ПРЕЗЕНТАЦИЯ ПО КУРСОВОЙ РАБОТЕ ПО “ЯЗЫКИ ПРОГРАММИРОВАНИЯ”

Студентка 1 курса
Соколенко Мария Владимировна
Группа БББО-06-19
Вариант 44

- Вариант 44.
- Разбить группу на 2 части, с поиском среди лиц определенного пола:

1) 50 процентов хороших и отличных оценок за все время обучения;
2) Все остальные студенты.

Распечатать в каждой части 2-х наиболее успевающих и наиболее неуспевающих студентов.

Реализация загрузки данных с клавиатуры:

```
void Students::addSTUD()
{
LoadG(&id, &kol);
id++;
kol++;
student* nd = new student;
nd->id = id;

cout << "Введите фамилию студента: \n";
vvodEl(nd->fam, qwe);

cout << "Имя: \n";
vvodEl(nd->name, qwe);

cout << "Отчество: \n";
vvodEl(nd->otch, qwe);

vvodPol(nd);

vvodDateR(nd);

vvodYear(nd);

vvodUniuc(head, nd, 2);
cout << "№ зачётки сгенерирован автоматически: \n";
cout << nd->Nstud << "\n";
vvodUniuc(head, nd, 1);
cout << "№ студака сгенерирован автоматически: \n";
cout << nd->Nzach << "\n"
cout << "Факультет: \n";
vvodEl(nd->fac, rty);

cout << "Кафедра: \n";
vvodEl(nd->kaf, rty);

cout << "Группа: \n";
vvodEl(nd->group, uio);

nd->sr_ball = 0;
nd->kol_obch = 0;
nd->kol_Otl_Hor = 0;
bool tmp = false;
int men_add_tmp = 0;
while (!tmp) {
cout << "\n\n № Сессии для которой ввести оценки: \n\n";
ses_num = vvod_sesNum();
vvod_predmets(nd, ses_num);
add_pred_rep_menu();
bool prov_tmp = false;
while (!prov_tmp) {
cin >> men_add_tmp;
ochiCIN();
if (men_add_tmp == 2)
{
tmp = true;
prov_tmp = true;
}
else
if (men_add_tmp == 1) {
break;
tmp = true;
}
else
eroor_menu();
}
}
nd->next = NULL;

if (head == NULL)
head = nd;
else
{
student* current = head;

while (current->next != NULL)
current = current->next;

current->next = nd;
}
tail = nd;
SaveG(&id, &kol);
saveOneStudToFile(nd);
cout << "\x1b[32m \n Успешно! \n \x1b[30m";
};
```

```
void vvodEl(char El[], int qwe) {
bool tmp = false;
while (!tmp) {
cin.getline(El, qwe);
ochiGET();

if ((!prov_prob(El) || !prov_pyst(El))) {
cout << "Не должно быть пробелов / строка не должна быть пустой!!!\n";
cout << "Повторите ввод: \n";
}
else
tmp = true;
}
}
```

Запуск программы и верное выполние задания согласно варианту :

```
student* stud = MyStud.head;
while (stud) {
int j = 0;
int ses_mas[9];
for (int i = 0; i < 9; i++)
ses_mas[i] = 0;
for (int i = 1; i <= 9; i++)
{
if (FindOch(stud->Nzach, i)) {
ses_mas[j] = i;
j++;
}
}
MyStud.schet_sr_ball(stud, ses_mas, j);
stud = stud->next;
}
bool prochent_sr_ball;
int tmp_zad_main_menu;
int tmp_zad_dop_menu;
while (1)
{
zad_main_menu44();
cin >> tmp_zad_main_menu;
ochiCIN();
if (tmp_zad_main_menu == 1) {
prochent_sr_ball = true;
while (1) {
zad_dop_menu44();
cin >> tmp_zad_dop_menu;
ochiCIN();
if (tmp_zad_dop_menu == 1)
{
MyStud.find_pech_stud_true(prochent_sr_ball);
}
else
if (tmp_zad_dop_menu == 2)
{
MyStud.find_pech_stud_true(prochent_sr_ball, 'ж');
}
else
if (tmp_zad_dop_menu == 3)
{
MyStud.find_pech_stud_true(prochent_sr_ball, 'м');
}
else
if (tmp_zad_dop_menu == 4)
{
break;
}
else
eroor_menu();
}
}
else
if (tmp_zad_main_menu == 2)
{
prochent_sr_ball = false;
while (1) {
zad_dop_menu44();
cin >> tmp_zad_dop_menu;
ochiCIN();
if (tmp_zad_dop_menu == 1)
{
MyStud.find_pech_stud_true(prochent_sr_ball);
}
else
if (tmp_zad_dop_menu == 2)
{
MyStud.find_pech_stud_true(prochent_sr_ball, 'ж');
}
else
if (tmp_zad_dop_menu == 3)
{
MyStud.find_pech_stud_true(prochent_sr_ball, 'м');
}
else
if (tmp_zad_dop_menu == 4)
{
break;
}
else
eroor_menu();
}
}
else
if (tmp_zad_main_menu == 3)
{
break;
}
else
eroor_menu();
}
```

