Linguagens de Programação
Professor Haniel Barbosa
Lista de exercícios 4

Luís Felipe Ramos Ferreira
2019022553

June 21, 2023

1 Passagem de Parâmetros

1. (a) O valor impresso pelo programa é 5

   (b) O valor impresso nesse caso é 7. Como *C++* possui escopo estático, o valor da variável x utilizado dentro da função `p` será o definido globalmente, isto é, onde x é giaul a zero. O variável x dentro do escopo da função *main* terá sua referência passada para os dois parâmetros da função `p`. Dentro do escopo da função, o valor da vriável global x será incrementada para 1, e as variáveis y e x serão incrementadas. Como elas são referências para a variável x com valor igual a 1 no escopo da função `main`, esse mesmo valor será incrementado 2 vezes. Podemos entender x e y como duas variáveis que apontam para a mesma posição da memória. Dessa maneira, o valor nessa posição da memória será incrementado para 3, e ele será somado duas vezes no comando de impressão. Dessa forma, teremos uma saída com valor 1 + 3 + 3, ou então 7.

2. (a)

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

```

   (b) A captura de variáveis é um problema que ocorre na expansão de macros quando dentro da definição da macro há a definição de uma variável cujo nome está sendo utilizado por outra variável passada como parâmetro para a macro, causando um conflito entre a qual variável cada operação se refere. Um exemplo da situação apresentado em sala foi o do uso da macro *SWAP*, que define uma função para a troca de valores entre duas variáveis, e que dentro do escopo da macro define uma variável auxiliar denominada `temp`. Abaixo, o código disponibilizado nas notas de aula pode ser visto:

```
#include "stdio.h"
#define SWAP(X, Y)    \
    {                 \
        int tmp = X;  \
        X = Y;        \
        Y = tmp;      \
    }
int main()
{
    int a = 2;
    int tmp = 15;
    printf("Before: %d, %d\n", a, tmp);
    SWAP(a, tmp);
    printf("After: %d, %d\n", a, tmp);
}

```

Neste cenário, após o pré processamento, o código da maneira a seguir. Nela, podemos ver que há a captura da variável `tmp`, e o comportamento não é o desejado, uma vez que a variável auxiliar `tmp` que é definida dentro da macro encobre e faz com que a variável `tmp` original, passada como parâmetro, se perca, e assim o *swap* de variáveis não ocorre como esperado.

```c
#include "stdio.h"
#define SWAP(X, Y)    \
    {                 \
        int tmp = X;  \
        X = Y;        \
        Y = tmp;      \
    }
int main()
{
    int a = 2;
    int tmp = 15;
    printf("Before: %d, %d\n", a, tmp);
    {
        int tmp = a;
        a = tmp;
        tmp = tmp;
    }
    printf("After: %d, %d\n", a, tmp);
}

```

(c) O código em *C* abaixo contêm o problema da múltipla avaliação de parâmetros. Ele foi apresentado e discutido durante as aulas da disciplina.

```c
#include "stdio.h"
int x = 0;
int foo()
{
    x++;
    return 1;
}
#define MAX(X, Y) ((X) > (Y) ? (X) : (Y))
int main()
{
    int y = MAX(0, foo());
    printf("Max: %d, global x: %d\n", y, x);
}

```

Após a expansão da macro, como os argumentos passados irão ser substituídos no corpo da macro, a função `foo` será chamada duas vezes, o que irá impactar no incremento da variável global `x`. A saída do código a seguir seria então:

Max: 1, global x: 2

3. (a) O valor de `z` na chamada é 30

   (b) O programa em questão irá calcular o resultado da seguinte expressão/soma para algum valor de a lido na entrada:

$$\sum_{i=1}^{100} \frac{1}{(1+a)^2}$$

4. (a) m1.i = 4 e m2.i = 4

   (b) m1.i = 3 e m2.i = 3

   (c) m1.i = 4 e m2.i = 4

   (d) Java adota o tipo de passagem por valor para tipos primitivos

   (e) Java adota o tipo de passagem por valor para objetos

2 Programação Lógica

1. As implementações para os predicados *firstCousin* e *descendant* estão abaixo dos comentários que indicam a letra respectiva à cada um.

```
parent(kim,holly).
parent(margaret,kim).
parent(margaret,kent).
parent(esther,margaret).
parent(herbert,margaret).
parent(herbert,jean).
parent(kent, bruce).
parent(jean, sam).
parent(sam, max).
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
firstCousin(X, Y) :- sibling(M, N), parent(M, X), parent(N, Y
    ), not(X=Y), not(sibling(X, Y)).

% b
descendant(X,Y) :- parent(Y, X); (parent(Y, K), descendant(X,
     K)).
```

2. Implementação do predicado *third*.

```
% fact
thirdFact([_, _, X|_], X).

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

3. Implementação do predicado *dupList*.

```
dupList([], []).
dupList([H|T], [G|[G|K]]) :- dupList(T, K), H=G.
```

4. Implementação do predicado *isEqual*, com o uso do predicado *subSet* da biblioteca padrão de manipulação de listas em *Prolog*. Podemos dizer que dois conjuntos são iguais se eles são subconjuntos um do outro. Poderiamos também utilizar a própria implementação de permutação da biblioteca citada.

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

5. Implementação do predicado *isDifference*. A implementação faz uso da função *subtract* da biblioteca padrão de manipulação de listas em *Prolog*.

```
isDifference(X, Y, Z) :- subtract(X, Y, Z).
```

6. Fato: append ([], B, B) Consulta: append (X, Y, [1, 2])

   Queremos mostrar que a consulta é valida. Assumimos então que ela é falsa, e, a partir dos fatos, concluímos falso. Logo, a consulta deve ser verdadeira, por contradição.

   not (append (X,Y,[1,2])) e append ([],B,B)
   Aplicando a substituição:

$$\sigma : \{X = [], Y = [1, 2], B = [1, 2]\}$$

not (append ([],[1,2],[1,2]))
Falso
Logo, a consulta é válida. Temos uma solução em que X=[] e Y=[1,2].

7. Implementação do predicado *maxList*. A implementação faz uso da função *max_member* da biblioteca padrão de manipulação de listas de *Prolog*.

```
maxList(L, M) :- max_member(M, L).
```

8. Implementação do predicado *nqueen*. A implementação seguiu o que foi implementado pelo professor em sala de aula, em relação ao mesmo problema. A implementação também baseou-se na implementação do mesmo predicado mas para 8 rainhas, disponibilizado no site da disciplina.

```
perm([], []).
perm(List, [H|Perm]) :- select(H, List, Rest), perm(Rest, Perm).

test([], _, _, _).
test([Y|Ys], X, Cs, Ds) :-
    C is X-Y, \+ member(C, Cs),
    D is X+Y, \+ member(D, Ds),
    X1 is X + 1,
    test(Ys, X1, [C|Cs], [D|Ds]).

nqueen(Q, N) :- numlist(1, N, L) , perm(L, Q), test(Q, 1, [], [])
    .

% aux
allnqueen(A, N) :- findall(Q, nqueen(Q, N), A).
countAllnqueen(C, N) :- allnqueen(A, N), length(A, C).
```