

# CTC-41



## Compiladores

**Carlos Alberto Alonso Sanches**

# CTC-41



## 6) Análise semântica

Erros de contexto, linguagem C-

# Análise semântica

- As sintaxes das linguagens de programação geralmente são definidas através de uma **gramática livre de contexto**, cujos *parsers* possuem uma eficiência bastante satisfatória.
- No entanto, nas linguagens de programação é preciso realizar algumas verificações que não podem ser descritas por uma gramática livre de contexto:
  - **Declarações e escopo**: declarar antes de usar, identificadores válidos e únicos, escopo correto, índices válidos para vetores, campos de registros, rótulos não repetidos, etc.
  - **Compatibilidade de tipos**: entre operadores e operandos, nos dois lados de uma atribuição, entre argumentos de chamada e parâmetros declarados, valor retornado de uma função, protótipo e definição de subprograma, etc.
  - **Fluxo de controle**: uso de *break* e *continue*, detecção de recursão infinita, etc.
- Para evitar o uso de uma gramática sensível ao contexto, estes aspectos semânticos são verificados com métodos específicos.
- A análise semântica tem caráter estático, isto é, não verifica possíveis erros que ocorrem em tempo de execução (exemplo: acesso inválido devido a valor de variável lida).

# Exemplo: declaração na linguagem C

- A linguagem C permite que o nome de uma função seja usado também como nome de variável local dessa mesma função. No entanto, uma função não pode ter o mesmo nome de uma variável global.
- Exemplo:

```
1 #include <stdio.h>
```

```
2 int a = 7;
```

Linha 2: declaração legal de a

```
3 int b ( ) { int b = 3; return a+b; }
```

Linha 3: declarações legais de b

```
4 float x ( ) { float a = 8.7; return a; }
```

Linha 4: declaração legal de a

```
5 void main ( ) {
```

```
6     char a = '$'; int c; float d;
```

Linha 6: declaração legal de a

```
7     c = b( ); d = x( );
```

```
8     printf ("a = %c, c = %d, d = %g", a, c, d);
```

```
9 }
```

Se a função da linha 3 tivesse nome a, seria uma declaração ilegal

# Exemplo: escopo na linguagem C

- Em linguagens organizadas por blocos como C (ou também naquelas que admitem aninhamento de subprogramas), será preciso incluir informação sobre o escopo na Tabela de Símbolos
- Exemplo:

```
void main ( ) {  
    ...  
  
    { // Início do Bloco 1  
        int a;  
        ...  
        { // Início do Bloco 2  
            ...  
            { // Início do Bloco 3  
                ... = ... + a + ... ;  
            } // Fim do Bloco 3  
        } // Fim do Bloco 2  
    } // Fim do Bloco 1  
  
    { // Início do Bloco 4  
        ... = ... + a + ... ;  
    } // Fim do Bloco 4  
  
}
```

Variável **a** é  
declarada no Bloco 1

Uso correto de **a**  
dentro do Bloco 3

Uso incorreto de **a**  
dentro do Bloco 4

# Tabela de símbolos

- A tabela de símbolos é uma estrutura de dados auxiliar, cujo conteúdo é criado ou alterado nas três fases de análise.
- Os dados armazenados nesta tabela dependem das ações semânticas implementadas. De modo geral, seus símbolos contêm as seguintes informações, entre outras:
  - Identificador associado
  - O que representa: variável ou função
    - Se for função: sua lista de parâmetros
  - Escopo ou nível da sua declaração (pode haver ponteiro para quem o define)
  - Seu tipo: *int*, *float*, *void*, etc.
  - Linhas do código fonte onde aparece
  - Endereço atribuído na geração de código
- Costuma ser implementada com *hashing*, que permite acesso em tempo quase constante:
  - Nome de função: *hashing* utiliza somente o nome da função
  - Nome de variável: *hashing* utiliza nome da variável e nome do escopo onde está declarada

# Algumas operações comuns

- **Busca:** busca um identificador na tabela de símbolos em um determinado escopo. Pode retornar uma *flag* que indica se o nome já estava presente; em caso negativo, retorna um ponteiro para a posição correspondente.
- **Inserir:** insere um identificador na tabela de símbolos em um determinado escopo. Pode retornar uma *flag* que indica se o nome já estava presente; em caso negativo, retorna um ponteiro para a posição correspondente.
- **Elimina:** remove todos os símbolos que estão em um determinado escopo.
- **Declarado:** verifica se um identificador está declarado em um determinado escopo.
- **DefineAtributos:** define os valores dos atributos em uma posição da tabela de símbolos.
- **ObtemAtributos:** obtém os atributos de uma posição da tabela de símbolos.
- etc.

