Требования к программам

1. В программе должны быть реализованы следующие структуры данных:

   - Контейнер данных объектов типа `record`:

```
# ifndef record_H
# define record_H
# include <memory>
# include <stdio.h>
# include "condition.h"

enum class read_status
{
  success = 0,
  eof = -1,
  format = -2,
  memory = -3,
};

class record
{
  private:
    std::unique_ptr<char []> name = nullptr;
    int phone = 0;
    int group = 0;
  public:
    record () = default;
   ~record () = default;
    char  * get_name () const { return name.get (); }
    int get_phone () const { return phone; }
    int get_group () const { return group; }
    int init (const char *n, int p, int g);
    // Allow as return value for functions
    record (record &&x) = default; // move constructor
    // Assignement move operator
    record& operator= (record&& x) = default;
    // Prohoibit pass by value
    // (it is default when move constructor is declared)
    record (const record &x) = delete;
    // Prohoibit assignement operator
    // (it is default when move constructor is declared)
    record& operator= (const record&) = delete;
    // Check condition 'x' for field 'name' for 'this' and 'y'
    bool compare_name (condition x, const record& y) const;
    // Check condition 'x' for field 'phone' for 'this' and 'y'
    bool compare_phone (condition x, const record& y) const;
    // Check condition 'x' for field 'group' for 'this' and 'y'
    bool compare_group (condition x, const record& y) const;
    void print (FILE * fp = stdout);
    read_status read (FILE *fp = stdin);
};
# endif
```

Функции сравнения в этом классе сравнивают одно из полей класса с соответствующим полем класса `y` согласно условию, заданному аргументом `x`.

- Enum class, задающий условия для полей:

```
# ifndef condition_H
# define condition_H

enum class condition
{
  none, // not specified
  eq,   // equal
  ne,   // not equal
  lt,   // less than
  gt,   // less than
  le,   // less equal
  ge,   // great equal
  like, // strings only: match pattern
};
# endif
```

- Класс, задающий условие для проверки:

```
# ifndef command_H
# define command_H

# include <stdio.h>
# include "record.h"

class command : public record
{
  private:
    condition c_name  = condition::none;
    condition c_phone = condition::none;
    condition c_group = condition::none;
  public:
    command () = default;
   ~command () = default;
    // Convert string command to data structure
    // Example 1: "phone = 1234567" parsed to
    //   (command::phone = 1234567, command::c_phone = condition::eq)
    //   other fields are unspecified
    // Example 2: "name like St%" parsed to
    //   (command::name = "St%", command::c_name = condition::like)
    //   other fields are unspecified
    bool parse (const char * string);
    // Print parsed structure
    void print (FILE *fp = stdout) const;
    // Apply command, return comparision result for record 'x'
    bool apply (const record& x) const;
};

# endif
```

2. Пример реализации некоторых функций из класса `record`:

```
# include <string.h>
# include <stdio.h>
# include "record.h"

# define LEN 1234
using namespace std;

int record::init (const char *n, int p, int g)
{
  phone = p;
  group = g;
  if (n)
    {
      name = std::make_unique<char []> (strlen (n) + 1);
      if (!name) return -1;
      strcpy (name.get(), n);
    }
  else
    name = nullptr;
  return 0;
}

read_status record::read (FILE *fp)
{
  char buf[LEN];
  name = nullptr;
  if (fscanf (fp, "%s%d%d", buf, &phone, &group) != 3)
    {
      if (feof(fp)) return read_status::eof;
      return read_status::format;
    }
  if (init (buf, phone, group))
    return read_status::memory;
  return read_status::success;
}

// Check condition 'x' for field 'phone' for 'this' and 'y'
bool record::compare_phone (condition x, const record& y) const
{
  switch (x)
  {
    case condition::none: // not specified
      return true; // unspecified opeation is true
    case condition::eq:   // equal
      return phone == y.phone;
    case condition::ne:   // not equal
      return phone != y.phone;
    case condition::lt:   // less than
      return phone < y.phone;
    case condition::gt:   // less than
```

```
      return phone > y.phone;
    case condition::le:   // less equal
      return phone <= y.phone;
    case condition::ge:   // great equal
      return phone >= y.phone;
    case condition::like: // strings only: match pattern
      return false; // cannot be used for phone
  }
  return false;
}
```

