Imię i nazwisko:..... **Y2** czas: 60 min

Przyjąć, że udostępniona jest przestrzeń nazw std, std::placeholders i boost

Zadanie 1 (5pkt)

NAME to Twoje nazwisko zapisane wielkimi literami ASCII (zamiast 'Ą' jest 'A'), np. `WROZKA` dla Wróżka.

```
const string NAME = "__________________________________";
```

Funkcja `zad1` jest niestabilna. Proszę usunąć problem poprawiając obiekty `Thread`.

```
void zad1() {
   VV vec;
   vec.push_back(NAME.size());
   vec.push_back(1);
   vec.push_back("5");
   vec.push_back(NAME);
   Thread t1(vec);
   Thread t2(vec);

   try {
      boost::thread thrd1( ref(t1) );
      boost::thread thrd2( ref(t2) );
      thrd1.join();
      thrd2.join();
   } catch(exception&) {
      cout << "error" << endl;
   }
}
```

```
using V = boost::variant<int,string>;
using VV = vector<V>;

class Thread {
public:
  Thread(const VV& tasks) : tasks_(tasks) {}

  void operator()() {
    for( V v : tasks_ ) {

      int i = boost::get<int>(v);

      // doCalculate(i);
      boost::this_thread::sleep_for( boost::chrono::milliseconds(200));
    }
  }
private:
  VV tasks_;
};
```

Zadanie 2 (6pkt)

Szablon `Vector` przechowuje elementy w specjalnym obszarze pamięci, wskazywanym przez składową `buffer`. Popraw implementację szablonu, aby pominąć pętlę w `~Vector`, gdy przechowujemy elementy, dla których wołanie destruktora może być pomijane. Fragment dokumentacji dla `is_trivially_destructible` i `boost::has_trivial_destructor` dodano poniżej. Obie klasy mają identyczne zastosowanie.

```
template < class T > struct std::is_trivially_destructible; (since C++11)
template < class T > struct boost::has_trivial_destructor;
```

If a type has a trivial destructor then the destructor has no effect: calls to the destructor can be safely omitted. `has_trivial_destructor<T>` derives from `true_type`, `has_trivial_destructor<T>::value` provides member equal to `true`, otherwise `has_trivial_destructor<T>` derives from `false_type`, `has_trivial_destructor<T>::value` equal to `false`.

```
template<typename T> class Vector {
public:
   typedef unsigned char BUF_ELEMENT;
   static const int BUFFER_SIZE = 100;

   Vector() : buffer_(new BUF_ELEMENT[BUFFER_SIZE]), size_(0) { }
   ~Vector() {
      for(int i=0;i<size_;++i) {
         T* t = reinterpret_cast<T*>(buffer_ + i*sizeof(T) );
         t->~T(); //wola destruktor
      }
      delete [] buffer_;
   }
   void add(const T& t) {
      new (buffer_ + size_*sizeof(T) ) T(t);
      ++size_;
   }
private:
   BUF_ELEMENT* buffer_;
   int size_;
};
```

Zadanie 3 (3pkt)
NAME to stała zdefiniowana w `Zad1`. Przepisz ją, aby uniknąć pomyłek.

```
const string NAME = "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;
```

Podaj napis, który zostanie wydrukowany przez funkcję `zad3`.

```
template <unsigned n> int fun(int k) { return n - fun<n-1>(k); }
template <> int fun<1>(int k) { return k; }
template <> int fun<0>(int k) { return 0; }
void zad3() {
  cout << fun<0>(NAME.size()) << "␣" << fun<2>(NAME.size()) << "␣" << fun<5>(NAME.size())<< flush;
}
```

Zadanie 4 (6pkt)
Dostarcz funkcję `less_common`, która zwraca liczbę z wektora liczb całkowitych, która występuje tam najmniejszą liczbę razy. Jeżeli kolekcja jest pusta zwracamy 0. Użyj algorytmów z biblioteki standardowej.

```
int less_common(const std::vector<int>& numbers) {
```

Pytanie 1 (1pkt)
Co dają testy jednostkowe?

Pytanie 2 (1pkt)
Ile godzin w semestrze poświęciłeś na przedmiot ZPR (wykład, projekt, kolokwia, nauka własna i inne)

Ile godzin w semestrze poświęciłeś na realizację projektu z ZPR

Pytanie 3 (3pkt)
Zaproponuj zagadnienie, które Twoim zdaniem warto byłoby omówić na przedmiocie ZPR

Notatki / uwagi do prowadzącego