1. Fundamentos da Linguagem C

Neste capitulo serão vistos os fundamentos da linguagem C. O conceito de linguagem de programação, linguagens de alto e baixo nível, linguagens genéricas e especificas. Será visto um pouco do histórico da criação da linguagem e a descrição das características mais importantes da linguagem C. Finalmente, será visto o aspecto geral de um código fonte escrito em C.

1.1 Linguagens de Programação

Um *programa* de computador é um conjunto *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. Dizemos que este conjunto de símbolos e regras formam uma *linguagem de programação*.

1.1.1 Exemplos de códigos.

Existem muitas linguagens de programação. Podemos escrever um algoritmo para resolução de um problema por intermédio de qualquer linguagem. A seguir mostramos alguns exemplos de trechos de códigos escritos em algumas linguagens de programação.

Exemplo: trecho de um algoritmo escrito em **Pseudo-linguagem** que recebe um número `num` e escreve a tabuada de 1 a 10 para este valor:

```
leia num
para n de 1 até 10 passo 1 faça
  tab ← num * n
  imprime tab
fim faça
```

Exemplo: trecho do mesmo programa escrito em **linguagem C**:

```
scanf(&num);
```

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

Exemplo: trecho do mesmo programa escrito em **linguagem Basic**:

```
10 input num
20 for n = 1 to 10 step 1
30 let tab = num * n
40 print chr$ (tab)
50 next n
```

Exemplo: trecho do mesmo programa escrito em **linguagem Fortran**:

```
   read (num);
   do 1 n = 1:10
     tab = num * n
     write(tab)
10 continue
```

Exemplo: trecho do mesmo programa escrito em **linguagem Assembly** para INTEL 8088:

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

1.1.2 Linguagens de baixo e alto nível.

Podemos dividir, genericamente, as linguagens de programação em dois grandes grupos: as linguagens de *baixo nível* e as de *alto nível*:

Linguagens de baixo nível: São linguagens voltadas para a máquina, isto é, são escritas usando as instruções do microprocessador do computador. São genericamente chamadas de linguagens *Assembly*.

Vantagens: Programas são executados com maior *velocidade* de processamento. Os programas ocupam menos *espaço* na memória.

Desvantagens: Em geral, programas em Assembly tem pouca *portabilidade*, isto é, um código gerado para um tipo de processador não serve para outro. Códigos Assembly não são estruturados, tornando a *programação* mais difícil.

Linguagens de 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 instruções do microprocessador. Interpretadores fazem a interpretação de *cada* instrução do programa fonte executando-a dentro de um ambiente de programação, **Basic** e **AutoLISP** por exemplo. Compiladores fazem a tradução de *todas* as instruções do programa fonte gerando um programa executável. Estes programas executáveis (`*.exe`) podem ser executados fora dos ambientes de programação, **C** e **Pascal** por exemplo. As linguagens de alto nível podem se distinguir quanto a sua aplicação em *genéricas* como **C**, **Pascal** e **Basic** ou *específicas* como **Fortran** (cálculo matemático), **GPSS** (simulação), **LISP** (inteligência artificial) ou **CLIPPER** (banco de dados).

Vantagens: Por serem compiladas ou interpretadas, tem *maior portabilidade* podendo ser executados em varias 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 maquina) são mais genéricas e portanto mais complexas e por isso são mais lentas e ocupam mais memória.

1.2 Linguagem C

A **linguagem C** é uma linguagem de *alto nível, genérica.* Foi desenvolvida *por* programadores *para* programadores tendo como meta características de flexibilidade e portabilidade. O C é uma linguagem que nasceu juntamente com o advento da teoria de *linguagem estruturada* e do *computador pessoal*. Assim tornou-se rapidamente uma linguagem "popular" entre os programadores. O C foi usado

para desenvolver o sistema operacional **UNIX**, e hoje esta sendo usada para desenvolver novas linguagens, entre elas a linguagem **C++** e **Java**.

1.2.1 Características do C

Entre as principais características do C, podemos citar:

- O **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.
- O **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.
- O **C** é uma linguagem estruturalmente simples e de grande portabilidade. O compilador C gera códigos mais enxutos e velozes do que muitas outras linguagens.
- Embora estruturalmente simples (poucas funções intrínsecas) o **C** não perde funcionalidade pois permite a inclusão de uma farta quantidade de rotinas do usuário. Os fabricantes de compiladores fornecem uma ampla variedade de rotinas pré-compiladas em bibliotecas.

