Лабораторна робота №25.
Перевантаження.

- **Вимоги**
- **Розробник**
  - Гладков Костянтинт Сергіович
  - Студент групи КІТ-320;
  - 01-march-2021.
- **Основне завдання**

Лабораторна робота №25. ООП. Перевантаження операторів

Загальне завдання.

Поширити попередню лабораторну роботу таким чином:

- у базовому класі, та класі/класах-спадкоємцях перевантажити:
  - оператор присвоювання;
  - оператор порівняння (на вибір: == , < , > , >= , <= , !=);
  - оператор введення / виведення;
- у класі-списку перевантажити:
  - оператор індексування ([ ]);
  - введення / виведення з акцентом роботи, у тому числі і з файлами. При цьому продовжувати використовувати регулярні вирази для валідації введених даних.

Додаткові умови виконання завдання:

- продемонструвати відсутність витоків пам'яті;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- не використовувати конструкцію "using namespace std;", замість цього слід робити "using" кожного необхідного класу: using std::string, using std::cout;
- у проекті не повинні використовуватися бібліотеки введення / виведення мови С, а також не повинні використовуватися рядки типу `char*`.

- **Опис роботи**

Програма працює за тим же принципом, що і колишні, однак використовує ООП, полегшуючи роботу. А використання перевантаження операторів допомагає нам ще сильніше спростити роботу.

- **Функціональне призначення**

Дана програма може бути використана для маніпуляцій з нашим динамічним масивом, який має в собі класи. Таким чином ми отримуємо великий функціонал на увазі більше кількості можливих методів для роботи з нашим масивом.

- **Опис логічної структури**

Данна програма працює за принципом створення класу, в якому зберігається динамічний масив іншого класу, а також за допомогою методів, які контролюється цей масив.

Функція main містить в собі виклик функцій, а також корисних властивостей для функцій

class List - клас, який зберігає в собі динамічний масив іншого класу, а також які всередині себе має всі методи роботи з даним класом.

class Agency - базовий клас.

void List::expand() - метод, який використовується для розширення обсягу нашого динамічного масиву

void List::add_ag - метод, який використовується для розширення нашого масиву, а точніше додавання в даній оновлений масив нового елемента за рахунок спеціального коструктора

Agency& List::get_ag - метод, який дозволяє отримати елемент масиву за індексом.

void List::remove_last_ag() - метод, який прибирає останнім агенство з масиву.

int List::get_oldest_ag() - метод, який дозволяє знайти самий старший елемент в масиві (індекс).

