Universidade Federal de Ouro Preto
Campus João Monlevade

CSI 488 – Algoritmos e Estruturas de Dados I

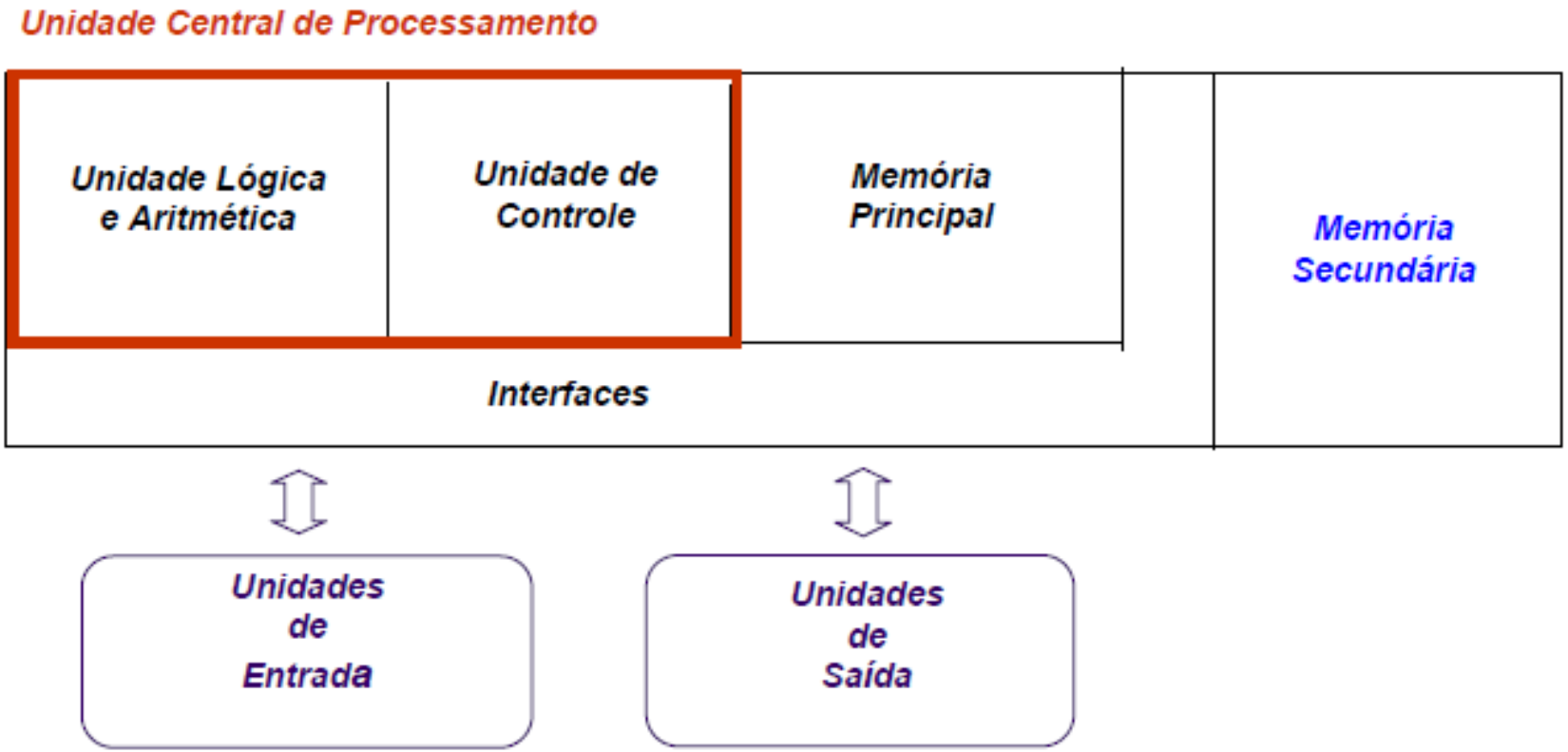
Tópicos de Revisão

Prof. Mateus Ferreira Satler

Índice

1. Computador Clássico

- Arquitetura de *von Neumann*



1. Computador Clássico

- **Memória Principal**
  - Representação de uma memória de 1 KB:
  - O processador acessa o conteúdo de um byte a partir do endereço desse byte.