1.2.2 Histórico

1970: *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**.

~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 C**.

~1992: O C se torna ponto de concordância entre teóricos do desenvolvimento da teoria de *Object Oriented Programming* (programação orientada a objetos): surge o **C++**.

1.3 Estrutura de um programa em C

Um programa em **C** é constituído de:

- 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.
- um bloco de instruções **principal** e outros blocos de **rotinas**.
- documentação do programa: comentários.

Programa Exemplo: O arquivo `e0101.cpp` contém um programa para calcular a raiz quadrada de um número real positivo:

1.3.1 Conjunto de caracteres

Um programa fonte em C é um **texto não formatado** escrito em um editor de textos usando um o conjunto padrão de caracteres ASCII. A seguir estão os caracteres utilizados em C:

Caracteres válidos:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
1 2 3 4 5 6 7 8 9 0
+ - * / \ = | & ! ? # % ( ) { } [ ] _ ‘ “ . , : < >
```

Caracteres não válidos:

```
@ $ □ á é õ ç
```

Os caracteres acima são válidos apenas em strings. **Veja seção 2.1.4.**

1.3.2 Comentários

Em C, 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. Observe que no exemplo `e0101.cpp` todo o cabeçalho esta dentro de um comentário.

Exemplo:

```
/* esta e´ uma linha de comentário em C */
```

Observação: O C++ permite que comentários sejam escritos de outra forma: colocando um `//` em uma linha, o compilador entenderá que tudo que estiver a direita do símbolo é um comentário. Observe no programa exemplo `e0101.cpp` as linhas de comentários colocadas a direita dos comandos.

Exemplo:

```
// este e´ um comentário valido apenas em C++
```

1.3.3 Diretivas de Compilação

Em C, existem comandos que são processados durante a **compilação** do programa. Estes comandos são genericamente chamados de *diretivas de compilação*. Estes comandos informam ao compilador do C basicamente quais são as **constantes simbólicas** usadas no programa e quais **bibliotecas** devem ser anexadas ao programa executável. 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. Voltaremos a trabalhas esta diretiva com mais detalhe no capítulo 5. A diretiva `#define` diz ao compilador quais são as constantes simbólicas usadas no programa. Veremos sobre esta diretiva no capitulo 2.

1.3.4 Declaração de variáveis

Em C, como na maioria das linguagens, as variáveis devem ser declaradas no inicio do programa. Estas variáveis podem ser de vários tipos: `int` (inteiro), `float` (real de simples precisão) e outras que serão vistas no capitulo 2. No exemplo acima `num`, `raiz`, `inf` e `sup` são declaradas como variáveis reais, enquanto `i` é declarada como uma variável inteira.

1.3.5 Entrada e saída de dados

Em C existem varias maneiras de fazer a leitura e escrita de informações. Estas operações são chamadas de operações de entrada e sadia. Veremos no capitulo 3 algumas funções de entrada e sadia de informações via teclado e tela. Outras funções de leitura e escrita em arquivos, sadia gráfica, funções de manipulação de *mouse,* entrada e sadia de informações via portas serial e paralela serão vistas em capítulos posteriores. No exemplo acima `printf` é uma função de escrita na tela, `scanf` é uma função de leitura de teclado.

1.3.6 Estruturas de controle

A linguagem C permite uma ampla variedade de estruturas de controle de fluxo de processamento. Estas estruturas serão vistas em detalhes nos capítulos 4 e 5. Duas estruturas das estruturas básicas (decisão e repetição) são muito semelhantes as estruturas usadas nas Pseudo-linguagem algorítmicas:

Estrutura de Decisão: Permite direcionar o fluxo lógico para dois blocos distintos de instruções conforme uma condição de controle.

Pseudo-linguagem

```
se condição
  então bloco 1
  senão bloco 2
fim se
```

Linguagem C

```
if( condição){
   bloco 1;
}else{
   bloco 2;
};
```

Estrutura de Repetição: Permite executar repetidamente um bloco de instruções ate que uma condição de controle seja satisfeita.

Pseudo-linguagem

```
faça
  bloco
até condição
```

Linguagem C

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