# Durante a análise léxica

- A tabela de símbolos pode ser consultada ou atualizada durante a análise léxica.
- Por exemplo, considere a situação em que um identificador ID é encontrado:
  - Verificar o próximo *token* :
    - Se for "(", então ID é nome de uma função;
    - Caso contrário, ID é nome de uma variável.
  - Se o *token* anterior for "int", "float" ou "void": inserir ID na tabela de símbolos.
  - Caso contrário, atualizar na tabela de símbolos as linhas em que ID aparece.



# Durante a análise sintática

- Durante a construção da árvore sintática, podem ser feitas consultas e atualizações na tabela de símbolos, com o objetivo de realizar verificações semânticas.
- Exemplo de análise semântica na produção gramatical  $E_1 \rightarrow E_2 \text{ div ID}$ :
  - Se  $\text{Busca}(\text{ID}) == \text{false}$ :
    - Então: Erro("variável não declarada")
    - Senão: Se  $(E_2.\text{tipo} \neq \text{inteiro} \text{ ou } \text{ID}.\text{tipo} \neq \text{inteiro})$ :
      - Então: Erro ("tipo inválido para a operação")
  - Se não ocorreu erro:
    - Então:  $E_1.\text{tipo} = \text{inteiro}; E_1.\text{val} = E_2.\text{val} / \text{ID}.\text{val}$

# Estrutura de blocos

- Blocos são comandos compostos com declarações no início.
- A linguagem C permite blocos, mas não possui aninhamento de subprogramas.
- Os escopos definidos pelos blocos formam uma árvore hierárquica:
  - O escopo global é a raiz dessa árvore.
  - Os subprogramas e os blocos internos são filhos do subprograma onde são declarados.
  - Os escopos dos blocos podem ser representados na tabela de símbolos através de identificadores artificiais (exemplo: ##bloco01, ##bloco02, etc.).
  - Cada identificador de escopo pode ter uma lista de escopos filhos.
  - Dois identificadores na tabela de símbolos podem ter o mesmo nome, desde que pertençam a escopos distintos.
  - Deste modo, uma busca na tabela de símbolos deve informar o identificador procurado e o escopo onde a procura será feita.

