Programação

Flavius Gorgônio
flavius@dct.ufrn.br

Agenda

- Algoritmos, linguagens de programação e programas de computador
- Entrada e saída de dados
- Variáveis e tipos de dados
- Alocação de variáveis em memória
- Nomes e identificadores
- Palavras especiais, palavras-chaves e palavras reservadas
- Atributos de uma variável
- Escopo e tempo de vida de variáveis
- Aliases (Apelidos)
- Conversão de tipos explícita e implícita
- Vinculação

Algoritmo

Alguns conceitos de ALGORITMO:

"Seqüência de passos para se atingir um objetivo" (Genérico, autor desconhecido)

"Sequência finita de instruções bem definidas e não ambíguas, cada uma das quais devendo ser executadas mecânica ou eletronicamente em um intervalo de tempo finito e com uma quantidade de esforço finita." (Wikipédia, 2018)

Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output. (Cormen, Introduction to Algorithms)

```
Algoritmo para substituir lâmpadas queimadas

Início
      Enquanto houver lâmpadas queimadas
            Escolher uma lâmpada queimada
            Verifica se o interruptor está desligado
            Pegar uma lâmpada nova
            Pegar uma escada
            Posicionar a escada embaixo da lâmpada
            Subir os degraus até a altura apropriada
            Retirar a lâmpada queimada
            Colocar a lâmpada nova
            Descer da escada
            Acionar o interruptor
            Se a lâmpada não acender, então
                  Chamar um eletricista
            Senão
                  Separar lâmpada queimada
            Fim_Se
      Fim_Enquanto
      Jogar as lâmpadas queimadas no lixo
      Guardar a escada
Fim
```

Programa de computador

O que é um PROGRAMA DE COMPUTADOR?

"Uma sequência de ações, escrita em uma linguagem de programação, que deve ser executada pelo computador de forma automática"

```
// Trecho de um programa em PASCAL


procedure preencheVetor(var nomes:VETOR);
var
      i : integer;
      nomeLido : PALAVRA;
begin
      i := 1;
      write('Digite nome: ');
      readln(nomeLido);
      while(nomeLido <> 'FIM') do
      begin
            nomes[i] := nomeLido;
            write('Digite nome: ');
            readln(nomeLido);
            i := i + 1;
      end;
end;
```

Linguagem de programação

“Conjunto de instruções predefinidas utilizada para escrever os programas de computador”

Exemplos de linguagens de programação:

- FORTRAN
- BASIC
- PASCAL
- PYTHON
- C
- C++
- C#
- JAVA
- PHP
- ...

A linguagem de programação C

- Conceitos fundamentais da linguagem C
  - Conjunto reduzido de instruções
  - Rico conjunto de operadores
  - Algo grau de expressividade
  - Implementação através de compiladores
- Poder, eficiência e flexibilidade
  - Poder e flexibilidade versus segurança
  - Eficiência versus segurança
- Fortemente baseada em funções
  - Encoraja a escrita de funções pequenas e modulares
  - Foca na reutilização de código

Exemplo de código em C

```
/* Programa para conversão de temperatura Celsius em Fahrenheit */

#include <stdio.h>

float converte (float c)                                    /* funcao auxiliar que faz a conversao */
{
      float f;
      f = 1.8*c + 32;
      return f;
}

int main (void)
{
      float t1;
      float t2;
      printf("Digite a temperatura em Celsius: ");          /* mostra mensagem para usuario       */
      scanf("%f",&t1);                                      /* captura valor entrado via teclado  */
      t2 = converte(t1);                                    /* chama a funcao que faz a conversao */
      printf("A temperatura em Fahrenheit é: %f\n", t2);    /* exibe resultado                    */
      return 0;
}
```

https://repl.it/@flaviusgorgonio/tempC2Fc

Entrada e saída de dados

Recursos de entrada e saída de dados são essenciais para o desenvolvimento de interface de programas com os usuários, com arquivos de dados e com outros programas

Algumas linguagens mais tradicionais possuíam *comandos* de entrada e saída, entretanto, as mais modernas optam por utilizar *funções* de entrada e saída

Exs.:

- Python substituiu o comando **`print`** da versão 2.x pela função **`print()`** na versão 3.x
- C não possui comandos de entrada e saída e as suas funções de entrada e saída estão em bibliotecas externas, que podem ou não ser carregadas
- Java, C++ e outras LPs OO usam métodos de entrada e saída

