Universidade Federal de Minas Gerais
Ciência da Computação

Igor Lacerda Faria da Silva

Lista de Exercícios 4

Passagem de Parâmetros

1. (a) É impresso o número 5.

   (b) Será impresso o número 7, pois a variável `y` e a variável `z` apontariam para o mesmo endereço. Dessa maneira, a variável global `x` manteria sua contribuição de 1 para a soma, mas o valor comum da referência dos parâmetros seria incrementado duas vezes.

2. Não, eu não diria que é um mecanismo bastante utilizado em C

   (a)
```c
#include <stdio.h>

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

   (b) A captura de variáveis ocorre quando uma macro recebe dois ou mais argumentos iguais e, durante a expansão, as variáveis são 'misturadas', o que pode causar um comportamento indesejado.

   (c)
```c
#include <stdio.h>

#define LIGMA(X) (X) = (X) + 1, (X) = 2 * (X)

int main(void) {
  int x = 0;
  printf("ligma = %d", LIGMA(x));
}
```

3. (a) 30

   (b) O programa lê um um inteiro a da entrada e calcula a soma do quadrado das frações $\frac{1}{i+a}$ com i variando de 1 até 100.

4. (a) m1: 4, m2: 3

   (b) m1: 4, m2: 3

   (c) m1: 4, m2: 4

   (d) Passagem por referência.

   (e) Passagem por cópia.

Programação Lógica

1. (a)
```
firstCousin(X, Y) :-
  parent(P1, X),
  parent(P2, Y),
  P1 \= P2,               % X e Y têm pais diferentes
  \+ sibling(X, Y).       % X e Y não são irmãos
```

   (b)
```
descendant(X, Y) :- parent(Y, X).
descendant(X, Y) :- parent(Z, X), descendant(Z, Y).
```

2.
```
third(X, Y) :- X = [_, _, Y|_].
```

3.
```
dupList([], []).
dupList([H|T], [H, H|X]) :- dupList(T, X).
```

4.
```
isEqual([], []).
isEqual(X, Y) :- permutation(X, Y).
```

5.
```
isDifference([], _, []).
isDifference([X|Xs], Y, Z) :-
  member(X, Y),
  isDifference(Xs, Y, Z).
isDifference([X|Xs], Y, [X|Z]) :-
  not_member(X, Y),
  isDifference(Xs, Y, Z).

not_member(_, []).
not_member(X, [Y|Ys]) :-
  X \= Y,
  not_member(X, Ys).
```

6. I Assumindo a negação da cláusula a ser demonstrada:
   $\sim$ (append(X, Y, [1, 2]))

   II Aplicando a negação na primeira cláusula do predicado append:
   $\sim$ (append([], Y, [1, 2]))

   III Aplicando a negação na segunda cláusula do predicado append:
   $\sim$ (append([Head|TailA], Y, [1, 2])) :- $\sim$ (append(TailA, Y, TailC))

   IV Aplicando a negação na primeira cláusula do predicado append novamente:
   $\sim$ (append([Head|TailA], Y, [1, 2])) :- $\sim$ (append(TailA, Y, TailC)), $\sim$ (Head = 1), $\sim$ (TailC = [2])

   V Aplicando a negação em Head = 1:
   $\sim$ (append([Head|TailA], Y, [1, 2])) :- $\sim$ (append(TailA, Y, TailC)), Head $\neq$ 1, $\sim$ (TailC = [2])

VI Aplicando a negação em TailC = [2]:
$\sim$ (append([Head|TailA], Y, [1, 2])) :- $\sim$ (append(TailA, Y, TailC)), Head $\neq$ 1, TailC $\neq$ [2]

VII Unificando a cláusula negada com a primeira cláusula do predicado append: append([], Y, [1, 2])

VIII Unificando a primeira cláusula com os termos correspondentes:
Y = [1, 2]
Contradição: com a unificação em VIII, temos Y igual a [1, 2], o que contradiz a negação em VI (TailC $\neq$ [2]).

7.
```
maxList([X], X).
maxList([X|Xs], M) :-
  maxList(Xs, Max),
  (X > Max -> M = X ; M = Max).
```

8.
```
nqueens(N, X) :-
  length(X, N),
  X ins 1..N,
  safe_queens(X).

safe_queens([]).
safe_queens([Q|Queens]) :-
  safe_queens(Queens, Q, 1),
  safe_queens(Queens).

safe_queens([], _, _).
safe_queens([Q|Queens], Q0, D0) :-
  Q0 #\= Q,
  abs(Q0 - Q) #\= D0,
  D1 #= D0 + 1,
  safe_queens(Queens, Q0, D1).
```