1. INTRODUÇÃO

A linguagem de programação C é uma linguagem de finalidade geral. Foi desenvolvida por programadores para programadores tendo como meta características de flexibilidade e portabilidade, pois não é "atada" a um sistema operacional ou a uma máquina particular. 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. C é estreitamente associada ao sistema operacional UNIX, já que foi usada para desenvolvê-lo. Hoje, também está sendo usada desenvolver novas linguagens, entre elas a linguagem C e Java [KER 88].

1.1. Definições Básicas

Existem dois tipos fundamentais de tradutores: interpretadores e compiladores. No caso de um interpretador, as instruções definidas na linguagem de alto nível são executadas diretamente. Ele traduz um comando de um programa de cada vez e então chama uma rotina para completar a execução do comando, como ilustrado na figura 1.1. Mais precisamente, um interpretador é um programa que executa repetidamente a seguinte seqüência:

- Pega a próxima instrução;
- Determina as ações a serem executadas;
- Executa estas ações [DER 90, GHE 97].

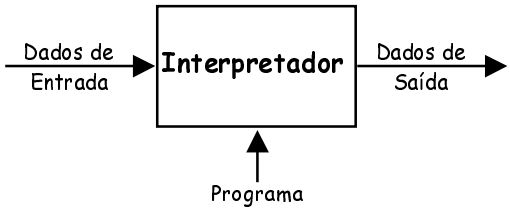

Figura 1.1 – Processo de interpretação [GHE 97]

Já um compilador produz a partir do programa de entrada, outro programa que é equivalente ao original, porém numa linguagem que é executável. Este programa resultante pode ser em uma linguagem que é diretamente executável, tal como linguagem de máquina, ou indiretamente executável, tal como outra linguagem para a qual já existe um tradutor.

Cada um destes processos tem suas vantagens e desvantagens. Interpretação, apesar de ter um tempo maior de execução, tem a vantagem de não traduzir instruções que nunca são executadas e de conseguir voltar à instrução correspondente na linguagem de programação a partir de qualquer ponto da execução. O compilador, por outro lado, precisa traduzir cada instrução somente uma vez, independente de quantas vezes a instrução é executada. Isto aplica-se tanto no caso de iteração como no caso de execuções repetidas do mesmo programa. As vantagens de um compilador em geral superam as do interpretador na prática, o que faz com que esta forma de tradução seja uma das mais usadas. Por esta razão, e porque a compilação é um processo mais complexo, as atividades de um compilador serão detalhadamente descritas [DER 90].

O objetivo de um compilador é traduzir um programa escrito em uma linguagem fonte em um programa equivalente expresso em uma linguagem que executável diretamente pela máquina. Estes dois

programas são chamados **programa fonte** (ou código fonte) e **programa objeto** (ou código objeto). A linguagem do programa objeto é chamada de linguagem *target*. A figura 1.2 mostra uma visão do processo de compilação onde o programa objeto é executado diretamente. O tempo durante o qual o compilador está "trabalhando", isto é, realizando a conversão entre código-fonte e código-objeto, é chamado de tempo de compilação.

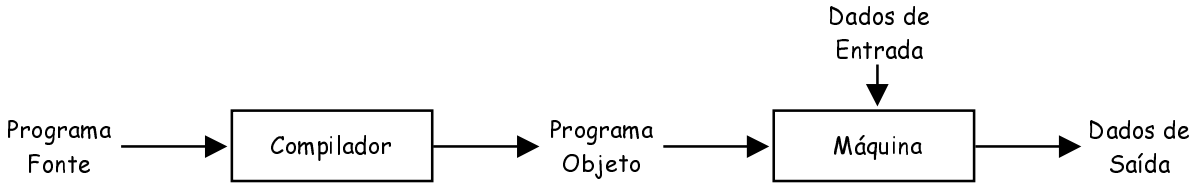

Figura 1.2 - Processo de compilação [DER 90]

Para geração do código executável final (código que pode ser executado pelo sistema operacional), entretanto, existem outros passos além da compilação. A figura 1.3 ilustra o processo de geração do código executável. No primeiro passo, o **pré-processador** mapea instruções escritas numa linguagem de alto nível estendida, para instruções da linguagem de programação original. Entre as funções que ele pode realizar incluem-se: processamento de macros (as evocações a macro-rotinas são traduzidas para o código original da macro); inclusão de arquivos (referências a arquivos são substituídas pelo próprio arquivo); racionalização (substituição de código não oferecido pelo compilador por código equivalente suportado por ele); e extensão da linguagem (suporte a novos aspectos). O **compilador** analisa o código-fonte e o converte este para um código-*assembly* (versão mnemônica da linguagem de máquina). O **montador** traduz código-*assembly* para código-objeto (ou programa objeto), que é a versão em linguagem de máquina do código-fonte. Porém, esta forma é intermediária, não podendo ser lida pelo programador, nem executada pelo computador. A razão da sua existência é porque todos programas em C devem ser "linkados" com rotinas de suporte da biblioteca de execução C. Finalmente, o **ligador** (ou *linkeditor*) "junta" o código objeto com as bibliotecas necessárias para gerar o programa executável. O tempo após a ativação do programa executável é chamado tempo de execução [HAN 86].