Entrada de dados padrão em C

A função **`scanf()`** é utilizada para entrada de dados em C:

Sintaxe:

```
scanf(formato, lista_de_enderecos_de_variaveis);
```

Exemplos:

```
scanf("%d", &x);

scanf("%c", &sexo);

scanf("%d %f %c", &idade, &salario, &estadoCivil);

scanf("%s", nome);
```

Saída de dados padrão em C

A função **`printf()`** é utilizada para saída de dados em C:

Sintaxe:

```
printf(formato, lista_de_variaveis_ou_expressoes);
```

Exemplos:

```
printf("%d", x);

printf("Sexo: %c\n", sexo);

printf("Salário: R$ %.2f", salario);

printf("Nome: %s", nome);
```

Variáveis e tipos de dados

Uma variável representa um espaço de memória para armazenar determinado tipo de dado

Em C, todas as variáveis devem ser declaradas explicitamente, de forma que o compilador conheça os tipos de cada variável em tempo de compilação

Isso permite identificar diversos erros de tipos (linguagem fortemente tipada ou tipificada)

```
// declaracao de variaveis de tipos primitivos

int a, b;
float c;
double d, e, f;
char g, h;



// tipo ponteiro

int *px, *py;



// tipo vetor

int vetor[10];
char matriz[3][3];
```

Tipos básicos em C

Tipo	Tamanho	Representatividade
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255
short int	2 bytes	-32 768 a 32 767
unsigned short int	2 bytes	0 a 65 535
long int	4 bytes	-2 147 483 648 a 2 147 438 647
unsigned long int	4 bytes	0 a 4 294 967 295
float	4 bytes	$\pm 10^{-38}$ a 10^{38}
double	8 bytes	$\pm 10^{-308}$ a 10^{308}

Atividade 1

Tipos de dados em C podem sofrer variações em função de detalhes de máquina ou da implementação do compilador. O operador **`sizeof()`** retorna o tamanho (número de bytes) de um tipo ou de uma variável

1. Descubra o tamanho dos tipos de dados básicos da implementação da linguagem C na sua máquina
2. Repita o processo para encontrar o tamanho dos tipos de dados básicos da implementação da máquina virtual C no site repl.it
3. Compare os resultados obtidos

```
// Use esse código como modelo

#include <stdio.h>

int main(void) {

    printf("Tamanho dos tipos de dados\n\n");

    printf("char: %lu bytes\n", sizeof(char));

    printf("int: %lu bytes\n", sizeof(int));

    printf("short int: %lu bytes\n", sizeof(short int));

    // inclua outros tipos de dados

    return 0;

}
```

Declaração de variáveis

```
int a;                  // declara a como uma variável inteira
int b, c;                     // declara b e c como variáveis inteiras
int x = 0;                    // declara x como inteira e inicia-a com o valor zero
float d;                      // declara d como uma variável ponto flutuante

a = 5;
b = c = 10;
d = 3.1415;                   // conversão implícita (pode gerar advertência)
d = 3.1415f;            // conversão explícita
float e = 3.1415f;      // boa prática de programação
double f = 1.45e-2      // f armazenará o valor 1.45 x 10^-2

int y = 3.14;                 // a variável y armazenará apenas a parte inteira

int v, w, u = 9;
w = u + v;                    // a variável v não foi inicializada, contém lixo
```

Sintaxe de operadores

```
int a = 5;
int b = c = 7;                  // como isso funciona?

i = i + 2;
i += 2;                         // equivale ao anterior

<var>  <op>=  <expr>;           // forma genérica descrita na gramática
x *= y + 1                      // equivale a x = x * (y + 1);

a = 3;
b = a++ * 2;
c = ++a * 2;

// nesse caso, qual a diferenca entre usar a++ e ++a?
// use printf() para descobrir

a = b = c = 0;
printf("a=%d\tb=%d\tc=%d\n", a, b, c);
b = a++ * 2;                                    // equivale a fazer: b = a * 2; a = a + 1;
printf("a=%d\tb=%d\tc=%d\n", a, b, c);
c = ++a * 2;                                    // equivale a fazer: a = a + 1; b = a * 2;
printf("a=%d\tb=%d\tc=%d\n", a, b, c);
```

