetros da função aparecem entre esses parenteses
  - printf(“printf eh uma funcao”);
- Uma função pode conter nenhum, um, ou vários parâmetros. Parâmetros são sempre separados por uma vírgula

A função **main**

Todo programa em C/C++ começa pela execução da função **main**

- O programa começa pela chamada à função **main**

A função **main**

Todo programa em C/C++ começa pela execução da função **main**

- O programa começa pela chamada à função **main**
- A partir daí, os comandos e funções inseridos em **main** são executados em sequência

Arquivos de cabeçalho

Quando compilamos um programa, o compilador invoca um pré-processador, que associa um rótulo a um trecho de código ou a um valor
A diretiva **#include** <arquivo> instrue o compilador a inserir o trecho de código armazenado em *arquivo*

- *O arquivo onde está o trecho de código a ser incluído é chamado de arquivo cabeçalho*

Arquivos de cabeçalho

Quando compilamos um programa, o compilador invoca um pré-processador, que associa um rótulo a um trecho de código ou a um valor

A diretiva **#include** <arquivo> instrue o compilador a inserir o trecho de código armazenado em *arquivo*

- *O arquivo onde está o trecho de código a ser incluído é chamado de arquivo cabeçalho*
- *Já conhecemos três:* **stdio.h**, **stdlib.h**, **math.h**

Arquivos de cabeçalho

Quando compilamos um programa, o compilador invoca um pré-processador, que associa um rótulo a um trecho de código ou a um valor

A diretiva **#include** <arquivo> instrue o compilador a inserir o trecho de código armazenado em *arquivo*

- *O arquivo onde está o trecho de código a ser incluído é chamado de arquivo cabeçalho*
- *Já conhecemos três:* **stdio.h**, **stdlib.h**, **math.h**
- *Cada arquivo contém trechos de código que implementam funções bem específicas*
  - *math.h: funções matemáticas*
  - *stdio.h: funções que imprimem e que lêem dados*
  - *stdlib.h: funções que implementam tarefas do sistema*

Constantes

Valores que não mudam no decorrer da execução do programa devem representados por constantes
A declaração é dada pela diretiva **#define** *rótulo valor*
Exemplo: #define PI 3.14

- O pré-processador substitui todas as referências a **PI** pelo valor 3.14
- Geralmente o rótulo é definido em letras maiúsculas

Constantes

Valores que não mudam no decorrer da execução do programa devem representados por constantes
A declaração é dada pela diretiva **#define** *rótulo valor*
Exemplo: #define PI 3.14

- O pré-processador substitui todas as referências a **PI** pelo valor 3.14
- Geralmente o rótulo é definido em letras maiúsculas

É claro que podemos utilizar variáveis ao invés de constantes, porém a variável ocupa mais memória. A variável é menos eficiente, pois é necessário realizar uma atribuição

Saída

A saída de dados na tela pode ser realizada pela função **printf**
Existem códigos de conversão que nos permitem imprimir dados de diferentes tipos:

- %c → para imprimir dados do tipo char
- %d ou %i → para imprimir dados do tipo inteiro
- %f ou %g → para imprimir dados do tipo float
- %s → para imprimir dados do tipo *string*

Saída

A saída de dados na tela pode ser realizada pela função **printf**
Existem códigos de conversão que nos permitem imprimir dados de diferentes tipos:

- %c → para imprimir dados do tipo char
- %d ou %i → para imprimir dados do tipo inteiro
- %f ou %g → para imprimir dados do tipo float
- %s → para imprimir dados do tipo *string*

Existem também os modificadores de tamanho:

- h → **short** (%hi, %hf, %hd)
- l → **long** (%li, %lf, %ld)

Saída

A saída de dados na tela pode ser realizada pela função **printf**
Existem códigos de conversão que nos permitem imprimir dados de diferentes tipos:

- %c → para imprimir dados do tipo char
- %d ou %i → para imprimir dados do tipo inteiro
- %f ou %g → para imprimir dados do tipo float
- %s → para imprimir dados do tipo *string*

Existem também os modificadores de tamanho:

- h → **short** (%hi, %hf, %hd)
- l → **long** (%li, %lf, %ld)

Podemos também definir a precisão:

- Casas decimais e "depois do ponto" → %6d, %5.1f

Entrada

A entrada de dados pelo teclado pode ser realizada pela função **scanf**

As mesmas regras de conversão, tamanho, e precisão são válidas

Contato

adrianov@dcc.ufmg.br