Datum: Jméno a Příjmení:.....
Předmět: .....PJC

B

1. Co je to tabulka virtuálních metod (VMT)? K čemu slouží? Demonstrujte na vhodném příkladu. [2B]

   Řešení:

   viz přednáška

2. Pokud je v inicializátoru chyba tak ji opravte. [2B]

```
struct X {
      int rem, base;
      X (int i, int j): base(i), rem(base % j) { }
};
```

   Řešení:

Proměnné v inicializátoru se inicializují podle pořadí v jakém jsou deklarovány. Takže je buď nutné prohodit pořadí v deklaraci nebo v inicializaci rem nahradit base i.

3. Předpokládejme, že Point je třída s veřejným kopírujícím konstruktorem. V následujícím fragmentu programu označte řádky na kterých bude volán kopírující konstruktor třídy Point. [3B]

```
Point global;

Point & foo_bar(Point arg)
{
     Point local = arg; ZDE
     Point *heap = new Point(global); ZDE
     *heap = local;
     Point pa[ 4 ] = { local, *heap }; ZDE
     return *heap;
}
```

4. Do následující třídy dopište konstruktor, kopírující konstruktor (uvažujte deep-copy), destruktor a operátor přiřazení. Uvažujte, že vstupem je vždy korektní C-řetězec. K dispozici máte funkce *strcpy(kam, odkud)* a *strlen(c_str)* - vrací délku řetězce bez ukončovací \0 [4B]

```
struct FunnyString {
    FunnyString(const char * c_str) {
```

```
        // dopiste telo konstruktoru
    }
private:
    char *f_string;
};
```

Řešení:

```
FunnyString::FunnyString(const char * c_str) {
     f_string = new char[strlen(c_str)+1);
     strcpy(f_string, c_str);
}

FunnyString & FunnyString::operator=(const FunnyString&
other) {
     if(this == &other) return *this;
     delete [] f_string;
     f_string = new char[strlen(other.f_string)+1);
     strcpy(f_string, other.f_string);
     return *this;
}

FunnyString::FunnyString(const FunnyString& other) {
     f_string = new char[strlen(other.f_string)+1);
     strcpy(f_string, other.f_string);
}

FunnyString::~FunnyString() {
     delete []f_string;
}
```

5. Co bude na výstupu po provedení následujícího programu? [4B]

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

struct A {
     A(int a) { cout << "konstruktor A" << endl;}
     virtual int x() = 0;
     ~A() { cout << "destruktor A" << endl;}

};

struct B : public A {
     B(int a, double b):A(a) {cout << "konstruktor B" << endl;}
     int x() {cout << "X" << endl; }
     ~B() { cout << "destruktor B" << endl; }
};
```

```
void fx(A& ref_a) {};
void fy(const A* p_a) {};
void fz(B a) {};

int main(void) {
      A * x = new B(10, 2.0);
      fx(*x);
      fy(x);
      fz(*(dynamic_cast<B*>(x)));
      delete x;
      return 0;
}
```

Řešení:

```
konstruktor A
konstruktor B
destruktor B
destruktor A
destruktor A
```

6. Navrhněte a implementujte třídu Fibonacci tak, aby šel následující kód přeložit a počítal Fibonacciho posloupnost. [5B]:

```
#include <iostream>
using namespace std;
 int main(void) {
  Fibonacci f1(5);
  Fibonacci f2(3);
  Fibonacci ff = f1 - f2; // ff je jako Fibonacci ff(2)
  Fibonacci ff2 = f2 – f1; // ff2 je jako Fibonacci ff(1), pokud
je 1. operand menší než druhý operand pak minus vrací
Fibonacci(1)

  cout << ff() << endl; //vrati fibonacciho cislo
  cout << ff++ << endl; //zvetsi svoji hodnotu o jedna, vraci
instanci Fibonacci
  cout << ++ff << endl;
}
 Nápověda: F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
```

Řešení:

```
Návrh třídy Fibonacci:
struct Fibonacci {
      int n;
      Fibonacci(int _n):n(_n){}
      Fibonacci operator-(const Fibonacci& other);
      int operator()();
      Fibonacci operator++();
```

```
      Fibonacci operator++(int);
      int count();
}
ostream &operator<<(ostream& out, const Fibonacci& f);
```