Revisão Linguagem C/C++

Clodoaldo A. M. Lima
cmoraeslima@com.br

Sumário

- História da linguagem C
- Características da linguagem C
- Tipos
- Variáveis e operadores
- Estruturas de controle
- Funções
- Entrada e Saída
- Vetores e Matrizes
- Ponteiros

Linguagem de Programação

- Um *programa* de computador é um conjunto de *instruções* que representam um *algoritmo* para a resolução de algum problema. Estas instruções são *escritas* através de um conjunto de *códigos* (símbolos e palavras).

- Este conjunto de códigos possui regras de estruturação lógica e sintática própria. Diz-se que este conjunto de símbolos e regras formam uma *linguagem de programação*.

L.P.: Exemplos de Códigos: Assembly

Pseudo-código

```
leia(num)
para n de 1
 até 10 passo 1
      faça
    tab←num*n
    imprima(tab)
  fim-para;
```

Assembly (Intel 8088)

```
MOV CX,0
IN  AX,PORTA
MOV DX,AX
LABEL:
INC CX
MOV AX,DX
MUL CX
CMP CX,10
JNE LABEL
OUT AX, PORTA
```

L.P.: Exemplos de Códigos: C

Pseudo-código

```
leia(num)
para n de 1
 até 10 passo 1
      faça
    tab←num*n
    imprima(tab)
  fim-para;
```

C

```
scanf(&num);
for(n=1;n<=10;n+
  +){
   tab=num*n;
   printf(”\n %d”,
    tab);
  };
```

Tipos de Linguagens: Baixo Nível

Baixo-Nível: São linguagens voltadas para a máquina, isto é, são escritas usando-se as instruções do microprocessador do computador. São genericamente chamadas de linguagens *Assembly* ou de montagem.

- *Vantagens*: Os programas são executados com maior *velocidade* de processamento e ocupam menor *espaço* na memória.
- *Desvantagens*: Em geral, programas em Assembly têm pouca *portabilidade*, isto é, um código gerado para um tipo de processador não serve para outro. Códigos Assembly não são estruturados, tornando a *programação* bem mais difícil.

Tipos de Linguagens: Alto Nível

Alto-Nível: São linguagens voltadas para o ser humano. Em geral utilizam sintaxe estruturada tornando seu código mais legível. Necessitam de *compiladores* ou *interpretadores* para gerar as instruções do microprocessador.

- *Vantagens*: Por serem compiladas ou interpretadas, têm *maior portabilidade* podendo ser executados em várias plataformas com pouquíssimas modificações. Em geral, a programação torna-se mais fácil por causa do maior ou menor grau de estruturação de suas linguagens.
- *Desvantagens*: Em geral, as rotinas geradas (em linguagem de máquina) são mais genéricas e portanto mais complexas e por isso são mais lentas e ocupam mais memória.

Linguagens de A. N. Quanto a Aplicação

As linguagens de alto nível podem se distinguir ainda quanto a sua aplicação:

- *Genéricas:* como **C**, **Pascal** e **Basic;**

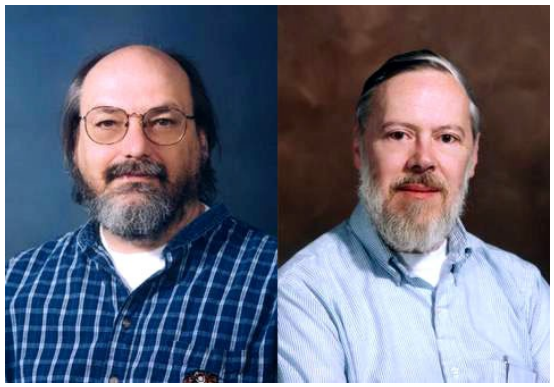
- *Específicas:* como **Fortran** (cálculo matemático), **GPSS** (simulação), **LISP** (inteligência artificial) ou **CLIPPER** (banco de dados).

A Linguagem C: Características

- **C** é uma linguagem de alto nível com uma sintaxe bastante estruturada e flexível tornando sua programação bastante simplificada.
- Programas em **C** são compilados, gerando programas executáveis.
- **C** compartilha recursos tanto de alto quanto de baixo nível, pois permite acesso e programação direta do microprocessador. Com isto, rotinas cuja dependência do tempo é crítica, podem ser facilmente implementadas usando instruções em Assembly. Por esta razão o **C** é a linguagem preferida dos programadores de aplicativos.