Endereço	Byte							
0								
1								
2	0	1	0	0	0	0	0	1
...								
1023								

No byte de endereço 2 está armazenado o código ASCII do caractere “A”.

1. Computador Clássico

- **Algoritmos**
  - São sequências de passos, precisos e bem definidos, que descrevem como realizar uma tarefa
  - Podem ser especificados em português, português estruturado, fluxogramas, linguagens de programação, etc.

2. Acesso à Entrada e Saída

- Geralmente, desejamos que nosso programa receba informações, processe-as e nos mostre um resultado.

- Saída de Dados
  - Comando **printf**
    - **Exemplo:** `printf("Meu primeiro programa!");`

- Entrada de Dados
  - Comando **scanf**
    - **Exemplo:** `int numero; scanf("%d", &numero);`

3. Acesso à Memória

1. **Variáveis**

   - A memória funciona como caixas empilhadas que guardam as informações que um programa manipula.
   - Cada caixa tem um tamanho diferente e guarda um tipo específico de informação.
   - Os programas acessam as informações em uma caixa através de uma variável, que funciona como um rótulo (ou nome) para esta caixa.
   - Uma variável possui, então, um tipo e um nome.

3. Acesso à Memória

1. **Variáveis**

   - **Sintaxe:**

     ```
     tipo nome;
     tipo nome1, nome2, nome3, ... ;
     ```

   - **Tipos:**

     - `int`
     - `float / double`
     - `Char`

   - Operador de Atribuição =

3. Acesso à Memória

2. Vetores

- Quando desejamos utilizar diversas variáveis de um mesmo tipo, pode ser inviável declarar cada uma das variáveis.
- Vetor é uma coleção de variáveis do mesmo tipo referenciadas por um nome comum.
  - Permitem acesso por meio de um índice inteiro;
  - São criados (alocados) em posições contíguas na memória;
  - Possuem tamanho (dimensão) pré-definido, ou seja, o tamanho de um vetor não pode ser alterado durante a execução do programa;
  - Índices fora dos limites causam comportamento imprevisível no programa.

3. Acesso à Memória

2. Vetores

- **Sintaxe:**

```
tipo_vetor nome_do_vetor [tamanho_do_vetor];
```

- Ler, preencher e imprimir um vetor.
- Vetores como parâmetros de sub-rotinas.
- Inicialização de vetores.

3. Acesso à Memória

2. Vetores

◦ Vetores de Caracteres (Strings)

- A linguagem C não possui o tipo string (cadeia de caracteres) explicitamente, mas podemos considerar um vetor de caracteres como uma string.
- Em C, uma string é sempre terminada pelo caractere especial: `'\0'`
- Logo, ao declararmos um vetor de caracteres, devemos somar 1 à quantidade desejada de caracteres!
- Exemplo: **`char st1[7] = "string";`**

3. Acesso à Memória

2. Vetores

◦ Vetores de Caracteres (Strings)

- Declaração e inicialização de cadeiras de caracteres.
- Leitura e impressão de cadeias de caracteres:
  - Para ler uma cadeia de caracteres contendo espaços, indique que a cadeia deve terminar com a quebra de linha, assim: `%[^\n]s`
  - Função **`gets()`**
  - Função **`puts()`**

3. Acesso à Memória

3. Matrizes

- Uma matriz (ou vetor multidimensional) é um vetor (uma coleção de variáveis de um determinado tipo) com mais de uma dimensão.
- O total de variáveis que uma matriz possui é igual ao produto entre a quantidade de variáveis de cada uma de suas dimensões.
- **Sintaxe**:

```
tipo nome [dim1] [dim2];
tipo nome [dim1] [dim2] [dim3] ... [dimN];
```

3. Acesso à Memória

3. Matrizes

- Ler, preencher e imprimir uma matriz.
- Matrizes como parâmetros de sub-rotinas.
- Inicialização de matrizes.
- Matrizes de caracteres.

