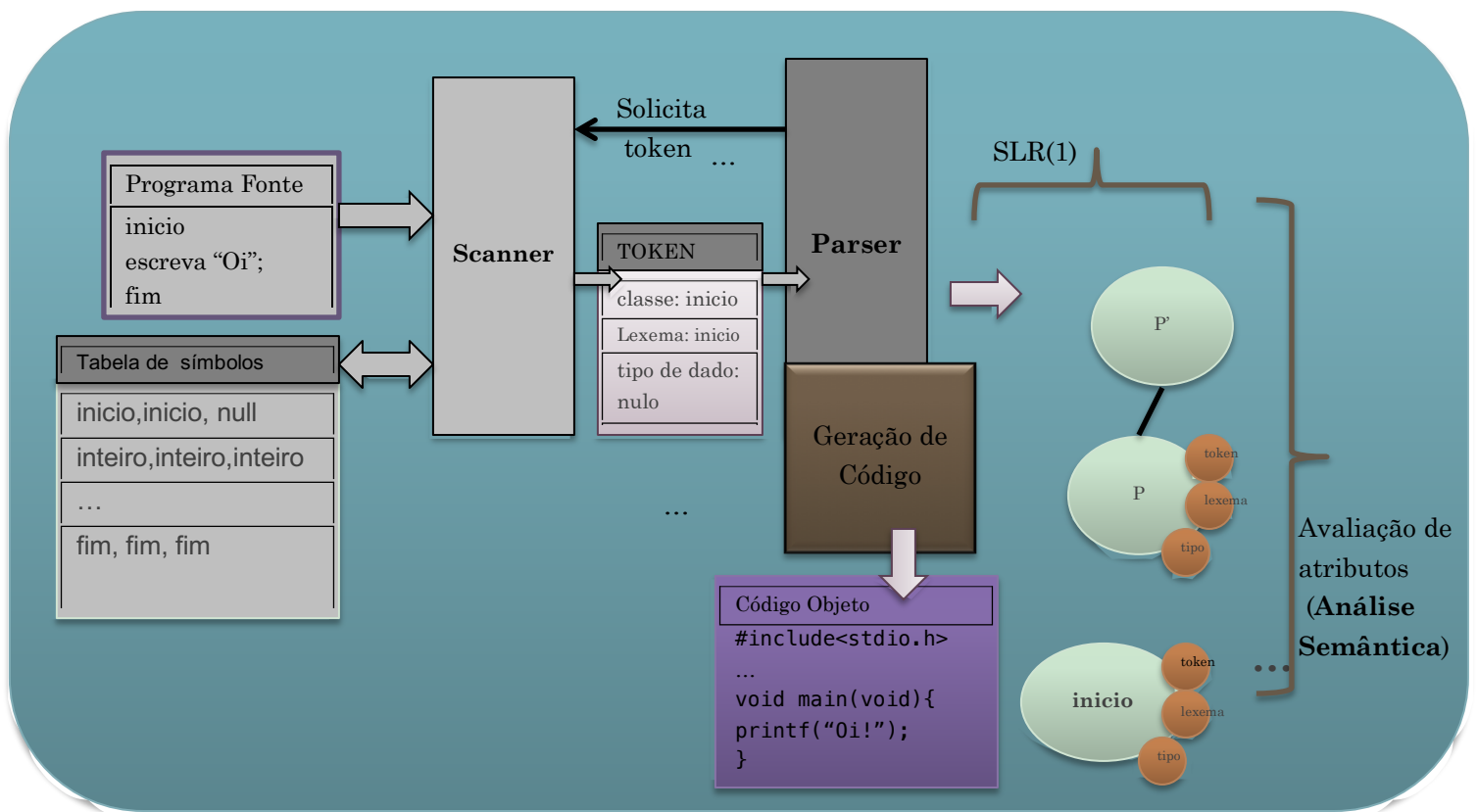


COMPILADORES – TRABALHO 3

Analizador Semântico e tradução dirigida pela sintaxe



1 . Descrição

A atividade prática Trabalho 3 (T3) – Analisador Semântico e Tradução Dirigida pela Sintaxe em Compiladores é um componente para a avaliação e desenvolvimento dos conhecimentos desenvolvidos nas disciplinas ofertadas para Ciência da Computação e Engenharia de Computação - Compiladores e Compiladores 1. O valor dessa atividade é 10,0 e compõe a média de aprovação na disciplina conforme definido no plano de curso.