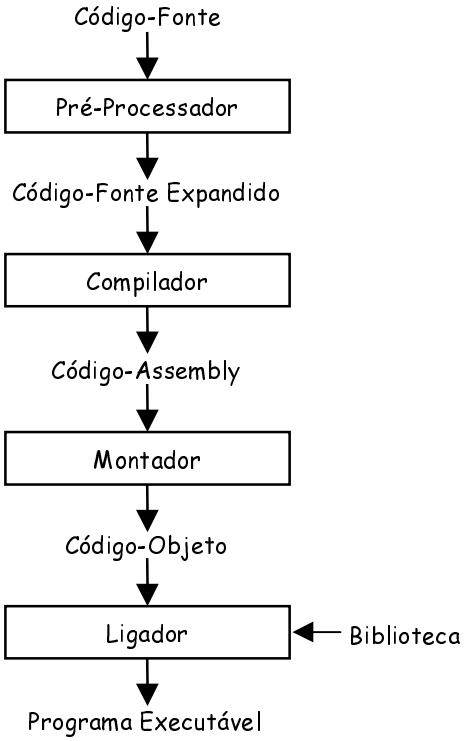

Figura 1.3 - Processo de geração do código executável

1.2. Histórico

Ascendência [HAN 86]:

Algol 60

Ao ponto de vista técnico, o surgimento do C iniciou com a linguagem ALGOL 60, definida em 1960. ALGOL era uma linguagem de alto nível, que foi criada para substituir o FORTRAN. Entretanto, ALGOL não teve sucesso, talvez por tentar ser de muito alto nível em uma época em que a maioria dos sistemas operacionais exigiam do usuário um grande conhecimento de hardware.

CPL (*Combined Programming Language*)

CPL surgiu em 1967 nas universidades de Londres e Cambridge com o objetivo, segundo a equipe do projeto, de "trazer ALGOL à terra", ou "manter contato com a realidade de um computador real". Da mesma forma de ALGOL, CPL não foi bem aceita, em especial pelos projetistas de sistemas operacionais que a consideravam difícil de implementar.

BCPL (*Basic Combined Programming Language*)

BCPL foi criada em Cambridge, ainda em 1967, com o objetivo de resolver os problemas de CPL, reduzindo-o às suas melhores funções básicas.

B

Em 1970, Ken Thompson, chefe da equipe que projetou o UNIX para o PDP11 dos Laboratórios Bell, implementou um compilador para uma versão mais simplificada do CPL, a linguagem B. Porém, com as simplificações, BCPL e B acabaram se tornando linguagens muito limitadas, úteis apenas para certos tipos de problemas.

C

Com os problemas das linguagens anteriores, o projetista Dennis Ritchie, dos Laboratórios Bell, foi encarregado de projetar uma nova linguagem, sucessora do B, que viria então, a ser chamada de C. A linguagem C buscou manter o "contato com o computador real" e ainda sim dar ao programador novas condições para o desenvolvimento de programas em áreas diversas, como comercial, científica e de engenharia. Assim, a primeira versão de C foi criada em 1972 para ser incluído como um dos software a serem distribuídos juntamente com o sistema operacional UNIX do computador PDP-11, na equipe certificada por Ken Thompson. Em 1985, ANSI (*American National Standards Institute*) estabeleceu um padrão oficial de C o chamado "C ANSI".

1.3. Características da Linguagem

- Programas em C são compilados, gerando programas executáveis.
- C é uma linguagem estruturalmente simples, e o seu compilador gera códigos mais enxutos e velozes do que muitas outras linguagens.
- C é uma linguagem de relativo "baixo nível", pois combina elementos de linguagens de alto nível com a funcionalidade da linguagem *Assembly*. Isto significa que o C permite a manipulação de *bits*, *bytes* e endereços – os elementos básicos com os quais o computador funciona.