# Exemplo para blocos

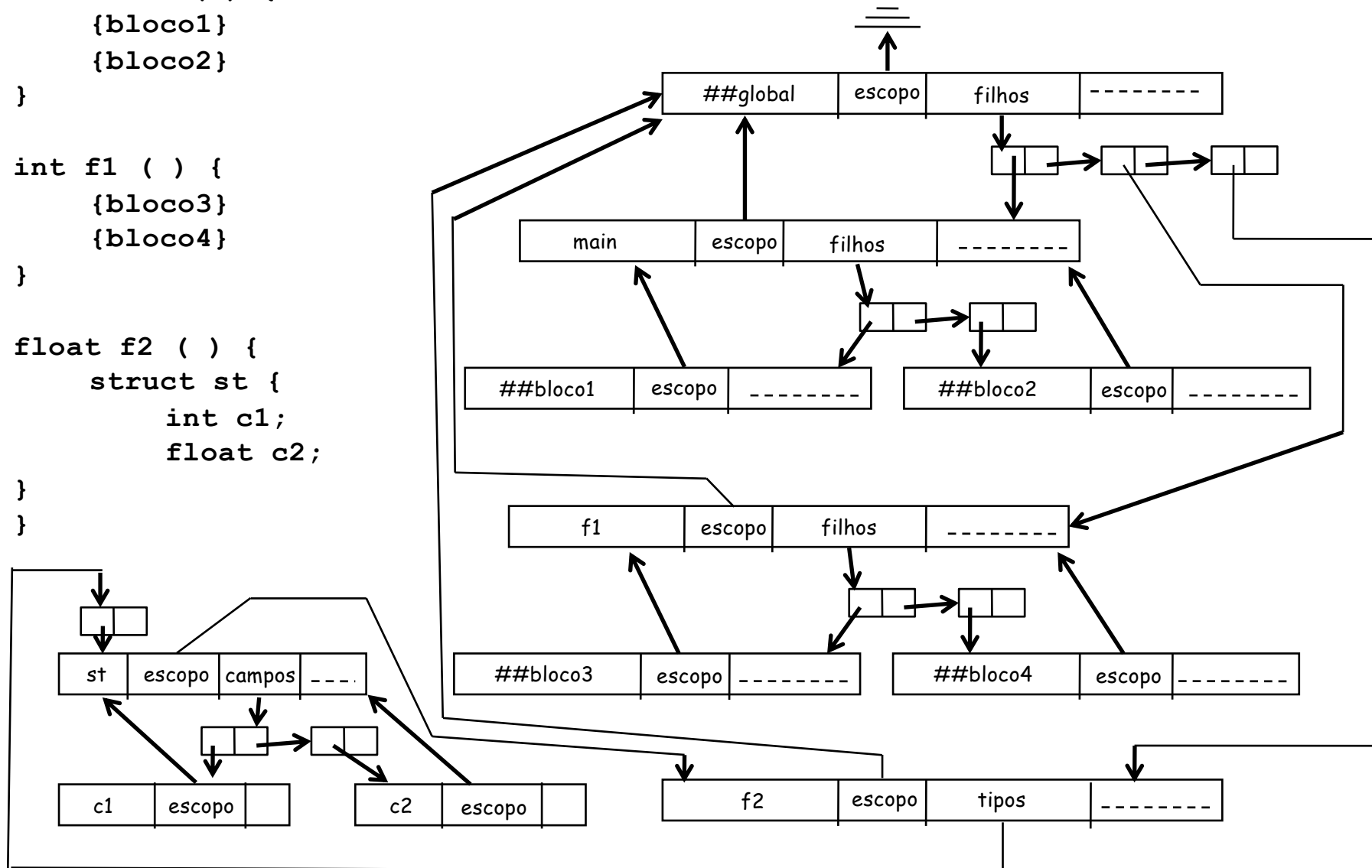
```

void main ( ) {
    {bloco1}
    {bloco2}
}

int f1 ( ) {
    {bloco3}
    {bloco4}
}

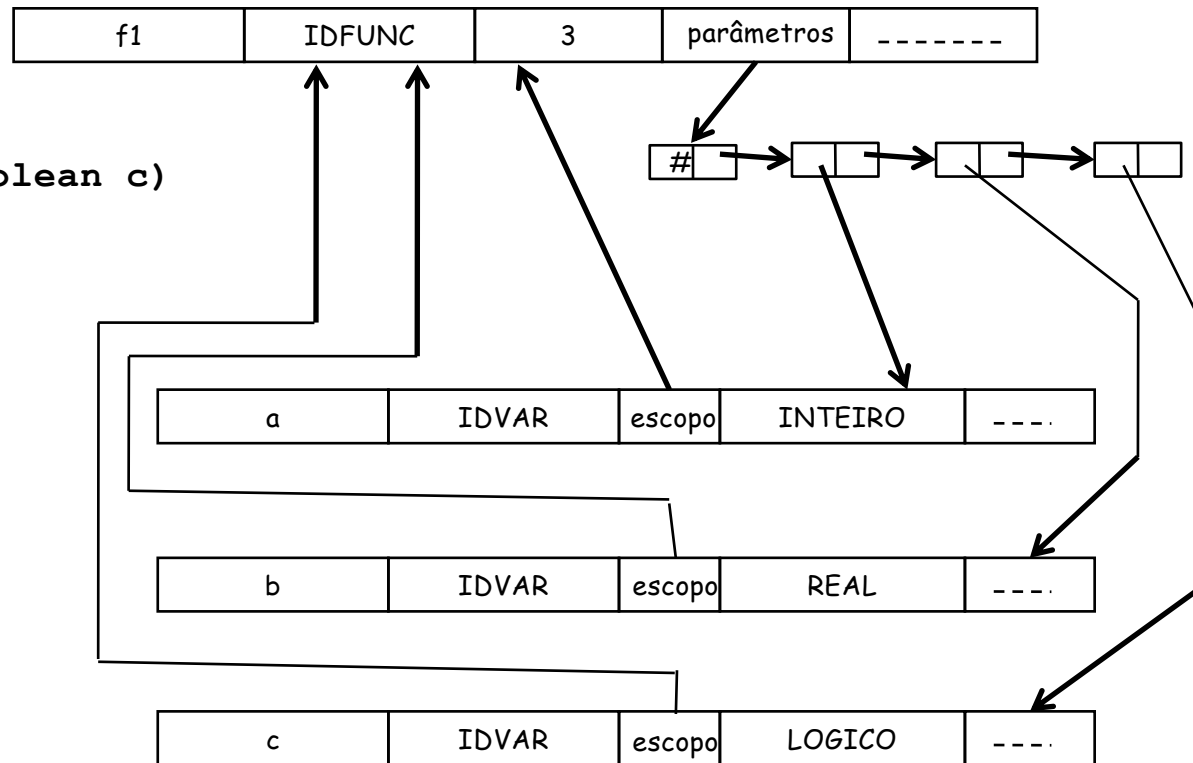
float f2 ( ) {
    struct st {
        int c1;
        float c2;
    }
}