Conceito de variável

LP's imperativas são abstrações da arquitetura de Von Neumann

Uma variável de um programa é uma abstração para uma célula ou para um conjunto de células de memória do computador

Programadores imaginam uma variável como um nome para uma posição de memória, o que é suficiente em uma análise mais simples

No entanto, há muito mais coisas associadas a uma variável do que apenas o seu nome e é importante conhecer também esses detalhes

Uma variável possui os seguintes atributos:

- Nome
- Endereço
- Valor
- Tipo
- Tempo de vida
- Escopo

Nomes ou identificadores

Um nome é uma *string* usada para identificar alguma entidade de um programa

Os nomes (ou identificadores) estão associados não apenas às variáveis, mas também a rótulos, subprogramas, etc.

Ao projetar uma LP, surgem algumas questões:

- Qual o tamanho máximo de um nome?
- Caracteres de conexão podem ser usados?
- A linguagem é *case sensitive*?
- Palavras especiais são palavras-chave ou palavras reservadas?

Tamanho dos nomes

As primeiras LP's usavam nomes de apenas um caractere, herança da matemática

Algumas LP's não limitam o tamanho, mas consideram apenas uma parte do nome como significativo

A maioria dos implementadores de ferramentas impõe limites, o que impõe o conhecimento desses limites

Trade-offs relacionados:

- Legibilidade x facilidade de escrita
- Simplicidade x expressividade

```
// veja esse trecho de código em C/C++/Java/C#

for (int indice=0; indice<maximo; indice=indice+1)
{
    vetor[indice] = indice * indice;
}
```

```
// compare com esse trecho a seguir

for (int i = 0; i < max; i += 1) {
    vetor[i] = i * i;
}
```

```
// qual dos dois é mais legível?

// mais simples? mais fácil de escrever?
```

Formato comum para nomes

A forma de nomes mais aceitável na maioria das LP's é:

- Limite de tamanho razoavelmente longo
- Algum caractere de conexão permitido, como por exemplo o underline "_"
- Normalmente o caractere de conexão é aceito apenas como substituto do espaço em branco, não podendo ser um caractere finalizador da string
- Preferencialmente, use a notação camelCase
  - Exs: `int totalAlunos; char nomeCliente[50]; class PessoaFisica`

A maioria das LP's atuais é *case sensitive*, ou seja, faz distinção entre maiúsculas e minúsculas em identificadores

- **`NOME, Nome, NoMe, nome`** podem representar variáveis diferentes
- O mesmo se aplica a nomes de métodos, por exemplo, qual o nome do método em Java que converte string em inteiro?
  `parseint(), ParseInt(), parseInt(), Parseint()`

Palavras especiais

São usadas para dar nomes a ações que devem ser executadas, tornando os programas mais legíveis

Também são usadas para separar as entidades sintáticas dos programas

Na maioria das LP’s, são classificadas como palavras reservadas, mas em algumas são somente palavras-chave

Uma palavra reservada não pode ser usada como um nome, uma palavra-chave pode

É melhor que uma LP possua palavras reservadas do que palavras-chave, porque a possibilidade de redefini-las pode acarretar problemas de legibilidade

Palavras especiais em C

```c
int main(void) {
    int printf;

    printf = 8;
    printf = 4 * printf;

    return 0;
}
```

```c
#include <stdio.h>

int main(void) {
    int printf;

    printf = 8;
    printf = 4 * printf;

    //printf("Valor: %d\n", printf);

    return 0;
}
```

Palavras reservadas em C

A linguagem C possui 32 palavras reservadas, conforme o padrão ANSI:

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

Obs: Alguns compiladores incluem outras palavras reservadas como, asm, cdecl, far, fortran, huge, interrupt, near, pascal, typeof.

Endereço

Apesar do endereço de uma variável ser a posição de memória à qual ela está associada, alguns problemas podem surgir

Suponha que um programa tem dois subprogramas. Cada subprograma utiliza uma variável local chamada AUX. Uma referência a variável AUX poderia causar confusão

Chamadas recursivas a um mesmo subprograma possuem o mesmo problema

Variáveis com mesmo nome

```
#include <stdio.h>
void troca(int*, int*);

int main(void) {
    int a = 3;
    int b = 4;
    int aux = 5;
    printf("A = %d\tB = %d\tAux = %d\n", a, b, aux);
    troca(&a, &b);
    printf("A = %d\tB = %d\tAux = %d\n", a, b, aux);
    return 0;
}

void troca(int *x, int *y) {
    int aux;
    aux = *x;
    *x = *y;
    *y = aux;
}
```