Реализация функции записи и чтения информации в/из файла:

```
void Students::loadFILE() {
LoadG(&id, &kol);
ifstream stud;
char filename[] = "students.txt";
stud.open(filename);
if (!stud.is_open())
cout << "\n\nФайл студетов не может быть открыт!\n\n";
else
if (!FileIsEmpty(filename))
cout << "\x1b[31m \n База данных пуста! Заполните базу данных!!!\n \x1b[30m";
else
{
for (int i = 0; i < kol; i++)
{
student* nd = new student;
stud >> nd->id;
stud >> nd->fam;
stud >> nd->name;
stud >> nd->otch;
stud >> nd->pol;
stud >> nd->day;
stud >> nd->mount;
stud >> nd->year;
stud >> nd->yPOST;
stud >> nd->fac;
stud >> nd->kaf;
stud >> nd->group;
stud >> nd->Nstud;
stud >> nd->sr_ball;
stud >> nd->kol_Otl_Hor;
stud >> nd->kol_obch;
stud >> nd->Nzach;
}
nd->next = NULL;
tail = nd;
if (head == NULL)
head = nd;
else
{
student* current = head;


while (current->next != NULL)
current = current->next;


current->next = nd;
}
}
cout << "\x1b[32m \n База данных успешно загружена!!! \n \x1b[30m";
cout << "Количество студентов в базе = " << kol << "\n\n";
}
stud.close();
}
```

Использование динамической памяти:

```
void Students::addSTUD()
{
    LoadG(&id, &kol);
    id++;
    kol++;
    student* nd = new student;
    nd->id = id;
```

```
while (head)
{
    tail = head->next;
    delete head;
    head = tail;

}
```

```
predmet* pr = new predmet;
vvodEl(vv, pred);
```

```
while (head)
{
    tail = head->next;
    delete head;
    head = tail;

}
```

Функция добавления или удаления записей в файле:

```
void Students::saveOneStudToFile(student* st) {
ofstream one_stud;
one_stud.open("students.txt", ios::app);
one_stud << st->id << " ";
one_stud << st->fam << " ";
one_stud << st->name << " ";
one_stud << st->otch << " ";
one_stud << st->pol << " ";
one_stud << st->day << " ";
one_stud << st->mount << " ";
one_stud << st->year << " ";
one_stud << st->yPOST << " ";
one_stud << st->fac << " ";
one_stud << st->kaf << " ";
one_stud << st->group << " ";
one_stud << st->Nstud << " ";
one_stud << st->sr_ball << " ";
one_stud << st->kol_Otl_Hor << " ";
one_stud << st->kol_obch << " ";
one_stud << st->Nzach << "\n";
one_stud.close();
}
```

```
void Students::delete_student(student* del) {
student* ptr;
if (del != NULL) {
if (del == head)
{
head = head->next;
delete(del);
del = head;
}
else
{
ptr = head;
while (ptr->next != del)
ptr = ptr->next;
ptr->next = del->next;
delete(del);
del = ptr;
}
}
LoadG(&id, &kol);
id = id;
kol--;
SaveG(&id, &kol);
}
```

После данной функции делается перезапись файла

Конструкторы и деструкторы:

```
class Students : private Global_Info, Predmets
{
    friend void main_menu();
    friend void print_och_all(student*);
protected:
    student* head, * tail;
    int kol_fin;
    student* find;
public:
    Students()
    {
        cout << "\n\nСрабатывает конструктор Students \n\n";
        head = NULL;
        tail = NULL;
        find = NULL;
        kol_fin = 0;

    }
```

```
~Students()
{
    while (head)
    {
        tail = head->next;
        delete head;
        head = tail;

    }
}
```

Друзья классов:

```
class Students : private Global_Info, Predmets
{
    friend void main_menu();
    friend void print_och_all(student*);
```

```
class StringsWork
{
    friend void Predmets::save_och(int, char*, student*);
    friend void Predmets::save_predmets(char*, int);
    friend void Predmets::delet_och_stud(char*);
    friend void Predmets::load_predmets(char*, int);
    friend void Predmets::load_och(char*, int);
    friend bool FindKaf(char[], int);
    friend bool FindOch(char[], int);
```