3. Acesso à Memória

4. Ponteiros

- Toda variável possui um endereço de memória.
- Ponteiro (para um tipo) é um tipo de dado especial que armazena endereços de memória (onde cabem valores do tipo apontado).
- Uma variável que é um ponteiro de um tipo A armazena o endereço de uma outra variável também do tipo A.
- Ponteiros permitem alocação dinâmica de memória, ou seja, alocação de memória enquanto o programa já está sendo executado.

3. Acesso à Memória

4. Ponteiros

- **Sintaxe:**

```
tipo * nome_variável;
```

- Operador **&**
- Operador *****
- Ponteiros e Vetores
- Passagem de parâmetros por **Cópia** e por **Referência**

3. Acesso à Memória

5. Alocação Dinâmica de Memória

- Pode-se alocar dinamicamente (quando o programa está em execução) uma quantidade de memória contígua e associá-la a um ponteiro.
- Isto permite criar programas sem saber, em tempo de codificação, qual o tamanho dos dados a serem armazenados (vetores, matrizes, etc).
- Desta forma, não é necessário armazenar mais memória do que de fato se deseja usar.

3. Acesso à Memória

5. Alocação Dinâmica de Memória

- **Sintaxe:**

```
void* calloc(int blocos, int tamanho);
void* malloc(int qtde_bytes);
free(void* ponteiro);
```

- Função **`sizeof()`**
- Alocação Dinâmica de Matrizes

4. Expressões

- Expressões são constantes, variáveis ou operações entre estas duas últimas.
- A atribuição de uma expressão a uma variável define o valor desta última.
- Basicamente 3 tipos de expressões:

1. Expressões Aritméticas
2. Expressões Relacionais
3. Expressões Lógicas

4. Expressões

1. Expressões Aritméticas

Operação	Operador	Expressão Algébrica	Expressão em C
Adição	+	f + 3	f + 3
Subtração	-	5 - p	5 - p
Multiplicação	*	b m	b * m
Divisão	/	4 / 3	4 / 3
Resto	%	4 mod 2	4 % 2

4. Expressões

2. Expressões Relacionais

- Os operadores de igualdade, em C, são:
  - `==` : igual;
  - `!=` : diferente;
- Os operadores relacionais, em C, são:
  - `>` : maior que;
  - `>=` : maior ou igual que;
  - `<` : menor que;
  - `<=` : menor ou igual que.

4. Expressões

3. Expressões Lógicas

- Os operadores lógicos, em C, são:
  - `&&  : operador E`
  - `| | : operador OU`
  - `!   : operador de NEGAÇÃO`

5. Controle

Bloco de Comandos

- Um Bloco de comandos é um conjunto de comandos delimitados por { e } e define o escopo das variáveis.
- Escopo de variável é a região do programa em que a variável pode ser acessada.

```
{
    int x;
    x = 1;
    x = x + 10;
}
```

5.1. Comandos Condicionais

- Permitem alterar o fluxo de execução, escolhendo se um bloco de comandos deve ou não ser executado, com base em uma expressão lógica ou relacional.

- Em C, existem os seguintes comandos condicionais:

1. `if`
2. `if-else`
3. `switch`

5.1. Comandos Condicionais

1. **Comando `if`**

   - Sintaxe:

```
if ( <expressão lógica ou relacional>)
  comando_único;
```

   - Ou:

```
if ( <expressão lógica ou relacional>) {
  comandos;
  outros_comandos;
}
```

   - O **`comando_único`** ou o **`bloco de comandos`** é executado quando **`<expressão lógica ou relacional>`** é verdadeira.