História da Linguagem C

- Surgiu no início dos anos 70
- Criada inicialmente para o UNIX
- Criadores:
  - Dennis Ritchie (direita)
  - Kenneth Thompson (esquerda)
- Baseada na Linguagem B
- Versão inicial bastante simples



Linguagem C: Histórico-2/2

- **BCPL** é uma linguagem de programação, criada por Martin Richards, da Universidade de Cambridge em 1966.
- **1970:** *Ken Thompson e Denis Ritchie* desenha uma linguagem a partir do **BCPL** nos laboratórios da *Bell Telephones, Inc*.

  Chama a linguagem de **B**.
- **1978:** *Brian Kerningham* junta-se a *Ritchie* para aprimorar a linguagem. A nova versão chama-se **C**. Pelas suas características de portabilidade e estruturação já se torna popular entre os programadores.
- **~1980:** A linguagem é padronizada pelo *American National Standard Institute*: surge o **ANSI C**.

Linguagem C: Histórico-2/2

- **~1990:** A *Borland International Co*, fabricante de compiladores profissionais escolhe o **C** e o **Pascal** como linguagens de trabalho para o seu *Integrated Development Enviroment* (Ambiente Integrado de Desenvolvimento): surge o **Turbo Pascal e o Turbo C** para DOS.
- **~1992:** C se torna ponto de concordância entre teóricos do desenvolvimento da teoria de *Object Oriented Programming* (programação orientada a objetos): surge então o **C++**.

Linguagem C

- Ampla popularização nos anos 80
- Muitas arquiteturas e compiladores
- Problemas com a incompatibilidade
- Padronização de 82 a 89 (C ANSI)
- Até hoje existem problemas entre os diversos compiladores e sistemas operacionais

Características

- Paradigma Procedural
- Flexível
- Alta performance
- Poucas restrições
- Ótima iteração com:
  - Sistemas Operacionais
  - Dispositivos de Hardware
  - Outras Linguagens

Estrutura dos programas em C

- Um cabeçalho contendo as **diretivas de compilador** onde se definem o valor de constantes simbólicas, declaração de variáveis, inclusão de bibliotecas, declaração de rotinas, etc.
- A diretiva `#include` diz ao compilador para incluir na compilação do programa outros arquivos. Geralmente estes arquivos contem bibliotecas de funções ou rotinas do usuário.
- A diretiva `#define` diz ao compilador quais são as constantes simbólicas usadas no programa.
- Um bloco de instruções **principal** e outros blocos de **rotinas**.

Normas Gerais

- **Formação de identificadores**: Ao contrário de outras linguagens, C faz distinção na capitalização dos identificadores de variáveis usados em um programa, i.e., os identificadores *soma*, *Soma* e *SOMA* são distintos para o compilador C.
- **Declaração de variáveis:** as variáveis devem ser declaradas no inicio do programa. Estas variáveis podem ser de vários tipos: `char` (literal/*string*), `int` (inteiro), `float` (real de simples precisão)
- **Comentários**: podem ser escritos em *qualquer lugar* do texto para facilitar a interpretação do algoritmo. Para que o comentário seja identificado como tal, ele deve ter um `/*` antes e um `*/` depois.
- Exemplo: `/* esta e´ uma linha de comentário em C */`
- `// este e´ um comentário valido apenas em C++`

C: Normas Gerais (cont.)

- **Entrada e saída de dados**: existem várias maneiras de fazer a leitura e escrita de informações. No exemplo apresentado, `printf` a seguir é uma função de escrita na tela e `scanf` é uma função de leitura de teclado.
- **Estruturas de controle**: A linguagem C permite uma ampla variedade de estruturas de controle de fluxo de processamento. Estas estruturas serão vistas a seguir

Exemplo: Um Programa em C-1/3

```
/* ***************************************************************
Programa: e0101.cpp
Proposito: Calcula a raiz quadrada de um numero real positivo
maior que 1.0 com precisao PREC (0.00001).
Ultima Revisao: 01/03/2013
*************************************************************** */
#define MAX 100       // numero maximo de iteracoes
#define PREC 0.000001    // precisao da raiz
void main(){          // inicia programa principal...
        float num     // numero do qual se quer saber a raiz quadrada
        float raiz;         // aproximacao para raiz de num
        float inf, sup;     // intervalo que contem a raiz procurada
```

Exemplo: Um Programa em C-2/3

```
do{  printf("\n\nDigite um numero real positivo: ");
     scanf("%f",&num;} while (num <= 1.0);   // aceita somente num >1.0
     inf = 0.0;            // inicializa intervalo inicial de busca
     sup = num;    i = 0;     // inicializa contador
     do{   // faca...
          i = i + 1;                 // incrementa contador
          raiz = 0.5 * (inf + sup);  // faz estimativa de raiz
          if(raiz*raiz > num){       // se chute foi alto...
               sup = raiz;           // baixa limite superior
          }else{                     // ...senao...
               inf = raiz; };        // sobe limite inferior
     } while( (sup-inf) > PREC && i < MAX);  // enquanto intervalo gde
```

