Zadanie A
Operacje zbiorowe

Punktów do uzyskania: **10**

Generalia

- Zadanie polega na implementacji zestawu podprogramów obsługi zbiorów w uniwersum obejmującym pięcioelementowe ciągi znaków `0` lub `1`.
- Zbiór jest reprezentowany pojedynczą daną typu **`int`**.
- Porządek zbiorów określają reguły:
  - Zbiór o większej liczności zawsze jest większy od zbioru o mniejszej liczności.
  - Dla zbiorów o równej liczności większy jest zbiór poprzedzający w odwrotnej kolejności leksykograficznej elementów.
- Kod źródłowy rozwiązania musi być rozdzielony na wiele plików o podanych dalej nazwach i zawartości.
- Wysyłane pliki źródłowe rozwiązania muszą być spakowane do jednego pliku programem *`Zip`*.
- Każdy plik źródłowy musi w pierwszej linii zawierać komentarz z imieniem i nazwiskiem autora.

Wymagane pliki i podprogramy

- Plik `emplace.cpp` zawieraja kod podprogramu o nagłówku:

  `void` `Emplace ( char*, int* );`

  Podprogram na podstawie przekazywanego pierwszym argumentem dowolnie długiego ciągu znakowego obejmującego wyłącznie spacje lub pięcioznakowe spójne sekwencje znaków `0` lub `1` wyznacza według własnej implementacji zbiór z odniesieniem przekazanym drugim argumentem.
- Plik `insert.cpp` zawiera kod podprogramu o nagłówku:

  `void` `Insert ( char*, int* );`

  Podprogram elementy przekazane pierwszym argumentem o warunkach jak dla procedury `Emplace` wstawia do zbioru przekazanego drugim argumentem.
- Plik `erase.cpp` zawiera kod podprogramu o nagłówku:

  `void` `Erase ( char*, int* );`

  Podprogram elementy przekazane pierwszym argumentem o warunkach jak dla procedur `Emplace` oraz `Insert` usuwa ze zbioru przekazanego drugim argumentem.
- Plik `print.cpp` zawiera kod podprogramu o nagłówku:

  `void` `Print ( int, char* );`

  Podprogram zawartość zbioru określanego pierwszym argumentem przekazuje do ciągu znakowego danego drugim argumentem, w postaci pięcioelementowych sekwencji znaków `0` lub `1` z następującą spacją i w malejącej kolejności leksykograficznej elementów. Długość danego ciągu znakowego jest minimalna dla poprawnego działania, a zbiór pusty jest opisany słowem `empty`.
- Plik `emptiness_nonemty.cpp` zawiera kody podprogramów:
  - **`bool`** `Emptiness ( int );`

    Zwracającego wartość logiczną pustości zbioru określonego argumentem.
  - **`bool`** `Nonempty ( int );`

    Zwracającego wartość logiczną niepustości zbioru określonego argumentem.
- Plik `member.cpp` zawiera kod podprogramu o nagłówku:

  `bool` `Member ( char*, int );`

  Podprogram zwraca wartość logiczną przynależności elementu przekazanego pierwszym argumentem w postaci dowolnie długiego ciągu znakowego obejmującego wyłącznie spacje oraz dokładnie jedną pięcioznakową spójną sekwencję znaków `0` lub `1` w zbiorze określonym drugim argumentem.
- Plik `disjoint_conjunctive.cpp` zawiera kody podprogramów:
  - **`bool`** `Disjoint ( int, int );`

    Zwracającego wartość logiczną rozłączności zbiorów określanych argumentami.
  - **`bool`** `Conjunctive ( int, int );`

    Zwracającego wartość logiczną niepustości przecięcia zbiorów określonych argumentami.
- Plik `inclusion_equality.cpp` zawiera kody podprogramów:
  - **`bool`** `Inclusion ( int, int );`

    Zwracającego wartość logiczną zawierania zbioru określonego pierwszym argumentem w zbiorze określonym drugim argumentem.
  - **`bool`** `Equality ( int, int );`

    Zwracającego wartość logiczną równości zbiorów określanych argumentami.
- Plik `operations.cpp` zawiera kody podprogramów:
  - **`void`** `Union ( int, int, int* );`

    Wyznaczającego sumę mnogościową zbiorów określonych dwoma pierwszymi argumentami przekazywaną do zbioru określonego trzecim argumentem.
  - **`void`** `Intersection ( int, int, int* );`

    Wyznaczającego iloczyn mnogościowy zbiorów określonych dwoma pierwszymi argumentami przekazywany do zbioru określonego trzecim argumentem.
  - **`void`** `Symmetric ( int, int, int* );`

    Wyznaczającego różnicę symetryczną zbiorów określonych dwoma pierwszymi argumentami przekazywaną do zbioru określonego trzecim argumentem.
  - **`void`** `Difference ( int, int, int* );`

    Wyznaczającego różnicę mnogościową zbioru określonego pierwszym argumentem i zbioru określonego drugim argumentem przekazywaną do zbioru określonego trzecim argumentem.
  - **`void`** `Complement ( int, int* );`

    Wyznaczającego dopełnienie mnogościowe zbioru określonego pierwszym argumentem przekazywane do zbioru określonego drugim argumentem.
- Plik `cardinality.cpp` zawiera kod podprogramu o nagłówku:

  `int` `Cardinality ( int );`

  zwracającego moc zbioru danego argumentem.
- Plik `relations.cpp` zawiera kody podprogramów:
  - **`bool`** `LessThen ( int, int );`

    Zwracającego wartość logiczną silnej mniejszości zbioru określonego pierwszym argumentem względem zbioru określonego drugim argumentem.
  - **`bool`** `LessEqual ( int, int );`

    Zwracającego wartość logiczną słabej mniejszości zbioru określonego pierwszym argumentem względem zbioru określonego drugim argumentem.
  - **`bool`** `GreatEqual ( int, int );`

    Zwracającego wartość logiczną słabej większości zbioru określonego pierwszym argumentem względem zbioru określonego drugim argumentem.
  - **`bool`** `GreatThen ( int, int );`

    Zwracającego wartość logiczną silnej większości zbioru określonego pierwszym argumentem względem zbioru określonego drugim argumentem.

Dodatkowe uwarunkowania

- W rozwiązaniu można założyć dostępność pliku nagłówkowego o nazwie `bitwise_operations.h` zawierającego nagłówki wszystkich wymienionych wyżej podprogramów.
- Kod rozwiązania może stosować wyłącznie wiedzę przedstawioną na wykładzie, ale zarazem nie może stosować:
  - Znaków kwadratowych nawiasów i ich równoważników.
  - Słów kluczowych pętli, czyli słów **`for`**, **`while`** oraz **`goto`**.
  - Rekordów, czyli słów kluczowych **`struct`** oraz **`class`**.
  - Sekwencji `string`.
  - Pamięci dynamicznej.
  - Typów własnych zmiennych innych niż **`int`**.
  - Własnych identyfikatorów zaczynających się znakiem podkreślenia.