Programação Estruturada

Aula 1 - Introdução à linguagem C

Yuri Malheiros (yuri@ci.ufpb.br)

História

- C foi criado por Dennis Ritchie em 1972
  - Uma evolução da linguagem B
- Desenvolvida no Bell Labs no projeto UNIX

História

- C tem tido grande influência em outras linguagens de programação
  - C++
  - Java
  - C#
  - Python
  - JavaScript

Características

- C é uma linguagem de nível "médio"
  - Ela está entre alto nível (Java) e baixo nível (Assembly)
  - C prover acesso a conceitos em nível de máquina, mas possui estruturas de linguagens de alto nível
- C é uma linguagem pequena
- C é uma linguagem permissiva

Pontos fortes

- Eficiência
- Portabilidade
- Poder
- Flexibilidade
- Biblioteca padrão
- Integração com UNIX

Pontos fracos

- Programas sujeitos a erro
- Programas podem ser difíceis de entender
- Programas podem ser difíceis de modificar

Escrevendo um programa

- Vamos escrever um programa que exibe no terminal: "C ou não C, eis a questão"
- Crie um arquivo chamado `first.c`:

```c
#include <stdio.h>

int main(void) {
    printf("C ou não C, eis a questão\n");
    return 0;
}
```

Compilando

- C é uma linguagem compilada
  - Precisamos compilar o arquivo `first.c` para gerar um executável
- No terminal vamos executar:
  `% cc first.c`
- `%` é usado aqui para representar a linha de comando, não é para digitar esse caractere

Executando

- O arquivo executável `a.out` é gerado
- Para executar o programa:
  `% ./a.out`
- É possível definir um nome para o executável, para isso usamos a opção `-o` na compilação:
  `% cc first.c -o first`

Compiladores

- O GCC é o compilador C disponível no Linux
- Para utilizá-lo basta usar `gcc` ou invés de `cc`
- Para usar o GCC no Windows, você pode instalar o Mingw: http://mingw-w64.org/doku.php

Forma geral de um programa

```
diretivas

int main(void) {
    comandos
}
```

- Diretivas são comandos que modificam um programa antes da compilação
- Comandos são operações executadas ao rodar um programa
- Funções são blocos de código executáveis, por exemplo o `main`

Diretivas

- Antes de ser compilado, um programa C é editado por um pré-processador
- `#include <stdio.h>` é uma diretiva
- Ela especifica que o `stdio.h` deve ser incluído no nosso programa
- `stdio.h` contém informações sobre a biblioteca de IO
- Diretivas sempre começam com `#`

Funções

- São blocos para construção de programas
- Uma função é uma série de comandos
- Funções podem receber entradas
- Funções podem retornar uma saída
    - Para isso usamos o comando `return`

Funções

- Um programa em C obrigatoriamente tem uma função `main`
- Ela é chamada automaticamente quando o programa é executado

Funções

```c
#include <stdio.h>

int main(void) {
    printf("C ou não C, eis a questão\n");
    return 0;
}
```

- A palavra `int` antes do main especifica que a função retorna um valor inteiro
- A palavra `void` entre parênteses especifica que a função não recebe argumentos
- O `return 0` faz com que a função termine

Comandos

- Um comando é uma instrução que vai ser executada
- `printf("C ou não C, eis a questão\n");` é uma chamada de função
  - Ao chamar uma função, ela é executada com os argumentos passados para ela
  - `printf` é uma função que exibe um texto
- Todo comando deve terminar com `;`

Comentários

- Comentários de uma linha são definidos usando `//`

```c
// Meu primeiro programa em C

#include <stdio.h>

int main(void) {
    printf("C ou não C, eis a questão\n");
    return 0;
}
```

Comentários

- Comentários de múltiplas linhas começam com `/*` e terminam com `*/`

```c
/* Meu primeiro programa em C
Autor: Yuri */

#include <stdio.h>

int main(void) {
    printf("C ou não C, eis a questão\n");
    return 0;
}
```

Comentários

- É comum formatar comentários de múltiplas linhas da seguinte maneira:

```c
/*
 * Meu primeiro programa em C
 * Autor: Yuri
 */

#include <stdio.h>

int main(void) {
    printf("C ou não C, eis a questão\n");
    return 0;
}
```

Comentários

- Quando o comentário é curto, podemos colocá-lo numa mesma linha com código

```c
#include <stdio.h>

int main(void) {
    printf("C ou não C, eis a questão\n");  // Se Shakespeare fosse programador
    return 0;
}
```

Variáveis e atribuições

- É comum um programa precisar armazenar dados
- Em C e em diversas outras linguagens, guardamos dados durante a execução de um programa usando variáveis

Tipos

- Cada variável deve ter um tipo que especifica o tipo de dado que ela guarda
    - Estudaremos tipos em mais detalhes em aulas futuras
- C possui diversos tipos, entre eles o tipo inteiro `int` e o tipo ponto flutuante `float`
- Variáveis do tipo `int` guardam números inteiros, por exemplo, 10, 1392 ou -2553
- Variáveis do tipo `float` guardam números com separador decimal (vírgula), por exemplo, 379,125
- Escolher o tipo é fundamental, ele especifica como a variável é armazenada e que operações são suportadas

Declaração

- Variáveis precisam ser declaradas antes de serem usadas
- Para declarar uma variável, primeiro especificamos o tipo e depois o nome da variável:

```
int quantidade;
float valor;
```

