Resolução dos Exercícios do Capítulo II

1. Liste pelo menos cinco diferentes tipos de amarrações que ocorrem no seguinte trecho de código C.

```
float j  = 3.2;
j =j - 1.7;
```

A associação do intervalo correspondente ao tipo *float* (tempo de implementação do tradutor).
A escolha do sinal "=" para denotar atribuição (tempo de projeto da linguagem).
A escolha do sinal "–" para denotar subtração (tempo de projeto da linguagem).
A escolha do sinal ";" para denotar separação de comandos (tempo de projeto da linguagem).
A associação dos sinais "=" e "–" às respectivas funções de atribuição e subtração (tempo de compilação).
A associação de *j* ao tipo *float* (tempo de compilação).
A atribuição de *3.2* a variável *j* (tempo de execução).
A atribuição do resultado da expressão *j – 1.7* a *j* (tempo de execução).

2. Especifique as regras de formação de identificadores de C, C++ e Java. Responda ainda se existem limites no número máximo de caracteres que podem ser usados e quais tipos de identificadores especiais são considerados.

C:

É *case sensitive*
O primeiro caractere deve ser uma letra ou sublinhado
Os demais caracteres podem ser letras, dígitos ou sublinhado
Podem ter qualquer número de caracteres, mas somente os 31 primeiros são significativos (isto é, identificadores distintos tem de ter pelo menos um caractere distinto dentre os 31 primeiros caracteres)
Possui vocábulos reservados (por exemplo, *int, char, while*) e vocábulos pré-definidos (por exemplo, *printf, fopen, fclose*)

C++:

É *case sensitive*
O primeiro caractere deve ser uma letra ou sublinhado
Os demais caracteres podem ser letras, dígitos ou sublinhado
Podem ter qualquer número de caracteres, todos significativos
Possui vocábulos reservados (por exemplo, *int, char, while*) e vocábulos pré-definidos (por exemplo, *new* e *delete*)

JAVA:

É *case sensitive*
O primeiro caractere deve ser uma letra ou sublinhado ou cifrão
Os demais caracteres podem ser letras ou dígitos ou sublinhado ou cifrão
Podem ter qualquer número de caracteres
Possui vocábulos reservados (por exemplo, *true, boolean, try*)

3. Considere o seguinte trecho de código em ADA:

```
procedure A is
        u : INTEGER;
        procedure B is
                v: INTEGER;
                 procedure C is
                        x . INTEGER;
                         procedure D is
                                u : INTEGER,'
                        begin
                                null;
                        end D;
                        procedure E is
                                v: INTEGER,'
                        begin
                                u : 7;
                        end E;
                begin
                        null;
                end C,'
                procedure F is
                        y: INTEGER;
                        procedure G is
                                x . INTEGER;
                        begin
                                null;
                        end G;
                begin
                        u := 10;
                end F;
        begin
                null;
        end B;
begin
        null;
end A;
```

Identifique quais variáveis e subprogramas são visíveis em cada um dos subprogramas desse trecho de código. Suponha que novos requisitos do problema demandem que a variável *u* de *D* possa ser acessada por *G*. Quais modificações necessitariam ser feitas no programa? Cite erros que poderiam ocorrer em situações como essa.

Variáveis:
A: u(A)
B: u(A) e v(B)
C: u(A), v(B) e x(C)
D: u(A), v(B), x(C) e u(D) (u(A) acessado através de referência seletiva *A.u*)
E: u(A), v(B), x(C) e v(E) (v(B) acessado através de referência seletiva *B.v*)
F: u(A), v(B) e y(F)
G: u(A), v(B), y(F) e x(G)

Subprogramas:
A: B
B: A, B, C e F

C: A, B, C, D, E e F
D: A, B, C, D, E e F
E: A, B, C, D, E e F
F: A, B, C, F e G
G: A, B, C, F e G

Para referenciar a variável u do subprograma D em G seria necessário definí-la em B.

A variável u de D também passaria a ser visível pelos subprogramas B, C, E e F, o que pode ser indesejado. Isso poderia provocar erros de alteração indevida dessa variável nesses subprogramas.

4. Indique qual valor será escrito pelo trecho de código seguinte no caso da linguagem de programação utilizada adotar escopo estático e no caso dela adotar escopo dinâmico.

```
procedimento sub () {
        inteiro x =  1;
        inteiro y =  1;
        procedimento sub1() {
                se (x = 1 &y = 1,) então
                        sub2();
                senão
                        sub3();
        }
        procedimento sub2() {
                inteiro x = 2;
                y = 0;
                sub1(),'
        }
        procedimento sub3() {
                escreva(x):
        }
        sub 1();
}
```

Cite e explique os problemas de legibilidade do trecho de código acima quando se adota o escopo estático e o escopo dinâmico.