2. Entregável e Notas

Entregar na data determinada pelo professor, EXCLUSIVAMENTE via plataforma Turing, O CÓDIGO desenvolvido para o analisador semântico com tradução dirigida por sintaxe a ser descrito nas seções abaixo.

- Caso seja realizado em equipe, apenas um componente deverá efetuar a entrega na plataforma. O NOME do código deverá seguir o padrão: ASem-NomeAluno1-NomeAluno2.extensão.

Exemplo: ASem-DeborahFernandes-FulanoPrado.c

- Se for entregar um projeto com vários arquivos, junte-os em uma pasta com o nome ASem-NomeAluno1-NomeAluno2 e inclua dentro da pasta um arquivo .txt explicando como abrir e rodar os códigos do programa. Utilize compressão .zip
- Nota Total da entrega com a avaliação oral = 10,0.

3 - O que fazer?

O programa a ser desenvolvido deverá estar de acordo com as definições de projeto descritas abaixo e será avaliado pelo professor em relação a cada critério estabelecido. Portanto, leia com atenção.

Desenvolver um programa computacional na linguagem escolhida para o projeto que, acoplado ao T1 e ao T2 (analisadores léxico e sintático), implemente o compilador que atenda às solicitações a serem descritas abaixo.

Observe o conjunto de regras semânticas que contemplam a avaliação de atributos, a análise semântica, a geração de código e o tratamento de erros semânticos da TABELA 1.

3.1. O símbolo “ – “ em ações semânticas indica que não há regra semântica associada.

3.2. Imprimir(...) indica que deverá ser realizada uma impressão no arquivo .obj que neste trabalho será PROGRAMA.C (arquivo objeto a ser gerado pelo compilador desenvolvido).

3.3 Emitir mensagem de **ERRO semântico**, indicará a impressão na saída padrão da mensagem do Erro semântico encontrado, seguido da **linha e coluna do código fonte** onde este ocorreu.

3.4 A impressão de linhas brancas (REGRA 5) no .obj indica o local **onde as variáveis deverão ser declaradas**. Pode ser ajustado pelo programador de acordo com sua necessidade.

3.5 Regras semânticas da forma: terminal ou não-terminal.atributo <- terminal ou não-terminal.atributo indicam a ocorrência de amarração de atributos. Aconselha-se o uso de uma pilha semântica para tal.

3.6 Nas marcações (A), (B), (C), (D) e (E) o aluno deverá analisar as **necessidades de regras semânticas associadas às sintáticas e construí-las**. Essas podem envolver **acesso e/ou atualização da tabela de símbolos**, escrita de código no arquivo .obj, emissão de mensagem de erro semântico, **avaliação de atributos**, etc.

Dicas importantes:

(1) Observe as demais regras na tabela para que possa ter uma ideia de como compor as novas e se há necessidade de criá-las.

(2) Nos slides de Análise Semântica disponíveis na TURING você encontrará anotações sobre a avaliação de esquema **L-atribuído em uma análise ascendente** (*bottom-up*).

3.7 Algumas regras utilizam a variável Tx. Esta é uma variável gerada automaticamente para a tradução das operações aritméticas e relacionais do programa fonte para o objeto. Tais variáveis são chamadas de “**variáveis temporárias**” e serão utilizadas e geradas em um **processo de compilação que possua código intermediário**. Para utilizar a variável Tx:

- É necessário desenvolver um contador que inicie de 0 até a quantidade de variáveis adequadas a tradução. Dessa forma, o código objeto possuirá as variáveis T0, T1, T2, ..., necessárias a execução dos comandos.
- A cada variável gerada, é necessário realizar sua declaração no programa obj. Para tal, deve ser desenvolvido um mecanismo que realize a produção dessas variáveis com geração de números sequenciais e sua declaração no programa objeto.