Declaração

- Podemos combinar a declaração múltiplas variáveis de um mesmo tipo:

```
float valor, desconto, tamanho;
```

Atribuição

- Para guardar um valor numa variável, nós fazemos uma atribuição:

```
quantidade = 10;
valor = 49.50;
```

- Só podemos atribuir um valor a uma variável depois dela ser declarada

Exibindo o valor de uma variável

- Usando o `printf` podemos exibir o valor de uma variável
- Para exibir o valor de um inteiro, temos:

```c
#include <stdio.h>

int main(void) {
    int quantidade;
    quantidade = 10;

    printf("Quantidade: %d\n", quantidade);
    return 0;
}
```

- `%d` indica que o valor da variável `quantidade` deve ser exibido no seu lugar

Exibindo o valor de uma variável

- Para exibir uma variável do tipo `float` usamos `%f`

```c
#include <stdio.h>

int main(void) {
    int quantidade;
    float valor;

    quantidade = 10;
    valor = 49.50;

    printf("Quantidade: %d\n", quantidade);
    printf("Valor: %f\n", valor);
    return 0;
}
```

Exibindo o valor de uma variável

- Por padrão, `%f` exibe 6 dígitos depois do ponto
- Podemos customizar isso utilizando `%.Nf`, onde `N` é a quantidade de dígitos depois do . que será exibida

```c
#include <stdio.h>

int main(void) {
    int quantidade;
    float valor;

    quantidade = 10;
    valor = 49.50;

    printf("Quantidade: %d\n", quantidade);
    printf("Valor: %.2f\n", valor);
    return 0;
}
```

Inicialização

- Podemos declarar uma variável e já atribuir um valor a ela:

```
int quantidade = 10;
```

- Para múltiplas variáveis, temos:

```
float valor = 9.50, desconto = 1.25, tamanho = 4.2;
```

Expressões

- A atribuição de um valor não é limitada a constantes
- Podemos usar expressões matemáticas:

```c
#include <stdio.h>

int main(void) {
    int quantidade;
    float valor, total;

    quantidade = 4 + 6;
    valor = 20.75 - 10.25;
    total = quantidade * valor;

    printf("Total: %.2f\n", total);
    return 0;
}
```

Entrada

- Usamos a função `scanf` Para ler a dados digitados pelo usuário
- Para ler um valor inteiro, temos:

```c
#include <stdio.h>

int main(void) {
    int quantidade;

    scanf("%d", &quantidade);

    printf("Quantidade: %d\n", quantidade);
    return 0;
}
```

- Para ler um `int` usamos `%d` e para ler um `float` usamos `%f`
- Em aulas futuras explicarei o significado de `&`

Entrada

- Vamos fazer uma versão do programa que calcula o valor do total
- Agora o usuário entrará com os valores

```c
int main(void) {
    int quantidade;
    float valor, total;

    printf("Digite a quantidade: ");
    scanf("%d", &quantidade);

    printf("Digite o valor: ");
    scanf("%f", &valor);

    total = quantidade * valor;

    printf("Total: %.2f\n", total);
    return 0;
}
```

Constantes

- Quando um programa possui valores constantes é uma boa ideia nomeá-las
  - Facilita a leitura do programa e melhora sua organização
- Suponha que no cálculo do valor total do programa anterior, seja necessário adicionar um valor de 10.00 que representa a taxa de entrega
- Como esse valor é fixo, podemos criar uma constante

Constantes

- A diretiva `#define` é utilizada para criar uma constante

```c
#define TAXA_DE_ENTREGA 10.00

int main(void) {
    int quantidade;
    float valor, total;

    printf("Digite a quantidade: ");
    scanf("%d", &quantidade);

    printf("Digite o valor: ");
    scanf("%f", &valor);

    total = quantidade * valor + TAXA_DE_ENTREGA;

    printf("Total: %.2f\n", total);
    return 0;
}
```

Constantes

- Quando o programa é compilado, o pré-processador substitui o nome da constante pelo seu valor
- Por convenção, usamos letras maiúsculas nos nomes das constantes

Identificadores

- Os nomes que escolhemos para variáveis, funções, constantes, etc. são chamados de identificadores
- Em C, um identificador pode conter letras, dígitos e underlines, mas eles não podem começar com dígitos

Identificadores

- Válidos:
  - times10
  - get_next_char
  - _done
- Inválidos:
  - 10times
  - get-next-char

Identificadores

- C é case-sensitive
- Ou seja, ele distingue maiúsculas e minúsculas
    - job, joB, jOb, jOB, Job, JoB, JOb e JOB são todos identificadores diferentes

Identificadores

- Por convenção, costuma-se usar apenas letras minúsculas (com exceção das constantes) e separa-se as palavras com underlines
    - symbol_table , current_page , name_and_address

Identificadores

- Mas essa não é a única convenção
- Também é comum utilizar letra maiúscula para identificar o início de outra palavra no identificador (Camel Case)
    - symbolTable , currentPage , nameAndAddress
- Não temos certo ou errado aqui, escolha uma convenção e a siga!

Palavras reservadas

- Alguns nomes são reservados pela linguagem e não podem ser usados
  - auto
  - break
  - case
  - char
  - const
  - continue
  - default
  - do
  - double
  - else
  - enum
  - extern
  - float
  - for
  - goto
  - if
  - inline
  - int
  - long
  - register
  - restrict
  - return
  - short
  - signed
  - sizeof
  - static
  - struct
  - switch
  - typedef
  - union
  - unsigned
  - void
  - volatile
  - while