3. **Задача программы**:
   - Построить **двунаправленный список** объектов типа `record` и считать его из указанного файла (аргумент командной строки)
   - Считывать команды поиска в этом списке **по одной со стандартного ввода (stdin)**, до тех пор пока команды не закончатся
   - Применять команду поиска к списку и выводить **только найденные (т.е. удовлетворяющие условию)** элементы в **стандартный вывод (stdout)**
4. Все команды поиска задают **одно условие на одно поле** записи `record`. Условия могут быть:
   - <поле> <оператор> <выражение>, где
     - <поле> – имя поля (name, phone, value)
     - <оператор> – логический оператор отношения: = – равно, <> – не равно, <, >, <=, >= – соответствуют языку C
     - <выражение> – константное выражение соответствующего типа
   - <поле> like <образец> где
     - <поле> – имя поля символьного типа (т.е. только name)
     - <образец> – образец поиска. Может включать в себя специальные символы:
       - '_' – соответствует 1 любому символу, а символы "_" и "\\" соответствуют литеральным символам "_" и "\"
       - '%' – соответствует 0 или более любым символам, а символы "\%" и "\\" соответствуют литеральным символам "%" и "\"
       - '[n–m]' (n, m – символы) – соответствует 1 любому символу, имеющему код в диапазоне $n \ldots m$, а символы "\[", "\]" и "\\" соответствуют литеральным символам "[", "]" и "\"
       - '[^n–m]' (n, m – символы) соответствует любому символу, имеющему код, не содержащийся в диапазоне $n \ldots m$, а символы "\[", "\]", "\^" и "\\" соответствуют литеральным символам "[", "]", "^" и "\"

   Примеры команд:
   - `phone >= 1234567` – вывести все элементы списка, у которых поле `phone` больше или равно 1234567.
   - `group = 208` – вывести все элементы списка, у которых поле `group` равно 208.
   - `name like St%` – вывести все элементы списка, у которых поле `name` соответствует образцу поиска `"St%"`.

5. Программа должна получать все параметры в качестве аргументов командной строки и стандартного ввода. Аргументы командной строки:

   1) `filename` – имя файла, откуда надо прочитать список.

   Например, запуск

   ```
   cat commands.txt | ./a.out a.txt > result.txt
   ```

   означает, что файл `commands.txt` подается на стандартный ввод, список надо прочитать из файла `a.txt`, а результаты будут перенаправлены со стандартного вывода в файл `result.txt`.

6. Класс "список" должен содержать функцию ввода списка из указанного файла.

7. Ввод списка из файла. В указанном файле находится дерево в формате:

Слово-1	Целое-число-1	Целое-число-2
Слово-2	Целое-число-3	Целое-число-4
…	…	
Слово-n	Целое-число-2n-1	Целое-число-2n

   где слово – последовательность алфавитно-цифровых символов без пробелов. Длина слова неизвестна, память под него выделяется динамически. Все записи в файле различны (т.е. нет двух, у которых совпадают все 3 поля). Концом ввода считается конец файла. Программа должна выводить сообщение об ошибке, если указанный файл не может быть прочитан или содержит данные неверного формата.

8. Вывод результата работы функции в функции `main` должен производиться по формату:

   ```
   printf ("%s : Result = %d Elapsed = %.2f\n",
           argv[0], res, t);
   ```

   где

   - `argv[0]` – первый аргумент командной строки (имя образа программы),
   - `res` – общее количество найденных элементов списка,
   - `t` – время работы на все команды.

   Вывод должен производиться в точности в таком формате, чтобы можно было автоматизировать обработку запуска многих тестов.

Задачи

Требуется написать программу, которая

- строит **двунаправленный список** объектов типа `record` и считывает его из указанного файла (аргумент командной строки)
- считывает команды поиска в этом списке **по одной со стандартного ввода (stdin)**, до тех пор пока команды не закончатся
- для каждой прочитанной команды применяет эту команду поиска к списку и выводит **только найденные (т.е. удовлетворяющие условию)** элементы в **стандартный вывод (stdout)**.

Задание оценивается по качеству реализованной системы классов и успешности реализации следующих элементов:

1. Поддержать условие вида <поле> <оператор> <выражение>
2. Поддержать условие вида <поле> like <образец> со специальным символом '_'
3. Поддержать условие вида <поле> like <образец> со специальным символом '%'
4. Поддержать условие вида <поле> like <образец> со специальными символами '[$n-m$]'
5. Поддержать условие вида <поле> like <образец> со специальными символами '[$\hat{}n-m$]'