ostream& operator<< - перевантаження

istream& operator>> - перевантаження

string List::operator[] - перевантаження

Agency& operator = - перевантаження

bool operator == - перевантаження

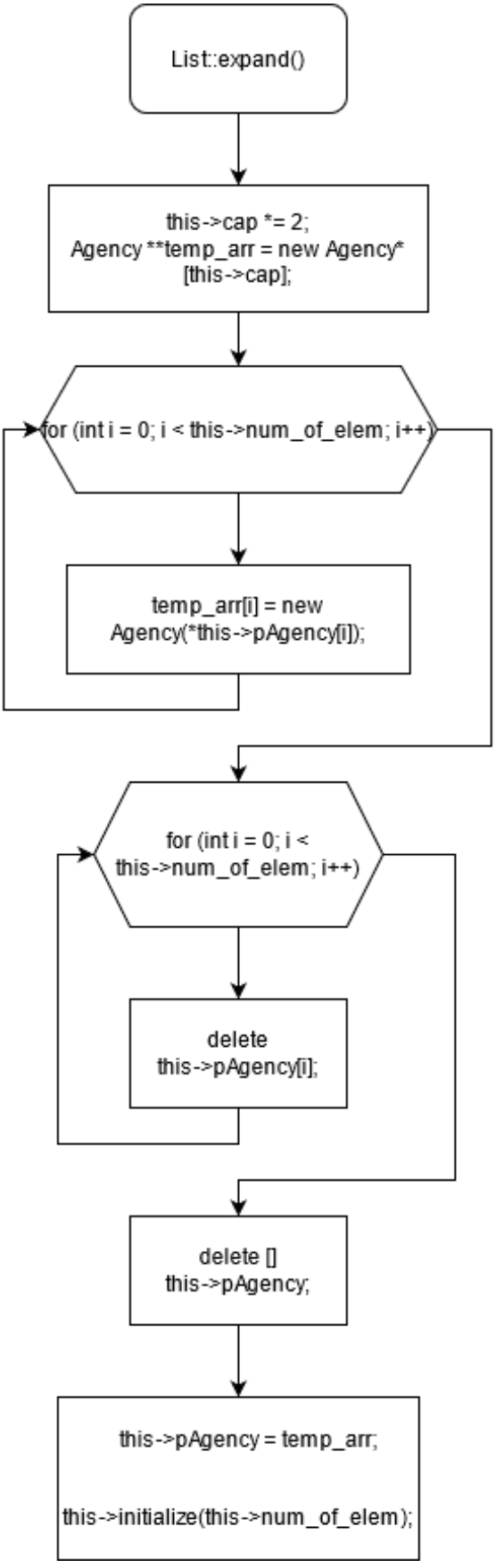

рис. 1 - void List::expand()

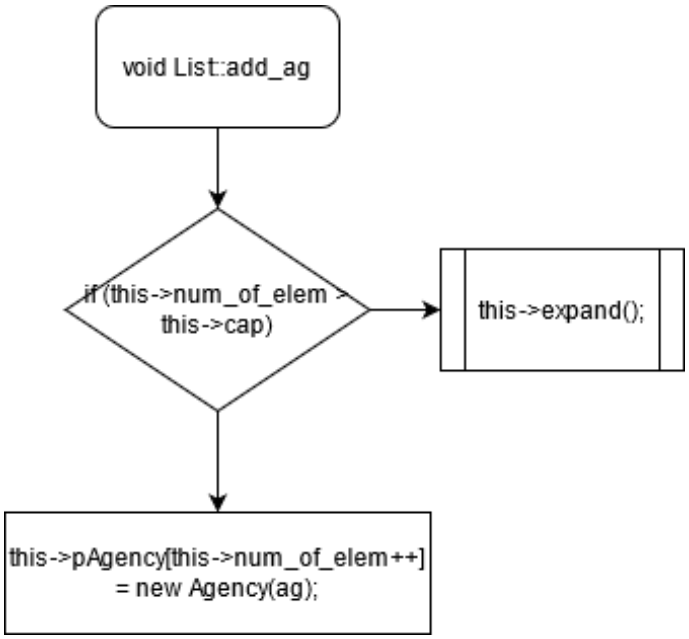

рис. 2 - void List::add_ag

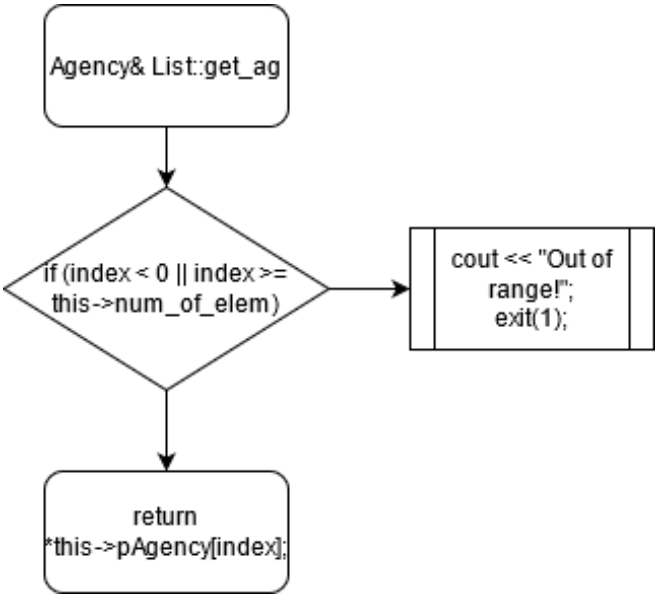

рис. 3 - Agency& List::get_ag

рис. 4 - void List::remove_last_ag()

