Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет информационных технологий и программирования

Лабораторная работа № 2

Классы

Вариант 6

Выполнил студент группы № M3111
Гонтарь Тимур Сергеевич
Подпись:

Санкт-Петербург
2023

Условие ЛР:

Согласно варианту описать указанные классы (варианты распределяются преподавателем лично). Написать программу, использующую
описанные классы: инициализация переменных (ввод пользователя), выполнение действий с экземплярами класса (в зависимости от
дальнейшего ввода пользователя).

Решение:

subset.h – header файл для класса подмножества целых чисел

```
#ifndef LAB2_SUBSET_H
#define LAB2_SUBSET_H

class Subset {
private:
    int maxsize;
    int cursize;
    int *dynset;
public:
    Subset();

    Subset(int);

    Subset(Subset &);

    ~Subset();

    Subset &operator=(const Subset &);

    bool checkInSubset(int);

    bool pushElem(int);

    bool popElem(int);

    Subset *intersectSubset(Subset &);

    Subset *unionSubset(Subset &);

    void addFromAnotherSubset(Subset &);

    void removeFromAnotherSubset(Subset &);

    void printSubset();

};


#endif //LAB2_SUBSET_H
```

subset.cpp – реализация класса подмножества

```
#include "subset.h"

#include <iostream>
#include <algorithm>
```

```
using std::cout;
using std::copy_n;

//default constructor
Subset::Subset() {
    maxsize = 0;
    cursize = 0;
    dynset = nullptr;
};

//constructor with maximum size
Subset::Subset(int mx) {
    maxsize = mx;
    cursize = 0;
    dynset = new int[mx];
}

//constructor of copying
Subset::Subset(Subset &s) {
    maxsize = s.maxsize;
    cursize = s.cursize;
    if (s.dynset != nullptr) {
        dynset = new int[s.maxsize];
        std::copy_n(s.dynset, s.maxsize, dynset);
    } else {
        dynset = nullptr;
    }
}

//destructor
Subset::~Subset() {
    delete[] dynset;
};

//overloading assignment operator with copying
Subset &Subset::operator=(const Subset &s) {
    if (this == &s) {
        return *this;
    }
    maxsize = s.maxsize;
    cursize = s.cursize;
    delete[] dynset;
    if (s.dynset != nullptr) {
        dynset = new int[s.maxsize];
        copy_n(s.dynset, s.maxsize, dynset);
    } else {
        dynset = nullptr;
    }

    return *this;
}

//check if integer n is in subset
bool Subset::checkInSubset(int n) {
    if (dynset == nullptr) {
        return false;
    } else {
        bool flag = false;
        for (int i = 0; i < cursize; i++) {
            if (dynset[i] == n) {
                flag = true;
                break;
            }
```

```
        }

        return flag;
    }
}

//push element n to the end of subset
bool Subset::pushElem(int n) {
    if (cursize == maxsize or checkInSubset(n)) {
        return false;
    } else {
        cursize += 1;
        dynset[cursize - 1] = n;
        return true;
    }
}

//pop element n from subset
bool Subset::popElem(int n) {
    if (cursize == 0 or !checkInSubset(n)) {
        return false;
    } else {
        int it = 0;
        while (it < cursize) {
            if (dynset[it] == n) {
                break;
            } else {
                it++;
            }
        }
        cursize -= 1;
        for (int i = it; i < cursize; i++) {
            dynset[i] = dynset[i + 1];
        }
        return true;
    }
}

//find the intersection of 2 subsets
Subset *Subset::intersectSubset(Subset &s) {
    int newmax = std::max(maxsize, s.maxsize);
    Subset *intersection = new Subset(newmax);

    for (int i = 0; i < cursize; i++) {
        if (s.checkInSubset(dynset[i])) {
            intersection->pushElem(dynset[i]);
        }
    }

    return intersection;
}

//find the union of 2 subsets
Subset *Subset::unionSubset(Subset &s) {
    int newmax = maxsize + s.maxsize;
    Subset *united = new Subset(newmax);

    for (int i = 0; i < cursize; i++) {
        if (!united->checkInSubset(dynset[i])) {
            united->pushElem(dynset[i]);
        }
    }

    for (int i = 0; i < s.cursize; i++) {
```

```
        if (!united->checkInSubset(s.dynset[i])) {
            united->pushElem(s.dynset[i]);
        }
    }

    return united;
}

//add all elements from another subset
void Subset::addFromAnotherSubset(Subset &s) {
    int newmax = maxsize + s.maxsize;
    int *newdynset = new int[newmax];
    std::copy_n(dynset, maxsize, newdynset);
    delete[] dynset;
    dynset = newdynset;

    for (int i = 0; i < s.cursize; i++) {
        pushElem(s.dynset[i]);
    }
}


//remove all elements from another subset
void Subset::removeFromAnotherSubset(Subset &s) {
    for (int i = 0; i < s.cursize; i++) {
        popElem(s.dynset[i]);
    }
}



//print the subset
void Subset::printSubset() {
    if (dynset == nullptr) {
        cout << "Subset is empty";

        return;
    } else {
        cout << "Subset: ";
        for (int i = 0; i < cursize; i++) {
            cout << dynset[i] << " ";
        }
        cout << "\n";

        return;
    }
}
```

menu.h – header файл для класса меню

```
#include "subset.h"

#ifndef LAB2_MENU_H
#define LAB2_MENU_H

class Menu {
private:
    Subset *currentSubset;
public:
    Menu();

    void menuInterface();
```

```
    Subset *createSubset();

    void checkElement();

    void addElement();

    void popElement();

    void intersectSubsets();

    void uniteSubsets();

    void addFromAnother();

    void rmFromAnother();

    void printSubset(Subset *);

    void stopProgram();
};

#endif //LAB2_MENU_H
```