Aliases (apelidos)

Algumas LP’s permitem que múltiplos identificadores façam referência ao mesmo endereço

Podem ser implementados também através de estruturas de registro variantes (Pascal, Ada) ou através do tipo união (C, C++)

Pode também ser implementado através de apontadores, na maioria das LP’s que suportam esse recurso (C, C++, Pascal)

Pode ainda ser implementado através de referências em linguagens mais modernas que não permitem o uso explícito de apontadores (Python)

Exemplos de aliases em Python e em C

```python
lista1 = ['a', 'b', 'c']
lista2 = lista1
lista1.append('d')
lista2.append('e')
print('Lista 1', lista1)
print('Lista 2', lista2)
```

```c
#include <stdio.h>

int main(void) {
    int a;
    int *pa;

    a = 8;
    pa = &a;
    printf("Valor de a: %d\n", a);
    printf("Endereço de a: %p\n", pa);
    a += 1;
    printf("Valor de a: %d\n", a);
    *pa += 1;
    printf("Valor de a: %d\n", a);

    return 0;
}
```

https://repl.it/@flaviusgorgonio/apelidospy

https://repl.it/@flaviusgorgonio/apelidosc

Tipo e valor de uma variável

O tipo da variável define não apenas a faixa de valores que a variável pode receber, mas também o conjunto de operações definidas para aquela variável

O valor de uma variável é o conteúdo da(s) célula(s) de memória associada(s) àquela variável

Essa abstração evita preocupações com o tamanho físico, em bytes, da variável

Ex.:

```
int a;
```

Dependendo da máquina, essa variável pode ocupar 2 ou 4 bytes, mas o programador não precisa se preocupar com isso

Conversão de tipos em C

Conversão automática de tipos em expressões:

```
float raio, area, perim, pi = 3.1415f;
printf("raio: ");
scanf("%f", &raio);
area = pi * raio * raio;
perim = 2 * pi * raio;
```

Conversão explícita (*cast*)

```
printf("raio = %fm\n", raio);
printf("area = %fm2\n", area);
printf("perim = %fm\n", perim);
printf("area inteira: %dm2\n", area);  // ira produzir uma saida incorreta
areaInt = (int) area;
printf("area inteira: %dm2\n", areaInt);
```

Vinculação

Vinculação é uma associação, como por exemplo, entre um atributo e uma entrada ou entre um operador e um símbolo

Tempo de vinculação é o momento em que uma vinculação ocorre

- ➢ Uma vinculação é estática se ocorrer antes do tempo de execução e permanecer inalterada ao longo da execução do programa
- ➢ Uma vinculação é dinâmica se ocorrer durante a execução ou puder ser modificada no decorrer da execução de um programa

Vinculação e tempo de vinculação são conceitos associados à semântica da linguagem

Tempo de vinculação

A vinculação pode ocorrer:

- Em tempo de projeto da linguagem
  - Um símbolo é vinculado a uma operação aritmética (p.ex. '*' significa a multiplicação)
- Em tempo de implementação da linguagem
  - Um tipo de dados (p.ex. int) é vinculado a uma variedade de valores possível
- Em tempo de compilação
  - Uma variável é vinculada a um tipo de dados (p.ex. float x)
- Em tempo de ligação
  - Uma chamada a um subprograma (biblioteca) é vinculada ao código do subprograma
- Em tempo de carregamento
  - Uma variável é vinculada a uma célula de memória (p.ex. no endereço 0x7ffd1eb54964)
- Em tempo de execução
  - Um valor é vinculado a uma variável (p.ex. a = 10)

Bibliografia recomendada

CELES FILHO, W. *et al*. Introdução a estruturas de dados. Rio de Janeiro: Elsevier, 2004.

HUNT, A.; THOMAS, D. O programador pragmático: de aprendiz a mestre. Porto Alegre: Bookman, 2010.

KERNIGHAN, B. W.; PIKE, R. The practice of programming. Boston: Addison-Wesley, c1999.

SEBESTA, R. W. Conceitos de linguagens de programação. Porto Alegre: Bookman, 2011.

VAREJÃO, F. M. Linguagens de programação: conceitos e técnicas. Rio de Janeiro: Elsevier, 2004.