Departamento de Informática – UFPR

Algoritmos e Estruturas de Dados I

Prof.ª Dr.ª Mariane Cassenote

mariane@inf.ufpr.br

Fluxo de execução, entrada e saída, erros

Objetivos da aula

1. Entender o fluxo de execução, entrada e saída, erros

Aula elaborada com base na bibliografia da disciplina e
nos materiais dos Profs. Marcos Castilho e André Vignatti,
disponíveis no link

https://www.inf.ufpr.br/cursos/ci055/

Conceitos elementares de linguagens de programação

- **Fluxo de execução de um programa**
- **Comandos que manipulam dados e permitem interação com o usuário**
- Expressões aritméticas e lógicas
- Comandos que permitem alteração do fluxo de execução do programa

Conceitos elementares de linguagens de programação

- Um programa em Pascal contém obrigatoriamente um cabeçalho e um programa principal

```
program exemplo1;

begin
    comando 1;
    comando 2;
    comando 3;
end.
```

O fluxo de execução de um programa

```
program alomamae;

begin
    write('alomamae');
    writeln;
end.
```

- O programa inicia após o **begin**
- O programa termina quando encontra o **end.**
- Os comandos são executados **de cima para baixo,** linha por linha
- Em cada linha, da esquerda para direita

Código fonte versus código executável

```
program alomamae;

begin
     write('alomamae');
     writeln;
end.
```

- O texto acima é o código fonte
- Para obter o executável é preciso compilar

Comandos de entrada e saída

- Permitem a interação com o usuário
- Comandos de leitura
  - `read` e `readln`
- Comandos de saída
  - `write` e `writeln`

Exemplo

```
program le_e_imprime;
var numero: integer;

begin
     read (numero);
     writeln (numero);
end.
```

- Este programa lê um valor inteiro do teclado
- Depois imprime o valor lido na tela

Variáveis

```
program le_e_imprime;
var numero: integer;

begin
    read (numero);
    writeln (numero);
end.
```

- Variáveis são **endereços de memória**
- São acessíveis por um **identificador** escolhido pelo programador
- No caso: **numero**

Variáveis

1. Antes de iniciar o programa principal, o compilador solicita ao sistema operacional que reserve um espaço de memória para a variável `numero`, que deve ser interpretada como um valor **real**
2. Inicia o programa pela primeira instrução (`read (numero);`), que faz com que o computador espere o usuário digitar algo no teclado e apertar ENTER. Se o usuário digitar, por exemplo, o número 2, então o endereço de memória associado à `numero` conterá o valor 2
3. Executa a segunda instrução (`writeln (numero);`), que faz com que o computador busque na memória o valor (o conteúdo) de `numero`, que é 2, e imprima esta informação, isto é, 2, na tela e **mude de linha**
4. O programa termina sua execução

Variáveis

```
program le_e_imprime;
var numero: integer;

begin
     read (numero);
     writeln (numero);
end.
```

IMPORTANTE!

`writeln (numero);`

é diferente de

`writeln (“numero”);`

- Variáveis são **endereços de memória**
- São acessíveis por um **identificador** escolhido pelo programador
- No caso: `numero`

Variáveis

```
program le_e_imprime;
var numero: integer;

begin
     read (numero);
     writeln (numero);
end.
```

IMPORTANTE!

Acessar uma variável

sem valor definido no programa

pode resultar em

lixo de memória!

- Variáveis são **endereços de memória**
- São acessíveis por um **identificador** escolhido pelo programador
- No caso: `numero`

O ciclo edição–compilação–testes

- Para a parte prática da disciplina serão necessários:
  - Compilador Free Pascal
  - Editor de textos ASCII (não utilizar pacote Office e similares!)

O ciclo edição–compilação–testes

- No Linux, para o programa no arquivo **`exemplo1.pas`**, teremos
  - **`fpc exemplo1.pas`**
  - **`./exemplo1`**
- Arquivos
  - O **.pas** é para humanos lerem
  - O **.o** é uma tradução intermediária
  - O **executável** (no LInux não tem extensão) é o que o Modelo de Von Neumann consome

O ciclo edição–compilação–testes

```
program le_e_imprime;
var numero: integer;

begin
     read (numero);
     writeln (‘O valor digitado no teclado foi: ’, numero:0:2);
end.
```

- Sempre que o código for alterado, é necessário **recompilar** antes de executar!

Tipos de dados em Pascal

- Inteiros: São usados para números sem casas decimais
  - **integer**: É o tipo padrão. No Free Pascal moderno, ele ocupa 2 ou 4 bytes
  - **longint**: Também ocupa 4 bytes. É o tipo clássico para garantir 32 bits de precisão
  - **int64**: Ocupa 8 bytes (64 bits). Usado quando você precisa de números enormes
  - **byte**: Ocupa apenas 1 byte (8 bits). Armazena valores de 0 a 255.

Tipos de dados em Pascal

- Reais (Ponto Flutuante): Usados para números com casas decimais
  - real: Ocupa 8 bytes (em sistemas modernos equivale ao double)
  - single: Ocupa 4 bytes. Tem menor precisão que o real
  - extended: Pode ocupar 10 ou 12 bytes, dependendo do processador. Oferece a maior precisão possível no hardware

Tipos de dados em Pascal

- Caractere e Lógicos
  - boolean: Ocupa 1 byte. Armazena apenas true ou false. Embora precise de apenas 1 bit para a informação, ele ocupa 1 byte inteiro por ser a menor unidade de endereço da memória
  - char: Ocupa 1 byte. Armazena um único caractere da tabela ASCII (como 'A', '7' ou '$')
- Tipos compostos
  - String e Array dependem do tamanho especificado na declaração

Dados e Memória

Se o seu computador tem 8GB de RAM e cada integer ocupa 4 bytes,

quantas variáveis inteiras caberiam teoricamente na memória?

Dados e Memória

Se o seu computador tem 8GB de RAM e cada integer ocupa 4 bytes, quantas variáveis inteiras caberiam teoricamente na memória?

1. Converter GB para Bytes

- 1 KB = 1024 bytes
- 1 MB = 1024 KB = 1.048.576 bytes
- 1 GB = 1024 MB = 1.073.741.824 bytes
- 8 GB = 8 × 1.073.741.824 = 8.589.934.592 bytes

2. Dividir pelo tamanho do integer (4 bytes)

Teoricamente, caberiam **2.147.483.648** variáveis inteiras (pouco mais de 2 bilhões)

Linguagens tipadas

- São as que exigem explicitamente dizer qual é o tipo da variável
- Ou seja, como os bits serão interpretados pelo compilador
- Pascal é uma linguagem tipada
- É necessário respeitar o tipo de valor esperado pelo programa
    - Usar valores maiores do que o limite do tipo da variável pode ocasionar **overflow**

Linguagens fortemente tipadas

- Não permitem que se altere o tipo do dado durante a execução do programa
- Pascal é fortemente tipada

AVISO

Até ordem contrária, todos os nossos programas usarão o tipo **`longint`**.

Isso não nos impedirá de aprender os princípios de programação e algoritmos para depois nos preocuparmos com detalhes.

Fim do tópico

- O conteúdo da aula até este ponto está no livro no Capítulo 5, Seções de 5.1 até 5.4
- Depois do esclarecimento de dúvidas passaremos às expressões aritméticas e booleanas

https://www.inf.ufpr.br/marcos/livro_alg1/livro_alg1.pdf

Departamento de Informática – UFPR

Algoritmos e Estruturas de Dados I

Prof.ª Dr.ª Mariane Cassenote

mariane@inf.ufpr.br

Fluxo de execução, entrada e saída, erros