4 – Passos para o desenvolvimento do projeto

4.1. Ler as regras semânticas associadas às sintáticas;

- 4.2. Ajustar o item 3.6;
- 4.3. Implementar as regras semânticas nas reduções das produções sintáticas do trabalho T2;
- 4.4. Criar o arquivo .obj que será PROGRAMA.C;
- 4.5. No arquivo obj PROGRAMA.C a ser gerado, serão necessários ajustes para que a tradução seja completa. O desenvolvedor é responsável por criar rotinas para adicionar cabeçalho com bibliotecas e ajustes finos para que o programa gerado (em linguagem C) funcione perfeitamente em um compilador C.
- 4.6. Saída do Sistema (T3): Impressão das regras sintáticas no prompt e caso não haja erros na fase de análise, gerar o arquivo PROGRAMA.C. Este será a tradução do FONTE.alg (MGOL) para C confeccionada a partir do método tradução dirigida pela sintaxe.
- 4.7. Mais algumas observações:
- As regras semânticas podem ser executadas assim que todas as atividades de uma ação de redução do sintático tenham sido realizadas. No momento após a impressão da regra reduzida, realiza-se uma chamada ao semântico que executará a(s) regra(s) semântica(s) associadas à(s) sintática(s) – (FIGURA 1, linha 12).
 - O sistema continuará identificando erros léxicos e sintáticos. Caso seja encontrado um ou mais erros léxicos, sintáticos e/ou semânticos o programa **continua a análise mas não poderá gerar código .obj.**

FIGURA 1 – Algoritmo de análise sintática ascendente *shift-reduce*.

Algoritmo de análise	
(1)	Seja a o primeiro símbolo de w\$;
(2)	while { /*Repita indefinidamente*/
(3)	seja s o estado no topo da pilha;
(4)	if (ACTION [s,a] = shift t) {
(5)	empilha t na pilha;
(6)	seja a o próximo símbolo da entrada;
(7)	Empilhar o token com seus atributos na pilha semântica
(8)	}else if (ACTION [s,a] = reduce A-> β) {
(9)	desempilha símbolos β da pilha;
(10)	faça o estado t agora ser o topo da pilha;
(11)	empilhe GOTO[t,A] na pilha;
(12)	imprima a produção A-> β ;
(13)	INVOCAR SEMÂNTICO(A-> β) e ajustar a pilha semântica
(14)	}else if (ACTION [s,a] = accept) pare; /* a análise terminou*/
(15)	else chame uma rotina de recuperação do erro ;

TABELA 1 – Regras sintáticas (T2) e regras semânticas (T3).

