Capacitação em Linguagem C
Parte 1

Andrey Souto Maior
Giuseppe Portolese

Universidade Estadual de Maringá - Centro de Tecnologia
Departamento de Informática

21 de outubro de 2015

Sumário I

Sumário II

Algoritmo

Definição Informal

"Procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor como ou conjunto de valores como saída. Portanto, um algoritmo é uma sequência de passos computacionais que transformam a entrada na saída" (CLRS, 2002).

Linguagem de programação

"Uma forma padronizada de comunicar instruções para um computador".

Linguagem utilizada para escrever algoritmos que podem ser entendidos pelo computador.

Categorias:

- Aplicações científicas - Fortran
- Aplicações comerciais - Cobol
- Inteligência artificial - LISP
- Software web - PHP
- Propósito geral - C, Java

Linguagem C

- Propósito geral
- Compilada (GNU Compiler Collection - GCC)
- Estruturada
- Imperativa
- Procedural
- Padronizada

Um simples programa em C

```c
#include <stdio.h>

main(){
    printf("Hello World");
}
```

Tipos de dados básicos

Tipo	Descrição	Tamanho
char	contém um único caractere	1 byte
int	um número inteiro	4 bytes
float	ponto flutuante de precisão simples	4 bytes
double	ponto flutuante de precisão dupla	8 bytes

- **Variações**
  - short - 2 bytes
  - long - 8 bytes

Constantes

- "Variáveis" de valor fixo

```
#define TRUE 1
#define FALSE 0

main(){
    ...
}
```

stdio.h

- **printf(Formato, Argumentos...)**:
  Função usada para imprimir na tela
- **scanf(Formato, EndereçoArgumentos)**:
  Função usada para ler a entrada do usuário e passar seu valor para o endereço dado.

stdio.h

```
main(){
    printf("Hello World");
}
```

stdio.h

```
int esperaTecla(){
    char in;
    scanf("%c",&in);
    return in;
}
```

Vetores sequenciais

Variável composta dada por uma coleção de elementos individuais com as seguintes características:

- É ordenado: os elementos de um vetor são indexados de forma ordenada
- É homogêneo: todo valor armazenado em um mesmo vetor deve ser do mesmo tipo

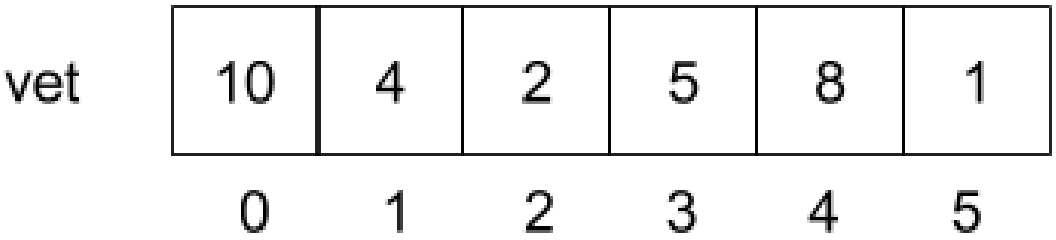

Figura 1: Vetor vet

Vetores sequenciais

```c
#include <stdio.h>

main(){
    int vet[6];
    int outroVet[] = {0, 1, 2, 3, 4, 5};
}
```

Vetores esparsos

Nem todas as posições de um vetor são ocupadas todo o tempo.
Em muitos casos declaramos um vetor muito grande e vários de seus espaços continuam com o valor nulo de seu tipo.
Veremos como resolver esse problema em breve.

Sequência de caracteres - String

```c
#include <stdio.h>

main(){
    char string[20];
    char outraString[] = "Esta eh uma string";
}
```

Sequencialidade e atribuição

É uma estrutura de desvio do fluxo de controle presente em linguagens de programação que realiza um conjunto predeterminado de comandos de forma sequencial, de cima para baixo, na ordem em que foram declarados.

Exemplo:

```
declaração1;
comando1;
comando2;
comando3;
```

Obs: na estrutura sequencial, as declarações sempre ocorrem antes dos comandos.

Desafio

Faça um código que:

1. Receba o primeiro nome, id e se é aluno (Ou não) de três participantes
2. Imprima as informações das três pessoas na tela

Exemplo

```c
#include <stdio.h>

int main(){
    char nome0[30], nome1[30], nome2[30];
    int id[3];

    printf("Nome0: ");
    scanf("%s",nome0);
    printf("ID: ");
    scanf("%d",&id[0]);

    printf("Nome=[%s] ID=[%d]\n",nome0,id[0]);

    return 0;
}
```

Operadores aritméticos

Operação	**Operador**
soma	$+$
subtração	$-$
multiplicação	$*$
divisão	$/$
módulo/resto	%
incremento	$++$
decremento	$--$

Operadores relacionais e lógicos

Operação	**Operador**
atribuição	=
maior	>
maior igual	>=
menor	<
menor igual	<=
igualdade	==
desigualdade	! =
negação	!

