1 Avaliação
Linguagens de Programação 2020.2

Aluno: _____

1. Considere o seguinte trecho de código:

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main()
{
    char *p,*q;
    p=(char*)malloc(3*sizeof(char));
    q=p;
    strcpy(p,"hello");
    printf("p=%s",p);
    printf("q=%s",q);
    free(q);
    q=NULL;
    gets(p);
    gets(q);
    printf("%s",p);
    printf(?%s?,q);
}
```

   (a) Qual é o problema relacionado com os ponteiros está exemplificado no código acima?

   (b) Cite um método que resolve o problema do código acima e avalie o seu custo de implementação.

2. Considere o seguinte trecho de código:

```c
#include <stdio.h>
#include <stdlib.h>

void foo(){
    int * ptr =(int *) malloc(sizeof(int));
   return ;
}
int main(void){

    while (1) foo();
    return 0;
}
```

   (a) Qual é o problema relacionado com os ponteiros está exemplificado no código acima?

   (b) A variável ptr é estática, dinâmica na pilha ou dinâmica no heap? O que acontece com a variável ptr após o retorno da função?

   (c) b. A função foo irá falhar, cedo ou mais tarde, qual é o motivo? Qual é a funcionalidade que os ponteiros não tem que reduziria tais erros de programação e como ela poderia atuar neste problema?

3. Considere o seguinte trecho de código em C:

```c
void fun1(void);
void fun2(void);
void fun3(void);
void main(){
int a,b,c;
...
}
void fun1(void){
int b,c,d;
...
}
void fun2(void){
int c,d,e;
...
}
void fun3(void){
int d,e,f;
...
}
```

Dada a seguinte seqüência de chamada e supondo que seja usado o escopo dinâmico, quais são as variáveis são visíveis durante a execução da última função chamada? Inclua, em cada variável visível, o nome da função em que ela foi definida.

(a) main chama fun2; fun2 chama fun3; fun3 chama fun1;

(b) main chama fun1; fun1 chama fun3; fun3 chama fun2;

4. m geral, simulações computacionais precisam de números randômicos. Uma maneira de gerar números pseudo-randômicos é através da seguinte função:

seed(0) = 0
seed(x+1) = (seed(x) + STEP) mod MOD

Por exemplo, se STEP=3 e MOD = 5, a função vai gerar a série de números pseudo-randômicos 0,3,1,2,4,2 em um ciclo de repetições. Podemos implementar esta função da seguinte maneira:

```
#include <stdio.h>
#define STEP 3
#define MOD 5
int rand(){
  static int seed = 0;
  seed = (seed + STEP)%MOD
  return seed;
}

int main(){
  int i;
  for(i=0;i<MOD;i++)
    printf("%d ",rand());

}
```

Explique por que a variável seed precisa ser estática.

5. Considere os seguintes trechos de códigos:

Trecho 1:

```
int indice = 0;
while(indice < MAXLEN && lista[indice] != chave){
  indice++;
}
```

Trecho 2:

```
int indice = 0;
while( lista[indice] != chave && indice < MAXLEN ){
  indice++;
}
```

A variável MAXLEN guarda o tamanho da lista e chave guarda o valor procurado na lista.

(a) Qual é o problema que acontece no Trecho 2?

(b) Por que esse mesmo problema não acontece no Trecho 1?

6. Considere o seguinte problema:

Você vai receber N inteiros ($1 \leq N \leq 10^6$). Cada inteiro x ($1 \leq x \leq 10^9$). Você precisar dizer qual é o elemento com a maior frequencia?

Solução 1:

```
#include <iostream>
#define MAXN 1000000000
using namespace std;
int freq[MAXN];
int main(){
  int n;
  int maior_frequencia = 0;
  int mais_frequente = -1;
  cin >> n;
  for(int i = 0; i < n; i++){
    int x;
    cin >> x;
    freq[x]++;
    if( freq[x] > maior_frequencia ){
      mais_frequente = x;
      maior_frequencia = freq[x];
    }
  }
  printf("%d\n", mais_frequente);
}
```

Solução 2:

```

#include <iostream>
#include <map>
using namespace std;

map <int, int > freq;

int main(){
  int n;
  int maior_frequencia = 0;
  int mais_frequente = -1;
  cin >> n;
  for(int i = 0; i < n; i++){
    int x;
    cin >> x;
    freq[x]++;
    if( freq[x] > maior_frequencia ){
      mais_frequente = x;
      maior_frequencia = freq[x];
    }
  }
  printf("%d\n", mais_frequente);
}
```

Compare as duas soluções com relação ao custo de tempo e memória.

7. Considere o seguinte programa em C:

```
int fun(int *i){
  *i += 5;
  return 4;
}
int main(){
  int x = 3;
  x = x + fun(&x);
  printf("%d\n", x);
}
```

Qual é o valor de x depois da instrução de atribuição em main, supondo que:

(a) os operandos são avaliados da esquerda para a direita.

(b) os operandos são avaliados da direita para a esquerda.

Observe que o compilador pode realizar uma *out-of-order execution* que consiste em reorganizar as instruções que serão executadas pelo processador. Por isso, as funções com efeito colateral podem ser perigosas.

8. Considere o seguinte código:

```

int add(int x, int y)
{
    return x + y;
}

int main()
{
    int x{ 5 };
    int value = add(x, ++x);
    std::cout << value;
    return 0;
}
```

A saída do programa pode ser 11 ou 12. Explique esse possível comportamento desse programa.

9. Considere o seguinte trecho de :

```
a = (c = c + 2) + c++;
```

A operação de atribuição podem ser operandos em expressões e têm como efeito colateral mudar o valor da variável. Listem todos os efeitos colaterais gerados pela expressão acima? Alguma variável na expressão acima, é modificada mais de uma vez? Se sim, qual é o problema que isso pode gerar?

10. As matrizes podem ser classificadas com relação a vinculação ao armazenamento e a vinculação às faixas dos valores dos subscritos em estáticas, dinâmica na pilha com tamanho fixo, dinâmica na pilha com tamanho variável e dinâmica no heap. Classifique os seguintes matrizes em cada declaração:

```
int f1(){
  static int v[10];
}

int f2(){
  int v[10];
}

int f3(int n){
  int v[n];
```

```
}

int f3(){
  int *v;
  v = (int*)malloc(10 * sizeof(int));
}
```

11. Considere os seguintes trechos de código:

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int m[1000][1000];
int main(){
   int i,j;
   clock_t begin = clock();
   for(i=0;i<1000;i++)
     for(j=0;j<1000;j++)
        m[i][j] = i+j;
   double elapsed = ((double) (clock() - begin)) / CLOCKS_PER_SEC;
   printf("EXECUTION TIME(seconds): %8.8f second(s). \n", elapsed );
   //EXECUTION TIME(seconds): 0.00800000 second(s).
   system("PAUSE");
   return 0;
}
```

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int m[1000][1000];
int main(){
    int i,j;
    clock_t begin = clock();
    for(j=0;j<1000;j++)
      for(i=0;i<1000;i++)
        m[i][j] = i+j;
    double elapsed = ((double) (clock() - begin)) / CLOCKS_PER_SEC;
    printf("EXECUTION TIME(seconds): %8.8f second(s). \n", elapsed );
    //EXECUTION TIME(seconds): 0.02800000 second(s).
    system("PAUSE");
    return 0;
}
```

Explique essa diferença no tempo de execução entre os programas.

12. Analise os seguintes códigos : Linguagem C

```
#define max(a,b) a>b?a:b
int c,d;
c = max(2,3);
d = max(2.0,3);
```

Linguagem C++ com sobrecarga de subprogramas

```
int max(int a,int b){ return a>b?a:b; }
double max(double a,double b){ return a>b?a:b; }
int c,d;
c = max(2,3);
d = max(2.0,3); //[Error]call of overloaded ‘max(double, int)’ is ambiguous
```

Linguagem C++ com subprogramas genéricos

```
template<class T> T max(T a ,T b){return a>b?a:b;}
c = max(2,3);
d = max(2.0,3.0);
d = max(2.0,3); //no matching function for call to ‘max(double, int)’
```

(a) Cite uma vantagem e uma desvantagem da utilização de um macro com relação aos subprogramas sobrecarregados?

(b) Cite uma vantagem da utilização de subprogramas genéricos com relação aos subprogramas sobrecarregados.