Identificação	Regra gramatical	Regras Semânticas
1	$P' \rightarrow P$	-
2	$P \rightarrow \text{inicio } V A$	-
3	$V \rightarrow \text{varinicio } LV$	-
4	$LV \rightarrow D LV$	-
5	$LV \rightarrow \text{varfim};$	Imprimir três linhas brancas no arquivo objeto;
6	$D \rightarrow \text{TIPO } L;$	(A) Ajustar a passagem do atributo de TIPO.tipo para o atributo tipo dos id(s) da lista de ids L na TABELA DE SÍMBOLOS.
7	$L \rightarrow \text{id}, L$	(B) Criar regras para: <ul style="list-style-type: none"> Ajustar a passagem do atributo de TIPO.tipo para o id.tipo na TABELA DE SÍMBOLOS. Promover a impressão do(s) lexema(s) dos id(s) no programa .obj na ordem na qual foram declarados no programa fonte. Pode ser realizada durante a ação de empilhamento, não há necessidade de ser apenas durante a redução.
8	$L \rightarrow \text{id}$	(C) Criar regras para: <ul style="list-style-type: none"> Ajustar a passagem do atributo de TIPO.tipo para o id.tipo na TABELA DE SÍMBOLOS. Promover a impressão do(s) lexema(s) dos id(s) no programa .obj na ordem na qual foram declarados no programa fonte. Pode ser realizada durante a ação de empilhamento, não há necessidade de ser apenas durante a redução.
9	$\text{TIPO} \rightarrow \text{int}$	TIPO.tipo \leftarrow inteiro.tipo Imprimir (TIPO.tipo);
10	$\text{TIPO} \rightarrow \text{real}$	TIPO.tipo \leftarrow real.tipo Imprimir (TIPO.tipo);
11	$\text{TIPO} \rightarrow \text{lit}$	TIPO.tipo \leftarrow literal.tipo Imprimir (TIPO.tipo);
12	$A \rightarrow \text{ES } A$	-
13	$\text{ES} \rightarrow \text{leia id};$	Verificar se o campo <i>tipo</i> do identificador está preenchido indicando a declaração do identificador (execução da regra semântica de número 6). Se sim, então: <ul style="list-style-type: none"> Se id.tipo = literal Imprimir (scanf("%s", id.lexema);) Se id.tipo = inteiro Imprimir (scanf("%d", &id.lexema);) Se id.tipo = real Imprimir (scanf("%lf", &id.lexema);) Caso Contrário: Emitir na tela "Erro: Variável não declarada", linha e coluna onde ocorreu o erro no fonte.

14	ES → escreva ARG;	Gerar código para o comando escreva no arquivo objeto. Imprimir (printf ("ARG.lexema");)
15	ARG → literal	ARG.atributos ← literal.atributos (Copiar todos os atributos de literal para os atributos de ARG).
16	ARG → num	ARG.atributos ← num.atributos (Copiar todos os atributos de literal para os atributos de ARG).
17	ARG → id	Verificar se o identificador foi declarado (execução da regra semântica de número 6). Se sim, então: ARG.atributos ← id.atributos (copia todos os atributos de id para os de ARG). Caso Contrário: Emitir na tela "Erro: Variável não declarada" , linha e coluna onde ocorreu o erro no fonte.
18	A → CMD A	-
19	CMD → id rcb LD;	Verificar se id foi declarado (execução da regra semântica de número 6). Se sim, então: Realizar verificação do <i>tipo</i> entre os operandos <i>id</i> e <i>LD</i> (ou seja, se ambos são do mesmo tipo). Se sim, então: Imprimir (id.lexema rcb.tipo LD.lexema) no arquivo objeto. Caso contrário emitir: "Erro: Tipos diferentes para atribuição", linha e coluna onde ocorreu o erro no fonte. Caso contrário emitir "Erro: Variável não declarada" , linha e coluna onde ocorreu o erro no fonte.
20	LD → OPRD opm OPRD	Verificar se tipo dos operandos são equivalentes e diferentes de <i>literal</i> . Se sim, então: Gerar uma variável numérica temporária Tx, em que x é um número gerado sequencialmente. LD.lexema ← Tx Imprimir (Tx = OPRD.lexema opm.tipo OPRD.lexema) no arquivo objeto. Caso contrário emitir "Erro: Operandos com tipos incompatíveis" , linha e coluna onde ocorreu o erro no fonte.
21	LD → OPRD	LD.atributos ← OPRD.atributos (Copiar todos os atributos de OPRD para os atributos de LD).
22	OPRD → id	Verificar se o identificador está declarado. Se sim, então: OPRD.atributos ← id.atributos Caso contrário emitir "Erro: Variável não declarada" , linha e coluna onde ocorreu o erro no fonte.
23	OPRD → num	OPRD.atributos ← num.atributos (Copiar todos os atributos de num para os atributos de OPRD).
24	A → COND A	-
25	COND → CAB CP	Imprimir (}) no arquivo objeto.
26	CAB → se (EXP_R) então	Imprimir (if (EXP_R.lexema) {) no arquivo objeto.