Comando if

```
if (expressão) {
    comando1;
} else {
    comando2;
}
```

```
if (expressão1) {
    comando1;
} else if (expressão2) {
    comando2;
} else if (expressão3) }
    comando3;
} else {
    comando4;
}
```

Exemplo

Faça um programa que leia uma entrada (Inteira) e tenha como saída:

- **Zero!** se a entrada for 0
- **Maior que zero!** se a entrada for maior que 0
- **Menor que zero!** você já entendeu

Exemplo

```c
#include <stdio.h>

int main(){
    int entrada;
    scanf("%d",&entrada);

    if(0 == entrada){
        printf("Zero!\n");
    }else if(0 < entrada){
        printf("Maior que zero!\n");
    }else{
        printf("Menor que zero!\n");
    }
}
```

Comando switch

```
switch (variável) {
    case constante1 : comando1;
        break;
    case constante2 : comando2;
        break;
    default : comando3;
        break;
}
```

Exemplo

Faça um programa de menu que leia uma entrada (Inteira) e tenha como saída:

- **Opcao 0** se a entrada for 0
- **Opcao 1** se a entrada for 1
- **Opcao Invalida** para qualquer outra entrada

Exemplo

```c
#include <stdio.h>

int main(){
    int entrada;
    scanf("%d",&entrada);

    switch(entrada){
        case 0:
            printf("Opcao 0\n");
            break;
        case 1:
            printf("Opcao 1\n");
            break;
        default:
            printf("Opcao Invalida\n");
            break;
    }
}
```

Comando for

```
for (expressão1, expressão2, expressão3) {
    comando;
}
```

expressão1 - inicialização do laço *expressão2* - condição que finaliza o laço *expressão3* - incremento do laço

Exemplo

Faça um programa que leia um número (Inteiro) N e escreva na tela:

- **Eu sei fazer for!**

N vezes

Exemplo

```c
#include <stdio.h>

int main(){
    int i, N;
    scanf("%d",&N);

    for(i = 0; i < N; i++){
        printf("Eu sei fazer for!\n");
    }
}
```

Comandos while e do-while

```
while (expressão) {
    comando;
}
```

```
do {
    comando;
} while (expressão);
```

Comandos while e do-while

```
while (expressão) {
    comando;
}
```

```
do {
    comando;
} while (expressão);
```

Pergunta: Existe diferença semântica entre *while* e *do-while*? Qual?

Exemplo

Faça um programa que leia um número (Inteiro) N e escreva na tela:

- **Eu sei fazer while!**

N vezes

Exemplo

```c
#include <stdio.h>

int main(){
    int i, N;
    scanf("%d",&N);

    i = 0;
    while(i < N){
        printf("Eu sei fazer while!\n");
        i++;
    }
}
```

Comandos break e continue

- **break:**
  Serve para interromper o laço atual, fazendo com que o programa continue a executar após o laço.
- **continue:**
  Serve para interromper somente a iteração atual do laço, fazendo com que o programa continue a executar na próxima iteração do laço atual.

Exemplo

Faça um programa que leia um número (Inteiro) N e escreva na tela a contagem de todos os números em sequencia, parando a execução se o número chegar em 5.

Exemplo

```c
#include <stdio.h>

int main(){
    int i,N;
    scanf("%d",&N);

    for(i=0; i<N; i++){
     if(5 == i){
            break;
        }
        printf("%d\n",i);
    }
}
```

Exemplo

Faça um programa que leia um número (Inteiro) N e escreva na tela a contagem de todos os números em sequencia, exceto se o número for 5.

Exemplo

```c
#include <stdio.h>

int main(){
    int i,N;
    scanf("%d",&N);

    for(i=0; i<N; i++){
     if(5 == i){
            continue;
        }
        printf("%d\n",i);
    }
}
```

Blocos

Sessão do código que possui suas próprias variáveis locais, cujo escopo é minimizado. As variáveis são alocadas quando entra na sessão e desalocada quando sai da sessão.

Exemplo

```c
#include <stdio.h>

int main(){
    int i,N;
    scanf("%d",&N);

    for(i=0; i<N; i++){
     int j = 5;
        printf("%d\n",i + j);
    }
}
```

j não pode ser acessada fora do for

Funções

Podemos dizer que funções são sub-rotinas que executam uma tarefa particular.

tipo do retorno nome da funcao (*arg1*, *agr2*, ...);

Procedimentos

A principal diferença entre uma função e um procedimento está no fato de que uma função obrigatoriamente retorna um valor, enquanto que um procedimento não retorna valor algum, ou seja o procedimento apenas executa uma ação.