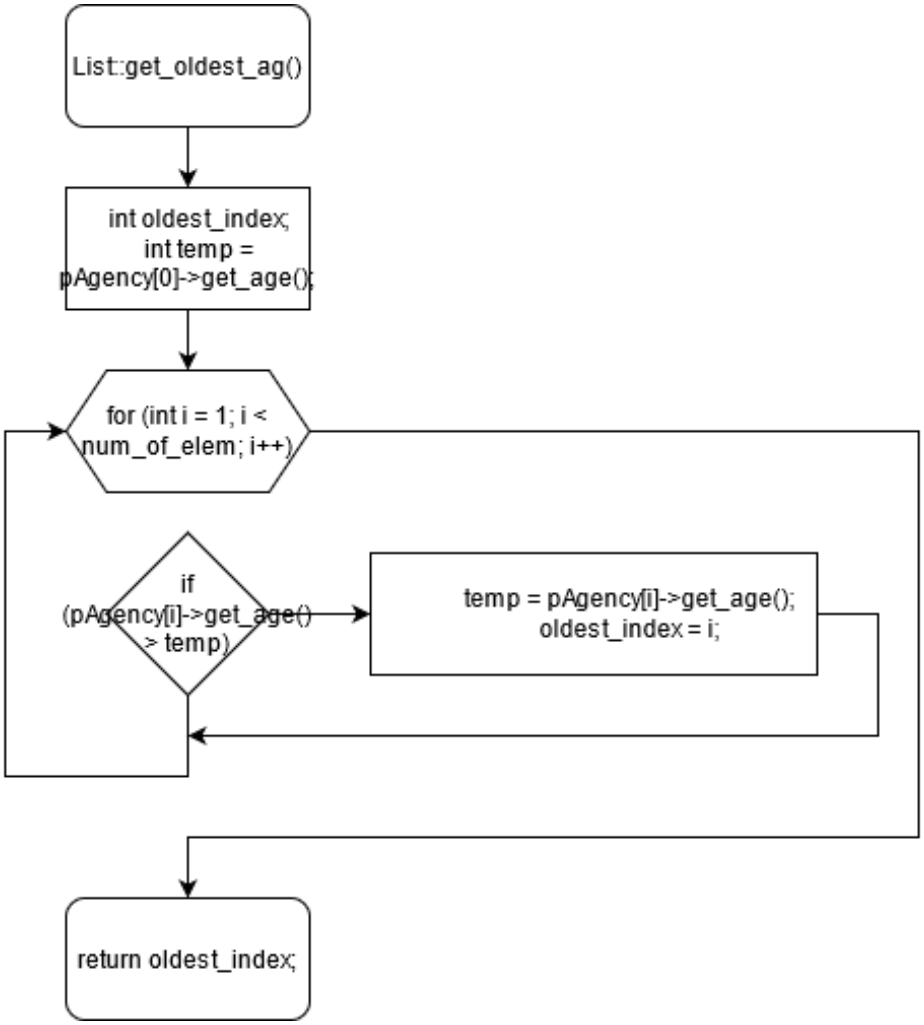

рис. 5 - int List::get_oldest_ag()