5.1. Comandos Condicionais

2. Comando `if-else`

- Sintaxe:

```
if ( <expressão lógica ou relacional>) {
  comandos_para_expressão_verdadeira;
}

else {
  comandos_para_expressão_falsa;
}
```

5.1. Comandos Condicionais

3. Comando `switch`

- Sintaxe:

```
switch (<variável int ou char>){
  case val1: comandos_para_variável_igual_a_val1;
  break;

  case val2: comandos_para_variável_igual_a_val2;
  break;

  default: comandos_em_caso_de_falha_dos_testes;
}
```

5.2. Comandos de Repetição

- Comandos de repetição permitem que se execute um bloco de comandos mais de uma vez.

- **Exemplo:**
  - Como imprimir os números de 1 a 10?
    Temos que repetir o comando print 10 vezes.

- Os comandos de repetição, em C, são:

1. `while`
2. `do ... while`
3. `for`

5.2. Comandos de Repetição

1. **Comando `while`**

   - Sintaxe:

```
while (condição)
  comando;
```

   - Ou:

```
while (condição){
  comandos;
  outros_comandos;
}
```

5.2. Comandos de Repetição

2. Comando `do ... while`

- Sintaxe:

```
do
   comando;
while (condição);
```

- Ou:

```
do{
   comando1;
   comando2; ...
} while (condição);
```

5.2. Comandos de Repetição

3. Comando `for`

- Sintaxe:

```
for (início; teste; modificação)
 comando;
```

- Ou:

```
for (início; teste; modificação){
   comando1;
   omando2;...
}
```

  - **`início:`** atribuições iniciais de variáveis
  - **`teste:`** teste a ser realizado para continuar o laço
  - **`modificação:`** alteração da variável de controle

5.3. Subrotinas

- Frequentemente, dividimos um problema maior em problemas menores e resolvemos os problemas menores.
- As sub-rotinas devem codificar a solução para um problema pequeno e específico.
- Sub-rotinas podem ser:

1. Funções
2. Procedimentos

5.3. Subrotinas

1. **Funções**

   - Uma função executa comandos e **retorna** algum resultado, cujo tipo é determinado por **`tipo_retorno`**.

   - Sintaxe:

```
tipo_retorno nome (tipo parâmetro1, ..., tipo parâmetroN)
{
  comandos;
  return variável_tipo_retorno;
}
```

5.3. Subrotinas

2. Procedimentos

- Um procedimento é um tipo especial de função que executa comandos e não retorna um resultado.

- Sintaxe:

```
void nome (tipo parâmetro1, ..., tipo parâmetroN)
{
  comandos;
}
```

6. Registros ou Estruturas

- Um registro (struct) é a forma de agrupar variáveis em C, sejam elas de mesmo tipo ou de tipos diferentes
- As variáveis unidas em um registro se relacionam de forma a criar um contexto maior
- Exemplos de uso de registros:
  - **Registro de Alunos**: armazena nome, matrícula, médias, faltas, etc.
  - **Registro de Pacientes**: armazena nome, endereço, convênio, histórico hospitalar

6. Registros ou Estruturas

- Sintaxe:

```
struct nome_tipo_registro {
  tipo_1 variavel_1;
  tipo_2 variavel_2;
  ...
  tipo_n variavel_n;
};
```

- Devemos declarar variáveis deste novo tipo assim:

```
struct nome_tipo_registro variavel_registro;
```

6. Registros ou Estruturas

- Sinônimos de tipos com **`typedef`**
  - Sintaxe:

```
typedef struct nome_tipo_registro {
  tipo_1 variavel_1;
  tipo_2 variavel_2;
  ...
  tipo_n variavel_n;
}nome_tipo;
```

  - Devemos declarar variáveis deste novo tipo assim:

```
nome_tipo variavel_registro;
```

6. Registros ou Estruturas

- Acesso aos campos do registro
  - Operador .
- Vetores de registros
- Registros como parâmetros de sub-rotinas
- Alocação dinâmica de registros

7. Referências

- Material de aula do Prof. Ricardo Anido, da UNICAMP: http://www.ic.unicamp.br/~ranido/mc102/

- Material de aula da Profa. Virgínia F. Mota: https://sites.google.com/site/virginiaferm/home/disciplinas

- DEITEL, P; DEITEL, H. *C How to Program*. 6a Ed. Pearson, 2010.