Exemplo

```c
#include <stdio.h>

int incrementa(int n){
    return n + 1;
}

void sorria(){
    printf(":)\n");
}

int main(){
    printf("%d\n",incrementa(0));
    sorria();
}
```

Escopo de variáveis

O escopo é a parte do programa dentro da qual o nome pode ser usado. Para uma variável declarada no início de uma função, por exemplo, o escopo é a função em que o nome é declarado. O escopo de uma variável dura desde o ponto em que é declarada até o final da função ou da execução do programa.

Pergunta: Existe diferença entre *bloco* e *escopo*? Qual?

Exemplo

```
void foo(){
    int count = 2;

    while (count < 10){
        int count = 0;
        count++;
    }
}

int main(){
    int count = 1;

    foo();

    return 0;
}
```

Alocação de variáveis

- Estática

  Os dados tem um tamanho fixo e estão organizados sequencialmente na memória do computador.

- Dinâmica

  Os dados não precisam ter um tamanho fixo, pois podemos definir para cada dado quanto de memória que desejamos usar. A distribuição na memória não é necessariamente sequencial.

  - ponteiros
  - *malloc*
  - *realloc*

Ponteiros

Um ponteiro (ou apontador ou pointer) é um tipo especial de variável que armazena endereços. Um ponteiro pode ter o valor especial **NULL** que não é endereço de lugar algum.

Operadores:

& - retorna o endereço da variável

* - declaração de ponteiro ou derreferenciação

-> - seleção de elemento por ponteiro

Ponteiros

```c
#include <stdio.h>

main(){
int *p;
    int num = 5;

    p = &num;
    printf("O numero eh: %d", *p);
}
```

Passagem de parâmetros

A linguagem de programação C permite que os parâmetros sejam passados para as funções de duas maneiras:

- **Por valor**

  Como o próprio nome diz, uma expressão pode ser utilizada na chamada. O valor da expressão é calculada, e o valor resultante é passado para a execução da função.

- **Por referência**

  O endereço de uma variável deve ser passado na chamada da função. Dessa forma, a função pode modificar a variável diretamente, o que em geral não é recomendável, mas há situações onde esse recurso é necessário, por exemplo, para a criação de funções que devolvem mais de um valor.

Arquivos

- Abertura de um arquivo (fopen)
- Leitura de um arquivo (fgets)
- Escrita em um arquivo (fwrite)
- Fechar um arquivo (fclose)

Exemplo

```c
#include <stdio.h>

int main(){
    FILE *fp;
    char in[10];

    fp = fopen("./teste.c","r");

    fgets(in, 10, fp);
    printf("%s\n",in);

    fclose(fp);

    return 0;
}
```

Refinamento sucessivo

Dividir e conquistar!

Partindo do problema geral, o dividimos em problemas menores (os subproblemas). Após resolver cada subproblema separado, juntamos todas as soluções e temos a solução do problema geral.

Etapas:

Dividir o problema em subproblemas.

Conquistar os subproblemas, resolvendo-os.

Combinar as soluções dadas aos subproblemas, a fim de formar a solução para o problema original.

Exemplo:

Merge-sort

Recursividade

A recursividade é a definição de uma subrotina (função ou procedimento) que pode invocar a si mesma.

- **Recursão direta**

  É quando uma função chama a si mesma fazendo com isso um loop.

- **Recursão indireta**

  É quando uma função faz a chamada a outra função e esta por sua vez faz uma chamada a primeira.

- **Caso base**

  É uma situação em que a função recursiva pára de chamá-la novamente. O caso base é de extrema importância na implementação da recursividade, pois ele evita o loop infinito.

Recursividade

Direta

```
foo() {
    if (expressao) {
        return 1;
    } else {
        foo();
    }
}
```

Indireta

```
foo1() {
    if (expressao1) {
        return 1;
    } else {
        foo();
    }
}

foo() {
    if (expressao2) {
        return 1;
    } else {
        foo1();
    }
}
```

Exemplo

```c
#include <stdio.h>

void decrementa(int n){
    if(0 >= n){
        printf("Fim c:\n");
    }else{
        printf("N == %d\n",n);
        decrementa(n-1);
    }
}

int main(){
    decrementa(5);
}
```

Desafio

Faça um programa que calcule o fatorial de uma entrada.

Desafio

```c
#include <stdio.h>

int fatorial(int n, int fat){
    if(1 >= n){
        return fat;
    }else{
        return fatorial(n-1, fat*n);
    }
}

int main(){
    int entrada;

    scanf("%d",&entrada);
    printf("%d\n", fatorial(entrada,1));
}
```