Imię i nazwisko: ..... **Z2** czas: 60 min

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

Zadanie 1 (6pkt)

Uruchomienie funkcji `zad1` prowadzi do blokady. Popraw kod klasy `Calc`, aby ją wyeliminować.

```
const int NUM = 100;
void zad1() {
   Calc p1;
   Calc p2;
   Calc p3;
   p1.setPrev(&p3);
   p2.setPrev(&p1);
   p3.setPrev(&p2);
   boost::thread thrd1( ref(p1) );
   boost::thread thrd2( ref(p2) );
   boost::thread thrd3( ref(p3) );
   thrd1.join();
   thrd2.join();
   thrd3.join();
}
```

```
class Calc {
public:
  Calc() : prev_(nullptr) { out_.push_back(0); out_.push_back(0);}
  void setPrev(Calc* p) { prev_ = p; }

  int get() {
    int ret = -1;
    boost::mutex::scoped_lock lock(m_);
    if(!out_.empty() ) { ret = out_.front(); out_.pop_front();}
    return ret;
  }

  void operator()() {
    for(int i=0;i<NUM;++i) {
      boost::mutex::scoped_lock lock(m_);
      if(prev_) {
        int x = prev_->get();
        if( x >= 0)
          out_.push_back(x + 1);
      }
    }
  }
private:
  Calc* prev_;
  list<int> out_;
  boost::mutex m_;
};
```

Zadanie 2 (6pkt)

Zaimplementować szablon **`fast_copy`**, który kopiuje tablice. Powinien używać `memcpy` gdy przekazano wskaźniki dla typu, który ma trywialny operator przypisania, albo `std::copy`, dla innych typów. Poniżej fragment dokumentacji.

`template <class T> struct boost::has_trivial_assign` bada, czy typ ma trywialny operator przypisania. Jeżeli przypisanie dla typu `T` jest równoznaczne z kopiowaniem pamięci zajmowanej przez obiekt to `has_trivial_assign<T>::type` jest typu `true_type`,
`has_trivial_assign<T>::value` jest typu `bool` i ma wartość `true`, w przeciwnym wypadku
`has_trivial_assign<T>::type` jest typu `false_type`, `has_trivial_assign<T>::value` ma wartość `false`.
`void* memcpy(void* destination, const void* source, size_t num)` kopiuje bloki pamięci, `num` to liczba bajtów.
`template <class In,class Out> Out copy(In first,In last,Out result)` kopiuje zakres wykorzystując operator przypisania.

```
template<class T> void fast_copy(const T* first, const T* last, T* result)
```

Notatki / uwagi do prowadzącego

Zadanie 3 (3pkt)

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

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

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

```
template <unsigned n> int fun(int k) { return n + fun<n-1>(k-1); }
template <> int fun<0>(int k) { return k; }

void zad3() {
   cout << fun<1>(NAME.size()) << "␣" << fun<2>(NAME.size()) << "␣" << fun<5>(NAME.size()) << endl;
}
```

Zadanie 4 (4pkt)

Podaj napis, który zostanie wydrukowany przez funkcję `zad4`.

```
void zad4() {
   using Graph = boost::adjacency_list<boost::vecS, boost::vecS, boost::bidirectionalS>;
   using Edge = pair<int, int>;
   static const int NUM_VERTICES = 5;
   const Edge EDGES[] = { Edge(0,1), Edge(1,2), Edge(1,3), Edge(2,3), Edge(3,4) };
   const int NUM_EDGES = sizeof(EDGES)/sizeof(EDGES[0]);
   Graph g(EDGES, EDGES + NUM_EDGES, NUM_VERTICES);
   int m = 0;
   for_each( vertices(g).first, vertices(g).second,
           [&](int idx){
              int num_edges = out_edges(idx, g).second - out_edges(idx, g).first;
              if(num_edges > m)
                 m = num_edges;
           });
   cout << m << endl;
}
```

Zadanie 5 (4pkt)

Dostarcz funkcję `count_common`, która analizuje wektor napisów i zwraca ile razy występuje najczęstszy napis. Jeżeli kolekcja jest pusta zwracamy zero. Przykładowo `count_common(vector<string>{"ALA","ELA","ALA"})` zwraca 2. Użyj algorytmów z biblioteki standardowej.

```
int count_common(const std::vector<string>& words)
```

Pytanie 1 (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 2 (1pkt)

Zaproponuj zagadnienie, które Twoim zdaniem warto byłoby omówić na przedmiocie ZPR