27	EXP_R → OPRD opr OPRD	<p>Verificar se os tipos de dados de OPRD são iguais ou equivalentes para a realização de comparação relacional.</p> <p>Se sim, então:</p> <p>Gerar uma variável booleana temporária Tx, em que x é um número gerado sequencialmente.</p> <p>EXP_R.lexema ← Tx</p> <p>Imprimir (Tx = OPRD.lexema opr.tipo OPRD.lexema) no arquivo objeto.</p> <p>Caso contrário emitir “Erro: Operandos com tipos incompatíveis” , linha e coluna onde ocorreu o erro no fonte.</p>
28	CP → ES CP	-
29	CP → CMD CP	-
30	CP → COND CP	-
31	CP → fimse	-
32	A → R A	-
33	R → facaAte (EXP_R) CP_R	(D) Verificar as necessidades e gerar as regras semânticas e de tradução.
34	CP_R → ES CP_R	(E) Verificar as necessidades e gerar as regras semânticas e de tradução.
35	CP_R → CMD CP_R	(F) Verificar as necessidades e gerar as regras semânticas e de tradução.
36	CP_R → COND CP_R	(G) Verificar as necessidades e gerar as regras semânticas e de tradução.
37	CP_R → fimFaca	(H) Verificar as necessidades e gerar as regras semânticas e de tradução.
38	A → fim	-

4 – Resultado final do Projeto

O Parser (FIGURA 2) realizará o processo de análise sintática e invocará o semântico que fará verificações semânticas e geração de código:

- invocando o SCANNER (T1), sempre que necessitar de um novo TOKEN, consultando as tabelas ACTION e GOTO para decidir sobre as produções a serem aplicadas até a raiz da árvore sintática seja alcançada e não haja mais tokens a serem reconhecidos pelo SCANNER;
- Quando houver uma redução, conduzirá uma chamada à rotina que executará as regras semânticas associadas à regra sintática que foi reduzida;
- Se não houver nenhum erro léxico, sintático ou semântico, um PROGRAMA.C será gerado, caso ocorra pelo menos um erro, a fase de análise continua, porém o arquivo .obj não será criado.