Escopo estático: 1
Escopo dinâmico: 2

No escopo dinâmico, ao chamarmos a função *sub3* depois de *sub2* ser executada, a função fará referência ao *x* criado em *sub2*. Por isso imprimirá o valor 2. Já no escopo estático *sub3* irá referenciar o valor de *x* em *sub* e imprimirá o valor 1.
No escopo dinâmico temos que analisar a seqüência em que os subprogramas são chamados para determinar o valor das variáveis que não são locais. É preciso mais cuidado e atenção por parte do programador ao manipular essas variáveis.

O escopo estático é bem mais legível, pois não é necessário seguir a cadeia de chamadas das funções para conhecer o valor de uma variável não local, tornando o entendimento do código menos confuso.

5. Compare, em termos de legibilidade, as opções de C e C++ relativas à localização das definições e declarações nos programas.

Em C, definições e declarações devem ser feitas globalmente ou no início de um bloco de código. Em C++, elas podem ser feitas globalmente ou em qualquer ponto de um bloco de código. Como variáveis globais possuem a mesma regra de localização, a comparação realizada aqui se refere às declarações e definições não globais. Enquanto a opção de C++ possibilita a colocação das definições e declarações bem próximas do local de seu uso, ela eventualmente pode ser mais difícil de ser encontrada, uma vez que pode estar localizada em qualquer ponto do bloco de código antes de seu primeiro uso. Por sua vez, a opção de C fixa o local onde as declarações e definições podem ser encontradas, facilitando a busca. Contudo, esse local pode ser bem distante do ponto de uso, o que pode dificultar a leitura.

6. Identifique o problema que ocorre no seguinte trecho de código C. Explique porque ele ocorre e indique como poderia ser resolvido.

```
void circulo () {
        #define pi 3.14159
        float raio =  3;
        float area = pi * raio * raio;
        float perimetro = 2 * pi * raio;
}
void pressao () {
        float pi = 3.2, pf = 5.3;
        float variacao;
        variacao =  pf - pi;
}
```

Quando o programa é compilado, o pré-processador encontra cada ocorrência da palavra *pi*, após o *define*, e a substitui por *3.14159*. Isto causará erro na *pressão*. As palavras *pi* serão substituídas por uma constante, o que não deveria ter sido feito, pois a intenção era a criação de uma variável de nome *pi* (representando pressão interna).

Para resolver este problema pode-se utilizar a definição "*const*" em vez de *define*. Uma definição através de *const* obedece a regras de escopo (a constante será reconhecida do ponto de definição até o final do subprograma na qual foi definida).

7. Indique quais valores serão escritos pelo seguinte programa em C. Explique sua resposta e discuta a postura da linguagem em termos de ortogonalidade e de potencialidade para indução de erros de programação.

```
int i;
void main () {
        printf("%d\n", i);
```

```
        f();
}
void f () {
        int i;
        printf(''%d\n '', i);
}
```

A variável *i* foi definida como global. Automaticamente ela será inicializada com o valor *0* (zero). Portanto, na execução da função *main*, o primeiro valor a ser impresso é *0* (zero). Contudo, quando a função *f* é chamada, cria-se uma variável local *i* na memória. Variáveis locais em C não são inicializadas automaticamente. Assim, a impressão desta variável será o conteúdo corrente da área de memória aonde ela foi alocada.
Isso caracteriza uma falta de ortogonalidade na linguagem, o que pode levar a erros de programação, visto que o programador pode achar que uma variável, seja global ou local, será sempre inicializada automaticamente com zero.

8. Uma declaração de função é um segmento de código contendo apenas a sua assinatura (isto é, um segmento de código com o cabeçalho da função, mas sem seu corpo). Apresente uma situação na qual a declaração de funções é útil (ou necessária) em C. Justifique sua resposta explicando para que o compilador utiliza a declaração.

Uma situação na qual a declaração de funções é necessária ocorre quando se deseja compilar um arquivo que utiliza funções definidas em outro arquivo já compilado ou que será compilado posteriormente. Nesse caso, as declarações são necessárias para que o compilador possa verificar se as chamadas das funções são consistentes com os tipos requeridos em sua assinatura.