```



# Exemplo para função

```
int f1 (int a, float b, boolean c)
{ ..... }
```



Em cada chamada de `f1`, deverá haver 3 argumentos cujos tipos devem ser respectivamente compatíveis com `a`, `b` e `c`

# Exemplo de regras semânticas

- Considere a gramática abaixo:

decl → tipo ListaVar

tipo → int | float

ListaVar → ID, ListaVar | ID

- Regras semânticas que devem ser verificadas durante a análise sintática:

Produção gramatical	Regras semânticas
decl → tipo ListaVar	ListaVar.TipoDado = tipo.TipoDado
tipo → int	tipo.TipoDado = integer
tipo → float	tipo.TipoDado = real
ListaVar <sub>1</sub> → ID, ListaVar <sub>2</sub>	ID.TipoDado = ListaVar <sub>1</sub> .TipoDado ListaVar <sub>2</sub> .TipoDado = ListaVar <sub>1</sub> .TipoDado if !Busca(ID) Inserir (ID, ID.TipoDado) else Erro("Identificador já declarado")
ListaVar → ID	ID.TipoDado = ListaVar.TipoDado if !Busca(ID) Inserir (ID, ID.TipoDado) else Erro("Identificador já declarado")

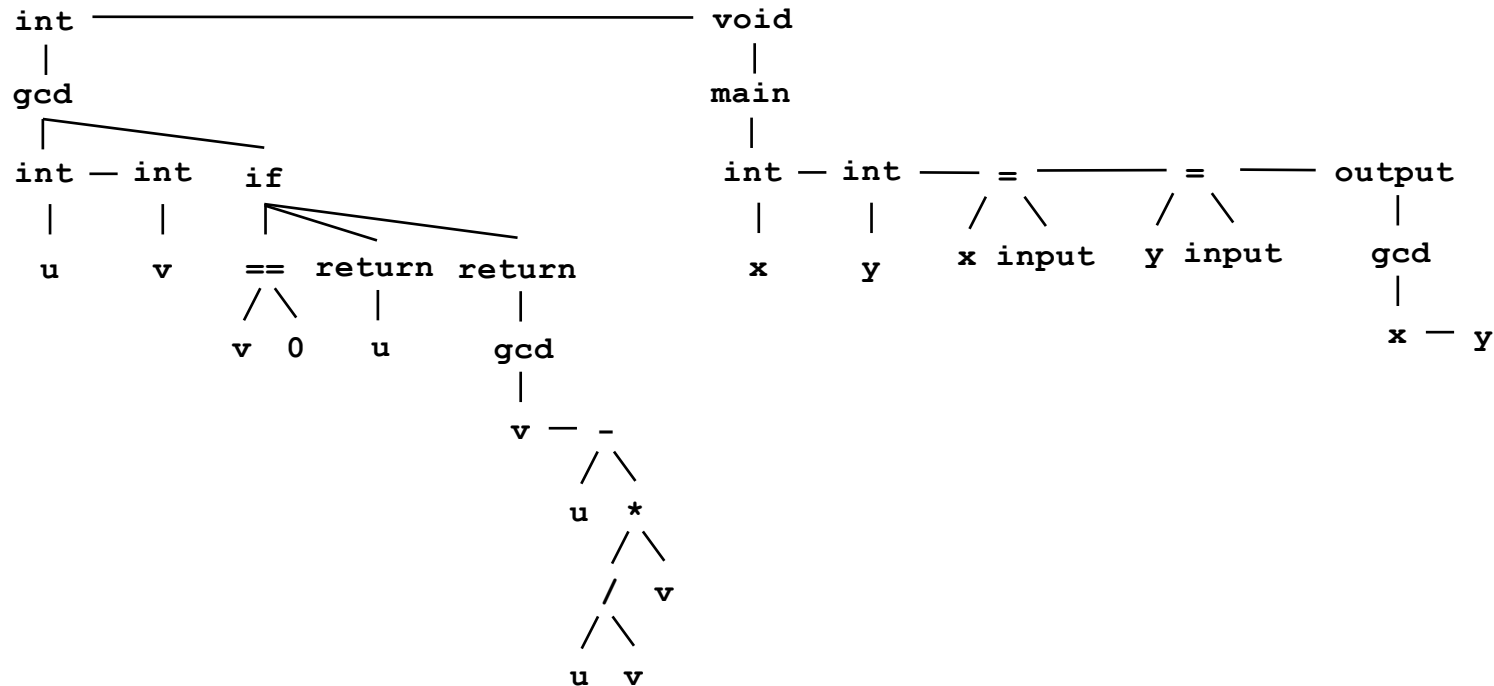
# Exemplo de árvore e de tabela

```

1 int gcd(int u, int v) {
2     if (v==0) return u;
3     else return gcd(v, u-u/v*v);
4 }
5 void main(void) {
6     int x; int y;
7     x = input(); y = input();
8     output(gcd(x,y));
9 }

```

Nome	Escopo	Forma	Tipo	Linhas
u	gcd	var	int	1, 2, 3
main	-	fun	void	5
y	main	var	int	6, 7, 8
gcd	-	fun	int	1, 3, 8
v	gcd	var	int	1, 2, 3
x	main	var	int	6, 7, 8



# Erros que devem ser detectados

- Na análise semântica a ser implementada neste curso, deverão ser detectados ao menos os seguintes erros:
  - Uso de variável não declarada no escopo corrente (haverá aninhamento de blocos, mas não de funções)
  - Atribuição inválida por disparidade de tipo