- Código escrito em C é muito portável, o que significa que os programas fonte criados para executar em um tipo de computador podem ser transportados e recompilados em outros sem muitos problemas.
- C não é uma linguagem rica em tipos de dados, mas permite quase todas conversões de tipos (por exemplo, os tipos caractere e inteiro podem ser livremente misturados na maioria das expressões).
- C não provê operações para manipular diretamente objetos compostos tais como cadeias de caracteres, conjuntos, listas, ou arranjos considerados como um todo.
- C não provê facilidades de entrada e saída, isto é, não há comandos *READ* ou *WRITE*, nem métodos de acesso a arquivos. Todos esses mecanismos devem ser fornecidos por funções explicitamente chamadas.
- C é uma linguagem estruturada, isto é, permite a divisão de um programa em módulos, escondendo do resto do programa todas as informações necessárias para se realizar uma tarefa específica.
- C tem como principal componente estrutural a função, que corresponde a um bloco de construção em que toda a atividade do programa ocorre. As funções permitem que as tarefas de um programa sejam definidas e codificadas separadamente (programa modular).
- C é uma linguagem para programadores, isto é, ela foi criada, influenciada e testada por programadores profissionais.
- C 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.
- C oferece somente construções simples de fluxo de controle: testes, laços, agrupamentos e subprogramas, mas não multiprogramação, operações paralelas, sincronização ou co-rotinas [KER 88, SCH 96].

1.4. Comparação com Pascal

Uma grande vantagem da linguagem C em relação a Pascal, é que C não possui tantas restrições quanto Pascal, além de oferecer a velocidade da linguagem *assembly*. Cada programador C pode, de acordo com as suas necessidades, criar e manter uma biblioteca única de funções customizadas, para ser usada em muitos programas diferentes. Por admitir a compilação separada, C permite que os programadores gerenciem facilmente grandes projetos com mínima duplicação de esforço [SCH 96].

No quadro abaixo, pode-se observar algumas características de C e Pascal [SUR 99]:

Linguagens / Características Ideais	Pascal	C
Executáveis Curtos	fraco ☹	bom ☺
Executáveis Rápidos	bom	bom
Portáveis	razoável	bom
Manipulação de *Bits*	razoável	ótimo ☺

Um programa escrito na linguagem de programação C consiste em uma coleção de funções, sendo que a "*main()*" é a primeira função a ser executada. Além disso, C é *case-sensitive*, ou seja, diferencia letras maiúsculas de letras minúsculas (por exemplo, int ≠ Int). Um programa C mínimo consiste em:

```
main ()
{
}
```

Este programa define a função *main*, que não possui argumentos e não faz nada. As chaves, { e }, são usadas para expressar agrupamentos. No caso do exemplo, elas indicam o início e o fim do corpo da função *main*, que está vazia, isto é, não faz nada. Cada programa em C deve ter uma função *main*.

Por outro lado, um programa escrito em Pascal é subdividido em três áreas distintas: cabeçalho do programa (*program <nome>;*), área de declarações (*uses, label, const, type, var, procedure, function*) e corpo do programa (*begin.....end*). Para ilustrar vamos ver a solução em Pascal e em C para o seguinte problema: elaboração de um programa que efetue a leitura de dois valores numéricos, some estes dois valores e apresente o resultado obtido na tela do computador [MAN 96].

Este programa define a função *main*, que não possui argumentos e não faz nada. As chaves, { e }, são usadas para expressar agrupamentos em C; no exemplo anterior, estas indicam o início e o fim do corpo da função (vazia) *main*. Cada programa em C deve ter uma função *main*.

Algoritmo:

1- Ler um valor para a variável A;
2- Ler outro valor para a variável B;
3- Efetuar a adição das variáveis A e B, implicando o seu resultado na variável X;
4- Apresentar o valor da variável X após a operação de adição dos dois valores fornecidos.

Programa completo na linguagem Pascal:

```
program ADICIONA_NUMEROS;
var
        X: integer;
        A: integer;
        B: integer;
begin
        readln(A);
        readln(B);
        x := a + b;
        writeln(X);
end.
```

Programa completo na linguagem C:

```
#include <stdio.h>
main()
{
        int X;
        int A;
        int B;

        scanf("%d", &A);
        scanf("%d", &B);
        X = A + B;
        printf("%d \n", X);
}
```

1.5. Utilização de um Ambiente de Programação

Em princípio, o ambiente de programação que será utilizado é o Turbo C++ da *Borland*, que é *free*. Apesar de ser interface DOS, o editor é de fácil utilização. O único cuidado que se deve ter é com a correta configuração dos diretórios (*Include*, *Library* e *Output*), que pode ser alterada a partir da opção "*Directories*" do menu "*Options*". Um manual de utilização deste compilador está disponível em http://www.inf.pucrs.br/~pinho/LaproI/Tclite/TcLite.htm.