Exemplo: Um Programa em C-3/3

```
        raiz = 0.5 * (inf + sup);              // estima a raiz
        printf("Raiz: %f +- %f",raiz,PREC);    // imprime o valor da raiz
};  // fim do programa1
```

Observações: Note-se que os comentários têm duas apresentações, a saber:

```
/* esta e´ uma linha de comentário em C */
// este e´ um comentário valido apenas em C++
```

Palavras Reservadas

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.: C é case sensitive

Variáveis

- Declaração:
  - *tipo nome = inicialização;*

- Escopo da variáveis:
  - globais: podem ser usadas em qualquer lugar do programa
  - locais: podem ser usadas apenas na função onde foi declarada

Tipos

- São as formas que utilizamos para representar dados
- C possui 5 tipos básicos:
  - **`char, int, float, double e void`**
- E 4 modificadores básicos:
  - **`signed, unsigned, long e short`**
  - Os 4 podem ser aplicados ao **`int`**
  - **`long`** pode ser aplicado ao **`double`**
  - **`signed`** e **`unsigned`** aplicados ao **`char`**

Tipos

- O tamanho do inteiro depende da arquitetura do sistema:
  - Sistemas de 32 bits ⇨ inteiro de 32 bits
  - Sistemas de 64 bits ⇨ inteiro de 64 bits

- Restrições:
  - **`short int`** e **`int`** devem ter pelo menos 16 bits
  - **`long int`** com no mínimo 32 bits
  - **`short int`** <= **`int`** <= **`long int`**

Tipos

- Modificadores de acesso:
  - **`const`**: a variável não pode ter o seu valor alterado
  - **`volatile`**: a variável pode ter o seu valor modificado fora do controle do programa
    - Por exemplo, um endereço de uma variável global pode ser passado para a rotina de relógio do sistema operacional e usado para guardar o tempo real do sistema,
  - Classes de Armazenamento:
  - **`auto`**: indica que uma variável é local (opcional), também é usada na declaração de *nested functions*
  - **`extern`**: variável declarada em outro arquivo
  - **`register`**: armazena, se possível, a variável em um registrador na própria CPU.

Tipos

- Arquivo 1
  - int x, y
  - char ch;
  - main(void)
    - {….}
  - Func1{
    - X=123
  - }

- Arquivo 2
  - extern int x, y
  - extern char ch;
  - Func22{
    - x=y/10;
  - }
  - Func22{
    - y = 10;
  - }

Tipos

- Classes de Armazenamento (Cont.):
  - **`static`**: não permite que um módulo externo possa alterar nem ver uma variável global, também é usado para manter o valor de uma variável local em uma função de uma chamada para outra.

```
series(void)
{
    static int serie num = 100 ;
    series_num = series_num + 23;
    return (series_num)
    }
```

Tipos

- static int series_num;
- void series_start(int seed);
- Int series (void);
- series(void)
  - {
    - series_num = series_num + 23;
    - return (series_num)
    - }
  - void series_start(int seed)
  - {
    - series_num = seed;
  - }

Exemplo

```
int a, b = 10; // Variáveis globais

void f(char c) {
    double d = 10.0; // Variável local
    int i = a; // Variável local
    // ...
}

int main() {
    int i = b;  // Variável local
    return 0;
}
```

```
int main() {
        register int i; // i pode não ser alocada em um registrador
        for (i = 0; i < 10; i = i + 1) {
                // código
        }

        return 0;
}
```

```
#include<stdio.h>

// Evita que outros módulos possam modificar e ver esta variável
static int numero = 0;

int f() {
        // Declara a como uma variável statica
        // assim o valor da no final de uma chamada da funcão
        // será o valor inicial de a na próxima chamada
        static int a = 0;

        a = a + 2;
        return a;
}
```

Exemplos

```
int somarQuadrado(int a, int b) {
    auto int quadrado(int); // protótipo da função

    // agora a função auto pode ser chamada antes de ser implementada
    int retorno = quadrado(a) + quadrado(b);

    int quadrado(int z) {
        return z * z;
    }

    return retorno;
}

int main() {
    printf("2^2 + 10^2 = %d\n", somarQuadrado(2,10));
    return 0;
}
```

Variáveis

- Restrições
  - O nome das variáveis deve começar com uma letra ou um sublinhado “_”
  - Os demais caracteres podem ser letras, números ou sublinhado
  - O nome da variável não pode ser igual a uma palavra reservada e aos nomes das funções
  - Tamanho máximo para o nome de uma variável:
    - 32 caracteres