  - Declarações inválidas de variáveis e funções:
    - Com tipo inválido (exemplo: variável *void*)
    - Identificador já utilizado (considerar regras de escopo)
  - Chamada de função não declarada
  - Compatibilidade entre argumentos e parâmetros (tipos e quantidade)
  - Ausência da função *main()*

# BNF da linguagem C-

programa → declaração-lista  
declaração-lista → declaração-lista declaração | declaração  
declaração → var-declaração | fun-declaração  
var-declaração → tipo-especificador ID ; | tipo-especificador ID [ NUM ] ;  
tipo-especificador → int | void  
fun-declaração → tipo-especificador ID ( params ) composto-decl  
params → param-lista | void  
param-lista → param-lista , param | param  
param → tipo-especificador ID | tipo-especificador ID [ ]  
composto-decl → { local-declarações statement-lista }  
local-declarações → local-declarações var-declaração | vazio  
statement-lista → statement-lista statement | vazio  
statement → expressão-decl | composto-decl | seleção-decl | iteração-decl | retorno-decl  
expressão-decl → expressão ; | ;  
seleção-decl → if ( expressão ) statement | if ( expressão ) statement else statement  
iteração-decl → while ( expressão ) statement  
retorno-decl → return ; | return expressão ;  
expressão → var = expressão | simples-expressão  
var → ID | ID [ expressão ]  
simples-expressão → soma-expressão relacional soma-expressão | soma-expressão  
relacional → <= | < | > | >= | == | !=  
soma-expressão → soma-expressão soma termo | termo  
soma → + | -  
termo → termo mult fator | fator  
mult → \* | /  
fator → ( expressão ) | var | ativação | NUM  
ativação → ID ( args )  
args → arg-lista | vazio  
arg-lista → arg-lista , expressão | expressão