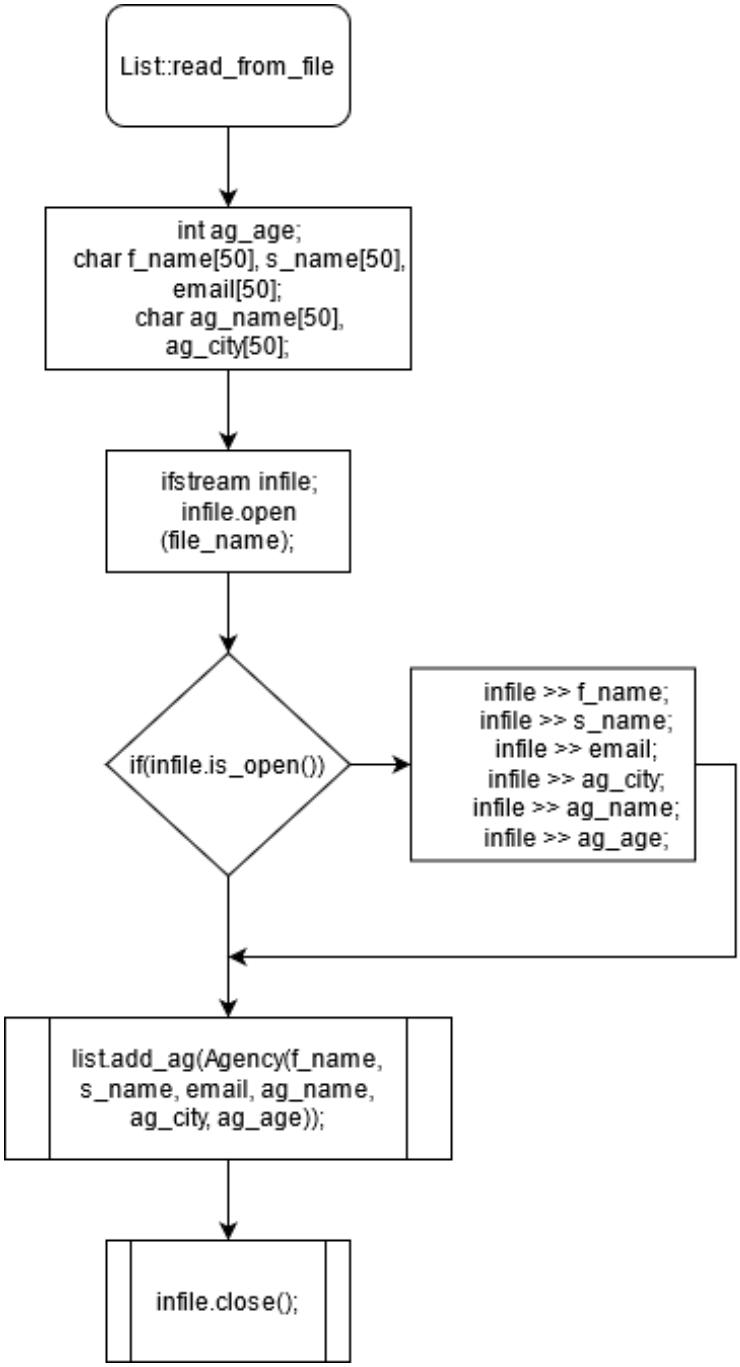

рис. 6 - List::read_from_file

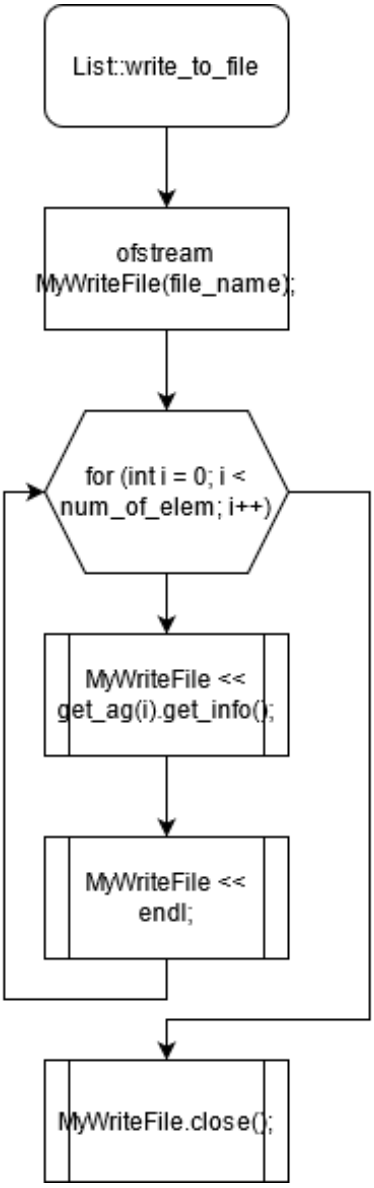

рис. 7 - List::write_to_file

```
Agency& operator = (const Agency &ag)
{
    strcpy(this->agency_boss.first_name, ag.agency_boss.first_name);
    strcpy(this->agency_boss.second_name, ag.agency_boss.second_name);
    strcpy(this->agency_boss.email, ag.agency_boss.email);
    strcpy(this->agency_name, ag.agency_name);
    strcpy(this->servicing_city, ag.servicing_city);
    this->agency_age=ag.agency_age;

    return *this;
}

bool operator == (const Agency &ag)
{
    return (
            strcmp(this->agency_boss.first_name, ag.agency_boss.first_name) == 0 &&
            strcmp(this->agency_boss.second_name, ag.agency_boss.second_name) == 0&&
            strcmp(this->agency_boss.email, ag.agency_boss.email) == 0 &&
            strcmp(this->agency_name, ag.agency_name) == 0 &&
            strcmp(this->servicing_city, ag.servicing_city) == 0 &&
            this->agency_age == ag.agency_age);
}
```

рис. 8 - перевантаження "=" та "=="

```
ostream& operator<< (std::ostream &out, const Agency &ag)
{
    out << "\nAgency boss first name: " << ag.agency_boss.first_name <<
        "\nAgency boss second name: " << ag.agency_boss.second_name <<
        "\nAgency boss email: " << ag.agency_boss.email <<
        "\nAgency name: " << ag.agency_name <<
        "\nAgency city: " << ag.servicing_city <<
        "\nAgency age: " << ag.agency_age;

    ofstream MyWriteFile("/home/blank/xd.txt");

    MyWriteFile << ag.agency_boss.first_name << endl;
    MyWriteFile << ag.agency_boss.second_name << endl;
