UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Métodos e Técnicas de Programação
2ª Prova (2017/2)

Rômulo de Barros Lima Júnior (Engenharia De Computação - 11511ECP011)

Lucas Alesterio Marques Vieira (Engenharia De Computação - 11621ECP016)

Uberlândia
16/10/2017

MARQUE SUA TURMA:

[ ] U (terças-feiras)

[X] V (segundas-feiras)

Tabela 1: **Registro do Grupo**

ID	Nome	Matrícula
1	Romulo de Barros Lima Junior	11511ECP011
2	Lucas Alestério Marques Vieira	11621ECP016

QUESTÃO 01

MAT0 = 11511ECP011	MAT1 = 11621ECP016	MAT2 = 10011EBI075
KANO0 = 1	KANO1 = 3	KANO2 = 4
KCUR0 = 3	KCUR1 = 3	KCUR2 = 2
KNUM0 = 3	KNUM1 = 8	KNUM2 = 4

QUESTÃO 02

a) O resultado apresentado foi:
"A forca gravitacional entre as esferas azul e vermelha, distantes 13.043197 metros uma da outra, eh de 4.704776 Newtons."

b) O programa recebe os dados na estrutura "Esfera" a qual tem seus membros "x, y, z e massa", logo as estruturas "Esfera" criada são: azul e vermelha na função "*main*" do programa, com isso têm os dados entrando através da estrutura. Logo temos funções que retornam um valor *"double"* mas com entrada de estrutura "Esfera" isso implica que dentro das variáveis da estrutura "esfera" tem os membro para serem utilizados, um jeito mais limpo de se passar dados diversos em uma função. Dentro da função é chamada as estruturas declaradas na função que são: um e dois, essas estruturas são todas do tipo esfera e dentro de cada uma tem os valores de "x, y, z e massa" com isso é feita as operações com os membros das duas estruturas sobre distancia e força gravitacional agindo sobre elas.

QUESTÃO 03

a) instancia: zupkfa\ 1.100000
instancia: ytoje`[1.100000
novainsta: ytoje`[-8.900000

b) Se formos olhar somente os membros da estrutura "Armazem" temos 16 bytes quando somado o tamanho do "*char*" (que no caso do nosso grupo foi de 8 bytes) com a primitiva "*double*" que armazena 8 bytes na memoria, logo temos que os membros da estrutura tem 16 bytes juntos

c) O programa cria um "*string*" da estrutura Armazem com tamanho conforme a constante de cada grupo, no nosso caso a constante foi o numero 8 , logo nossa "*string"* tem 8 de tamanho, e é preenchida com uma lógica usando a variável de

controle de um laço usando a letra ‘z’ e fazendo uma operação com mais 4 constantes geradas na Questão 1, logo é colocado um caracter ‘\0’ na ultima posição da “*string*” para que o compilador possa entender o final da “*string*”, logo temos o preenchimento da variável preço também sendo preenchida com uma logica usando 4 constantes e dividindo por 10.0, depois disso é apresentada a estrutura “Armazem” conforme ela foi preenchida, logo temos uma nova variável referente a estrutura “Armazem” que é a “novainsta” que chama a função “criaInstancia” a qual tem como a passagem de valores: o endereço na memoria do membro código da estrutura “Armazem” instancia e o valor de preco da mesma estrutura, logo todas as mudanças que ocorrerem em no membro código permanecerá fora do escopo da função “criaInstancia” mas as alterações no membro preco não, pois não foi passado seu endereço, mas sim somente o seu valor, logo as alterações não permaneceram fora do escopo da função, com isso temos mais algumas logicas dentro da função as quais alteram os valores de “instancia.codigo” e armazena na outra estrutura “novainsta” e as operações com “instancia.preco” são armazenadas no retorno da função que é uma estrutura “Armazem” também.

d) Logo os membros de “instancia” e “novainsta” que são iguais são os membros “código” das duas e seus membros “preco” são distintos e isso está tudo explicado na alternativa anterior

QUESTÃO 04

a) :
```
: (2) 10011EBI075 : (1) 11621ECP016 : (0) 11511ECP011
```

b)
```
void imprime(Texto * dado, int qtde) {
    if(qtde > 0) {
    printf(": (%d) %s ", dado->id, dado->mat);
    imprime(dado+1,qtde-1);
    }
else printf(":\n");
}
```

c) Não, isso não pode ser chamado de um erro, pois a função imprime usa de recursão, logo no programa original nós tínhamos que a recursão era chamada antes de executar a ação (que nesse exemplo era de mostrar na tela o “id” e a “matricula”) com isso nos tínhamos que a função ia chegar ao seu ponto de parada “qtde > 0” e ir executando o resto do escopo de cada uma das vezes em que ela foi chamada, com isso apresentava em ordem contraria. Resumindo, tínhamos que colocar a ação da função antes de chamar a recursividade para esse exemplo.

QUESTÃO 05

a) Fibonacci(2) = 1
Fibonacci(7) = 13
Fibonacci(3) = 2
Numero secreto = 5

b) A função funciona da seguinte maneira,

c)

Id escopo	Escopo origem	Retorno de	Índice	Condição	Resposta
0	Main	-	5	Falsa	Fib(4)+fib(3)
1	0	-	4	Falsa	Fib(3)+fib(2)

QUESTÃO 06

```
#include <stdio.h>
#define N 256

long long unsigned int trifib(long long unsigned int * vec) {
if ((*vec) == '*') //onde no main um vetor deve ser preenchido
com os * para todos os índices maiores que 2
*vec = trifib(vec-1)+ 2 * trifib(vec-2)+3 * trifib(vec-3);
return *vec;
}

int main() {
long long unsigned int A[N]; A[0] = 0; A[1] = 1; A[2] = 2;
int n = 100, i;
for(i = 3; i < N; i++) A[i] = '*' ; //exemplo necessário para se
usar a função criada
for(i = 0; i < n; i++)
printf("%llu%s", trifib(A+i), (i==n-1)? ".\n" : "; ");
return 0;
}
```