Constantes

- São valores que são mantidos fixos pelo compilador
- Também podem ser:
  - Octais - 0NUMERO_OCTAL
  - Hexadecimais - 0xNUMERO_HEXADECIMAL
- Exemplos:
  - **‘\n’** (caractere), **“C++”** (string), **10** (inteiro), **15.0** (float), **0xEF** (239 em decimal), **03212** (1674 em decimal)

Operadores Aritméticos

Operador	Ação
+	Soma
-	Subtração ou troca de sinal
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incremento
- -	Decremento

Exercícios

1) Qual o valor das variáveis x, y e z após o seguinte trecho de código:

```
int x, y, z;
x = y = 10;
z = ++x;
x = -x;
y++;
x = x + y - (z--);
```

Operadores Relacionais

Operador	Relação
>	Maior que
>=	Maior que ou igual a
<	Menor que
<=	Menor que ou igual a
==	Igual a
!=	Diferente de

Operadores Lógicos

Operador	Função
&&	AND
||	OR
!	NOT

Operadores Lógicos Bit a Bit

Operador	Ação
&	AND Lógico
|	OR Lógico
^	XOR (OR exclusivo)
~	NOT
>>	Shift Rigth
<<	Shift Left

Exercícios

3) Qual o valor das variáveis a, b, c, d, e, f após a execução do seguinte trecho de código:

```
int x = 2, y = 4;
int a, b, c, d, e, f;
a = x & y;
b = x | y;
c = x ^ y;
d = ~x;
e = x << 3;
f  = x >> 1;
```

Tabela de Precedências

,	= += -= *= /=	?	||	&&	|	^	&	== !=	<< = >> =	<< >>	+ -	* / %	! ~ ++ -- . - * &	( ) [ ] ->

\-

Precedência

\+

Casts (Conversão Explicita de tipo)

- Sintaxe:
  - *(tipo) expressão*

- Exemplo:

```
long a = 10, b = 4;

// f = 2.5

double f = (double) a/b;
```

Controladores de Fluxo

- C possui 7 controladores de fluxo básicos:
  - `if`
  - `?:`
  - `switch`
  - `for`
  - `while`
  - `do-while`
  - `goto`

if

```
if (condição) { declaração }
```

```
if (condição) { declaração1 }
else { declaração2 }
```

```
if (condição1) { declaração1 }
else if (condição2) { declaração2 }
else { declaração3 }
```

Exemplo

```
int a = 2;
if (a==2) {
  a = 4;
} else if (a==0) {
  a = 0;
}
if (a>0) {
  a = -a;
}
```

?:

- condição ? expressão1 : exepressão2;
- Equivalente a:

```
if (condição) { expressão1 }
else { expressão2 }
```

- O operador ? é restrito, mas pode simplificar expressões grandes
- Uma expressão é diferente de uma declaração

Exemplo

```
int a = 9, b = -4, c;

b = (a > b) ? a : b;

c = (a > b) ? b : a;
```

Estrutura de Controle: Repetição (while)

- Pseudo-linguagem

```
enquanto condição
   faça bloco
fim-enquanto;
```

- Note-se que enquanto `condição` for *true* (ou `1`, ou não zero) o laço será feito, *só findando* quando `condição` for `false` (ou zero).

- C

```
while(condição){
    bloco;
};
```

Funções

- Sintaxe:

tipoDeRetorno
nomeDaFunção(declaraçãoDosParâmetros)
{
corpoDaFunção;
}

declaraçãoDosParâmetros = tipo1 nome1,..., tipoN nomeN

- Funções que não possuem retorno são **`void`**
- O retorno de uma função é feito através do comando **`return`**

Exemplos

```
int dobro(int a) {
  return 2 * a;
}

int main(int a) {
  int c;

  c = 8;
  c = dobro(c);
  return 0;
}
```

Entrada e Saída de Dados

- A função scanf é utilizada para a leitura dados do dispositivo de entrada padrão

- A função printf é utilizada para a escrita de dados do dispositivo de saída padrão

Exemplos

```
int a;
scanf(“%d”, &a); // lê a
printf(“%d”, a); // escreve a

char ch;
scanf(“%c”, &ch); // lê c
printf(“%c”, ch); // escreve c

float num;
scanf(“%f”, &num); // lê f
printf(“%f”, num); // escreve f
```

Obs.: Cuidado para não esquecer do & no uso da função scanf

Tabela de Formato para E/S

Tipo	Formato para escrita e leitura
[signed | unsigned] char	%c
[signed] int	%i ou %d
unsigned int	%u
[signed] short int	%hi
unsigned short int	%hu
[signed] long int	%li
unsigned long int	%lu
float	%f
double	%lf
long double	%Lf