menu.cpp – реализация класса меню

```
#include "subset.h"
#include "menu.h"

#include <iostream>


using std::cout;
using std::cin;
using std::endl;

//default constructor
Menu::Menu() {
    currentSubset = nullptr;
}

//create a subset with a maximum number of elements given
Subset *Menu::createSubset() {
    int mx;
    cout << "Write the maximum number of elements in a subset: " << endl;
    cin >> mx;
    while (mx <= 0) {
        cout << "The maximum number of elements should be positive" << endl;
        cin >> mx;
    }

    Subset *temp = new Subset(mx);

    int cnt;
    cout << "Enter the number of elements to push in new subset " << endl;
    cin >> cnt;
    while (cnt <= 0 or cnt > mx) {
        cout << "The number shouldn't be negative and should be less then
maxsize" << endl;
        cin >> cnt;
    }

    cout << "Enter the integers you want to add into subset: " << endl;
```

```
    for (int i = 0; i < cnt; i++) {
        int val;
        cin >> val;
        temp->pushElem(val);
    }

    return temp;
}

//print the subset
void Menu::printSubset(Subset *s) {
    if (s == nullptr) {
        cout << "The subset doesn't exist" << endl;
    } else {
        s->printSubset();
    }
}

//add the element
void Menu::addElement() {
    int val;
    cout << "Enter a value you want to add: " << endl;
    cin >> val;
    if (currentSubset == nullptr or !currentSubset->pushElem(val)) {
        cout << "Cannot add a value: the subset isn't created or it's
overloaded or the element is already in subset"
             << endl;
    } else {
        cout << "Success" << endl;
    }
}

//pop the element
void Menu::popElement() {
    int val;
    cout << "Enter a value you want to pop: " << endl;
    cin >> val;
    if (currentSubset == nullptr or !currentSubset->popElem(val)) {
        cout << "Cannot pop a value: the subset isn't created or it's empty
or the element isn't in subset" << endl;
    } else {
        cout << "Success" << endl;
    }
}

//check the element
void Menu::checkElement() {
    int val;
    cout << "Enter a value you want to check: " << endl;
    cin >> val;
    if (currentSubset->checkInSubset(val)) {
        cout << "The element " << val << " is in the subset" << endl;
    } else {
        cout << "The element " << val << " is not in the subset" << endl;
    }
}

//intersect two subsets: current and the new one
void Menu::intersectSubsets() {
    cout << "First, give a Subset to intersect with current." << endl;
    Subset *temp = createSubset();
    Subset *intersect = currentSubset->intersectSubset(*temp);
    cout << "The intersection of current subset and new subset:" << endl;
    printSubset(intersect);
```

```
    delete temp;
    delete intersect;
}

//unite two subsets: current and the new one
void Menu::uniteSubsets() {
    cout << "First, give a Subset to unite with current." << endl;
    Subset *temp = createSubset();
    Subset *unite = currentSubset->unionSubset(*temp);
    cout << "The union of current subset and new subset:" << endl;
    printSubset(unite);

    delete temp;
    delete unite;
}

//add all elements from new subset to current
void Menu::addFromAnother() {
    cout << "First, give a Subset, which will be added to current." << endl;
    Subset *temp = createSubset();
    currentSubset->addFromAnotherSubset(*temp);
    cout << "The result:" << endl;
    printSubset(currentSubset);

    delete temp;
}

//remove all elements of new subset from current
void Menu::rmFromAnother() {
    cout << "First, give a Subset, which will be removed from current." <<
endl;
    Subset *temp = createSubset();
    currentSubset->removeFromAnotherSubset(*temp);
    cout << "The result:" << endl;
    printSubset(currentSubset);

    delete temp;
}

//stop the program
void Menu::stopProgram() {
    cout << "Bye" << endl;
    delete currentSubset;
}

//interface
void Menu::menuInterface() {
    cout << "Hello, to continue please create a subset" << endl;
    currentSubset = createSubset();
    bool stop = false;
    while (!stop) {
        int choice = 0;

        cout << endl;
        cout << "Please, choose what to do with your subset" << endl;
        cout << "1. Check if the element n is in subset" << endl;
        cout << "2. Add element n to subset" << endl;
        cout << "3. Remove element n from the subset" << endl;
        cout << "4. Find the intersection of current subset with a new one"
<< endl;
        cout << "5. Find the union of current subset with a new one" << endl;
        cout << "6. Add all the elements from a new subset to the current
one" << endl;
```

```
        cout << "7. Remove all the elements of a new subset from the current
one" << endl;
        cout << "8. Print the current subset" << endl;
        cout << "9. End the program" << endl;

        cin >> choice;
        if (choice <= 0 or choice >= 10) {
            cout << "Insert a number between 1 and 9" << endl;
        } else if (choice == 1) {
            checkElement();
        } else if (choice == 2) {
            addElement();
        } else if (choice == 3) {
            popElement();
        } else if (choice == 4) {
            intersectSubsets();
        } else if (choice == 5) {
            uniteSubsets();
        } else if (choice == 6) {
            addFromAnother();
        } else if (choice == 7) {
            rmFromAnother();
        } else if (choice == 8) {
            printSubset(currentSubset);
        } else if (choice == 9) {
            stop = true;
        }
    }

    stopProgram();
}
```

main.cpp

```
#include "menu.h"

int main() {
    Menu m = Menu();
    m.menuInterface();
    return 0;
}
```

Вывод: в ходе данной лабораторной работы были реализованы классы, с которыми можно проводить различные операции.