# Características da linguagem C-

- Palavras-chave (sempre minúsculas): `else if int return void while`
- Símbolos especiais: `+ - * / < <= > >= == != = ; , ( ) [ ] { } /* /*`
- `ID` e `NUM` são definidos por expressões regulares (há distinção entre maiúsculas e minúsculas):
  - `ID = letra letra*`
  - `NUM = dígito dígito*`
  - `letra = a | .. | z | A | .. | Z`
  - `dígito = 0 | .. | 9`
- Espaço em branco (composto por brancos, mudanças de linha e tabulações) é ignorado, exceto como separador de `ID`, `NUM` e palavra-chave
- Comentários (delimitados por `/*` e `*/`) não podem ser aninhados, nem colocados dentro de *tokens*
- Todas as variáveis e funções devem ser declaradas antes do uso
- A última declaração deve ser da forma `void main(void)`
- Não existem protótipos, ou seja, não há distinção entre declaração e definição
- Os únicos tipos básicos são `int` e `void`
- `void` é exclusivo para funções
- Uma variável é do tipo inteiro ou um vetor de inteiros, indexado a partir do índice 0

# Características da linguagem C-

- Os parâmetros de uma função são `void` (sem parâmetros) ou uma lista
- Parâmetros seguidos de colchetes são vetores passados por referência, e devem casar com uma variável de tipo vetor durante a ativação
- Não existem parâmetros de tipo função
- Os parâmetros de uma função têm escopo igual ao da declaração composta na declaração de função
- Cada ativação de uma função tem um conjunto próprio de parâmetros
- Funções podem ser recursivas na medida permitida pela declaração antes do uso
- As declarações locais têm escopo igual ao da lista de declarações da declaração composta e se sobrepõem a qualquer declaração global
- `vazio` é um não-terminal que identifica cadeia vazia (equivalente a  $\epsilon$ )
- A ambiguidade do `else` deve ser resolvida através do aninhamento mais próximo
- Será preciso verificar se os índices dos vetores obedecem aos limites especificados
- Será preciso verificar a quantidade de argumentos e o tipo de cada um deles
- Serão consideradas duas funções pré-definidas no ambiente global:
  - `int input (void) { ... }` : retorna um valor inteiro lido do teclado
  - `void output (int x) { ... }` : imprime seu parâmetro inteiro no monitor e pula linha

# Programa em C-: exemplo 1

```
/* SelectionSort em vetor com 10 inteiros */

int x[10];

int minloc (int a[], int low, int high) {
    int i; int x; int k;
    k = low;
    x = a[low];
    i = low + 1;
    while (i < high) {
        if (a[i] < x) {
            x = a[i];
            k = i;
        }
        i = i + 1;
    }
    return k;
}
```

```
void sort (int a[], int low, int high) {
    int i; int k;
    i = low;
    while (i < high-1) {
        int t;
        k = minloc(a,i,high);
        t = a[k];
        a[k] = a[i];
        a[i] = t;
        i = i + 1;
    }
}

void main(void) {
    int i;
    i = 0;
    while (i < 10) {
        x[i] = input();
        i = i + 1;
    }
    sort(x,0,10);
    i = 0;
    while (i < 10) {
        output(x[i]);
        i = i + 1;
    }
}
```

# Programa em C-: exemplo 2

```
/* Variáveis em blocos aninhados */
```

```
void teste (int b, int a) {  
    output(a);  
    output(b);  
    { int a;  
      a = 3;  
      int b;  
      b = 4;  
      output(a);  
      output(b);  
    }  
    a = 5;  
    output(a);  
}
```

```
void main(void) {  
    int a;  
    a = 1;  
    int b;  
    b = 2;  
    teste(a,b);  
    output(a);  
    output(b);  
}
```

# Programa em C-: exemplo 3

```
/* Impressão recursiva da representação  
binária do módulo de um inteiro */
```

```
void binario (int x) {  
    if (x < 2) output(x);  
    else {  
        binario(x/2);  
        output(x - 2*(x/2));  
    }  
}
```

```
void main(void) {  
    int x;  
    x = input();  
    if (x >= 0) binario(x);  
    else binario(-x);  
}
```