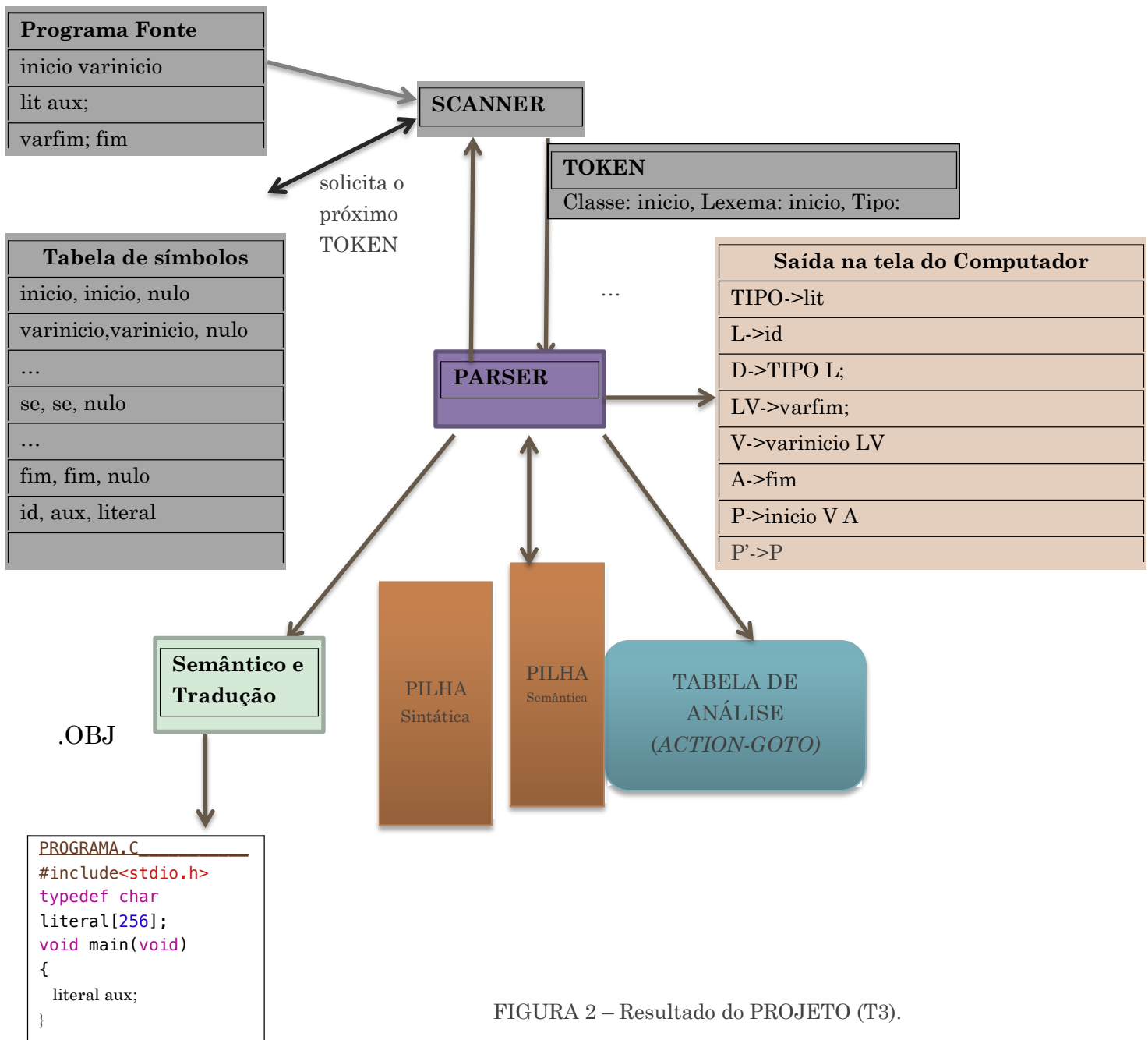


FIGURA 2 – Resultado do PROJETO (T3).

5 – Resultado Final

Ao final de todos os três trabalhos práticos da disciplina, teremos como resultado do estudo de caso um pequeno compilador. Este, utilizando dos tokens reconhecidos e as palavras da linguagem definidas em T1, da gramática livre de contexto (TABELA 1) e do analisador sintático de T2 unido à análise semântica e fases de síntese do T3, compilará o programa fonte em linguagem MgoI, FONTE.ALG - FIGURA 3 (a) em PROGRAMA.C, FIGURA 3(b).

Fonte.ALG	PROGRAMA.C
<pre> inicio varinicio literal A; inteiro B; inteiro D; real C ; varfim; escreva "Digite B"; leia B; escreva "Digite A:"; leia A; se(B>2) entao se(B<=4) entao escreva "B esta entre 2 e 4"; fimse fimse B<-B+1; B<-B+2; B<-B+3; D<-B; C<-5.0; escreva "\nB=\n"; escreva D; escreva "\n"; escreva C; escreva "\n"; escreva A; fim </pre>	<pre> #include<stdio.h> typedef char literal[256]; void main(void) { /*----Variaveis temporarias----*/ int T0; int T1; int T2; int T3; int T4; /*-----*/ literal A; int B; int D; double C; printf("Digite B"); scanf("%d",&B); printf("Digite A:"); scanf("%s",A); T0=B>2; if(T0) { T1=B<=4; if(T1) { printf("B esta entre 2 e 4"); } } T2=B+1; B=T2; T3=B+2; B=T3; T4=B+3; B=T4; D=B; C=5.0; } </pre>

	<pre>printf("\nB=\n"); printf("%d",D); printf("\n"); printf("%lf",C); printf("\n"); printf("%s",A); }</pre>
--	---

(a)

(b)

FIGURA 3– Código fonte (a), código objeto (b).