    MyWriteFile << ag.agency_boss.email << endl;
    MyWriteFile << ag.agency_name << endl;
    MyWriteFile << ag.servicing_city << endl;
    MyWriteFile << ag.agency_age << endl;

    MyWriteFile.close();

    return out;
}

istream& operator>> (std::istream &in, Agency &ag)
{

    in >> ag.agency_boss.first_name;
    in >> ag.agency_boss.second_name;
    in >> ag.agency_boss.email;
    in >> ag.agency_name;
    in >> ag.servicing_city;
    in >> ag.agency_age;

    return in;
}
```

рис. 8 - перевантаження "<<" та ">>"

- **Важливі елементи програми**

```
List my_list;

my_list.add_ag(Agency("Kostya", "Gladkov", "Email", "help", "London", 10));
my_list.add_ag(Agency("Someone", "Noone", "Email", "nope", "London", 15));

for (int i = 0; i < my_list.get_num(); i++){
    cout << my_list.get_ag(i).get_info() << "\n";
}
```

```
this->cap *= 2;

Agency **temp_arr = new Agency*[this->cap];

for (int i = 0; i < this->num_of_elem; i++)
{
    temp_arr[i] = new Agency(*this->pAgency[i]);
}

for (int i = 0; i < this->num_of_elem; i++)
{
    delete this->pAgency[i];
}
delete [] this->pAgency;

this->pAgency = temp_arr;

this->initialize(this->num_of_elem);
```

```
Agency to_input;
    cin >> to_input;

    cout << "\nYou entered: " << to_input << '\n';
```

- **Варіанти використання**

Дана програма може бути використана для роботи з базовим класом, а також класом-списком, який буде зберігати в собі динамічний масив базового класу.

Висновки

У даній лабораторній роботі були придбані знання роботи з ООП. Також були отримані знання роботи з перевантаженістю операторів