Universidade Federal de Minas Gerais
Ciência da Computação

Linguagens de Programação - Haniel Barbosa

Lista de Exercícios 4

Passagem de Parâmetros

1. Considere o seguinte programa em C++:

```
#include <stdio.h>
int x=0;
void p(int, int);
void main(){
    int x = 1;
    p(x, x);
}
void p(int y, int z){
    x = x+1;
    y = y+1;
    z = z+1;
    printf("%d\n", x+y+z);
}
```

   (a) Qual o valor impresso pelo programa? Lembre-se que C++ possui escopo estático.

   (b) Suponha que nós modifiquemos a declaração de p tal que os parâmetros sejam passados por referência. Isto é, `p(int &y, int &z)`.... Qual o valor impresso neste caso? Justifique.

2. A passagem de parâmetros por expansão de macros é um mecanismo bastante utilizado em C. Um exemplo é dado abaixo:

```
#define SUM(X, Y) (X) + (Y)

int main(int argc, char** argv){
    printf("sum = %d\n", SUM(argc, argv[0][0]));
}
```

   (a) Macros são expandidas por um componente do compilador chamado pré-processador. Escreva o código do programa acima após o pré-processamento.

   (b) Um dos problemas com expansão de macros é a chamada capture de variáveis. Explique o que é esse problema.

(c) Um outro problema é a múltipla avaliação de parâmetros. Essa múltipla avaliação faz com que o programa possa ter um significado diferente da intenção do programador. Escreva um programa em C que prove que parâmetros de macros são avaliados múltiplas vezes.

3. Um dos mecanismos de passagem de parâmetros é chamado de passagem por nome. Nesse tipo de passagem, os parâmetros não são avaliados imediatamente. Cada parâmetro real é avaliado no contexto da chamada da função, no momento em que são usados. Esta ideia foi lançada em Algol, e também teve seu uso em Simula. Apesar de ter sido abandonado por ser muito difícil de implementar, coisas interessantes podem ser feitas utilizando passagem por nome. Considere o programa abaixo, escrito em Simula, onde os parâmetros k e u são passados por nome.

```
Integer Procedure Sigma (k ,m ,n , u);
    Name k,u;
    Integer k,m,n,u;
Begin
    Integer s;
    k := m;
    While k <= n Do Begin s := s + u; k := k + 1; End;
    Sigma := s;
End;
```

(a) Qual o valor de Z na chamada abaixo?

```
Integer Z;
Integer i;
Z := Sigma (i,1,4,i**2);
```

(b) Explique o que o programa abaixo faz. Não precisa escrever o valor calculado, apenas dizer o que ele calcula.

```
Integer Z;
Integer i;
a := io.read_integer();
Z := Sigma (i,1,100,1 / (i+a) **2);
```

4. Consider a classe abaixo, implementada em Java:

```
class MyInt {
    int i;
    MyInt(int k) {
        i = k;
    }
    void swap1(MyInt j) {
        MyInt tmp = j;
        j = new MyInt(i);
        i = tmp.i;
    }
```

```
    void swap2(MyInt j) {
        MyInt tmp = j;
        j.i = i;
        i = tmp.i;
    }
    void swap3(int j) {
        int tmp = j;
        j = i;
        i = tmp;
    }
}
```

Cada uma das próximas questões é completamente independente uma das outras. Estas questões devem ser respondidas com base nas definições abaixo:

```
MyInt m1 = new MyInt(3);
MyInt m2 = new MyInt(4);
```

(a) Qual é o valor de m1.i e m2.i depois da chamada m1.swap1(m2)?

(b) Qual é o valor de m1.i e m2.i depois da chamada m1.swap2(m2)?

(c) Qual é o valor de m1.i e m2.i depois da chamada m1.swap3(m2.i)?

(d) Qual é o tipo de passagem da parâmetros que Java adota para tipos primitivos (int, float, char, etc)?

(e) Qual é o tipo de passagem da parâmetros que Java adota para objetos?

Programação Lógica

1. Dados os seguintes predicados:

```
parent(kim,holly).
parent(margaret,kim).
parent(margaret,kent).
parent(esther,margaret).
parent(herbert,margaret).
parent(herbert,jean).
greatGrandParent(GGP,GGC) :- parent(GGP,GP), parent(GP,P), parent(P,GGC).
sibling(X,Y) :- parent(P,X), parent(P, Y), not(X=Y).
```

(a) Defina um predicado `firstCousin` tal que `firstCousin(X, Y)` seja verdadeiro se e somente se `X` for um primo em primeiro grau de `Y`. Lembre-se que uma pessoa não é prima em primeiro grau dela mesma. Tampouco irmão são.

(b) Defina um predicado `descendant` tal que `descendant(X, Y)` seja verdadeiro se e somente se `X` for um descendente de `Y`.

2. Defina um predicado `third` tal que `third(X, Y)` seja verdadeiro se e somente se `Y` for o terceiro elemento da lista `X`. Isto pode ser expresso como um fato.

3. Defina um predicado `dupList` tal que `dupList(X, Y)` seja verdadeiro se e somente se `X` for a mesma lista que `Y`, mas com cada elemento de `Y` repetido uma vez. Por exemplo, se `X` for a `lista [1, 2, 3]`, então `Y` deverá ser a `lista [1, 1, 2, 2, 3, 3]`. Se `X` for `[]`, então Y também será `[]`. Verifique se seu predicado funciona em ambas as direções, i.e., a consulta `dupList(X,[1, 1, 2, 2, 3, 3]).` produz o resultado `X = [1, 2, 3]`.

4. Vendo listas como conjuntos, defina um predicado `isEqual` tal que `isEqual(X, Y)` seja verdadeiro se e somente se `X` e `Y` forem conjuntos iguais. Dois conjuntos são iguais se eles possuem os mesmos elementos, independente da ordem. Seu predicado não precisa funcionar direito quando `X` ou `Y` não estão completamente instanciados.

5. Defina um predicado isDifference tal que `isDifference(X, Y, Z)` seja verdadeiro se e somente se `Z` contiver os elementos de `X` que não aparecem em `Y`. Seu predicado deverá funcionar independente da ordem em que os elementos de `Z` são dados. Seu predicado não precisa funcionar quando `X` ou `Y` não estiverem instanciados.

6. Considere o predicado append:

```
append([], B, B)
append([Head|TailA], B, [Head|TailC]) :-
append(TailA, B, TailC).
```

   Mostre uma refutação que demonstre a validade de `append(X, Y, [1, 2])`

7. Escreva um predicado `maxList(L, M)` que seja verdadeiro quando recebe uma lista `L` de números e unifique `M` com o maior número nesta lista. O predicado deve ser falso se a lista estiver vazia.

8. Escreva um predicado `nqueens(N, X)`, que para um inteiro `N` seja verdadeiro quando `X` for uma solução para o problema das `N` rainhas. O problema das `N` rainhas é a generalização do problema das oito rainhas: deve-se colocar N rainhas em um tabuleiro de tamanho `N × N`.