Наследование:

```
class Students : private Global_Info, Predmets
```

Перегрузка операций:

```
class StringsWork
{
    friend void Predmets::save_och(int, char*, student*);
    friend void Predmets::save_predmets(char*, int);
    friend void Predmets::delet_och_stud(char*);
    friend void Predmets::load_predmets(char*, int);
    friend void Predmets::load_och(char*, int);
    friend bool FindKaf(char[], int);
    friend bool FindOch(char[], int);
private:
    char str[256] = {};
public:
    StringsWork()
    {
        for (int i = 0; i < 256; i++)  str[i] = '\0';
    }

    void operator +(char*);
};
```

```
void StringsWork::operator +(char* s)
{
    int len = strlen(str), lens = strlen(s);
    for (int i = len, j = 0; j <= lens; i++, j++)
        str[i] = s[j];

}
```

```
StringsWork file_name;
file_name + pts;
file_name + kafed;
file_name + dop;
file_name + rasz;
```

Шифрование файла:

```
hSourceFile = CreateFile(TEXT("global/key.txt"), FILE_WRITE_DATA, FILE_SHARE_READ, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if (INVALID_HANDLE_VALUE != hSourceFile)
{
cout << "\x1b[32m \n The source plaintext file, %s, is open. \n \x1b[30m";
}
else
{
cout << "\x1b[31m \n Error opening source plaintext file! \n \x1b[30m";
}
if (CryptAcquireContext(&hCryptProv, NULL, MS_ENHANCED_PROV, PROV_RSA_FULL, 0))
{
cout << "\x1b[32m \nA cryptographic provider has been acquired. \n \x1b[30m";
}
else
{
cout << "\x1b[31m \nError during CryptAcquireContext! \n \x1b[30m";
}
if (CryptGenKey(hCryptProv, ENCRYPT_ALGORITHM, KEYLENGTH | CRYPT_EXPORTABLE, &hKey))
{
cout << "\x1b[32m \nA session key has been created.\n \x1b[30m";
}
else
{
cout << "\x1b[31m \nError during CryptGenKey. \n \x1b[30m";
}
if (CryptGetUserKey(hCryptProv, AT_KEYEXCHANGE, &hXchgKey))
{
cout << "\x1b[32m \nThe user public key has been retrieved.\n \x1b[30m";
}
else
{
if (NTE_NO_KEY == GetLastError())
{
// No exchange key exists. Try to create one.
if (!CryptGenKey(hCryptProv, AT_KEYEXCHANGE, CRYPT_EXPORTABLE, &hXchgKey))
{
cout << "\x1b[31m \nCould not create a user public key. \n \x1b[30m";
}
}
else
{
cout << "\x1b[31m \nUser public key is not available and may not exist. \n \x1b[30m";
}
}
```

```
if (CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, NULL, &dwKeyBlobLen))
{
cout << "\x1b[32m \nThe key BLOB is %d bytes long. \n \x1b[30m";
}
else
{
cout << "\x1b[31m \nError computing BLOB length! \n \x1b[30m";
}

if (pbKeyBlob = (BYTE*)malloc(dwKeyBlobLen))
{
cout << "\x1b[32m \nMemory is allocated for the key BLOB. \n \x1b[30m";
}
else
{
cout << "\x1b[31m \nOut of memory. \n \x1b[30m";
}
if (CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, pbKeyBlob, &dwKeyBlobLen))
{
cout << "\x1b[32m \nThe key has been exported. \n \x1b[30m";
}
else
{
cout << "\x1b[31m \nError during CryptExportKey! \n \x1b[30m";
}
if (hXchgKey)
{
if (!(CryptDestroyKey(hXchgKey)))
{
cout << "\x1b[31m \nError during CryptDestroyKey. \n \x1b[30m";
}

hXchgKey = 0;
}
if (!WriteFile(hSourceFile, &dwKeyBlobLen, sizeof(DWORD), &dwCount, NULL))
{
cout << "\x1b[31m \nError writing header. \n \x1b[30m";
}
else
{
cout << "\x1b[32m \nA file header has been written. \n \x1b[30m";
}
```

```
if (!WriteFile(hSourceFile, pbKeyBlob, dwKeyBlobLen, &dwCount, NULL))
{
cout << "\x1b[31m \nError writing header. \n \x1b[30m";
}
else
{
cout << "\x1b[32m \nThe key BLOB has been written to the file. \n \x1b[30m";
}

// Free memory.
free(pbKeyBlob);
bool fEOF = false;
```

Непосредсвенно выполнение самого шифрования:

```
count = strlen(current->name);
if (!CryptEncrypt(hKey, NULL, fEOF, 0, (BYTE*)current->name, &count,
strlen(current->name)))
{
cout << "\x1b[31m \nError during CryptEncrypt. \n \x1b[30m";
}
stud << current->name << " ";
```

Дешифрование файла:

```
hSourceFile = CreateFile(
TEXT("global/key.txt"),
FILE_READ_DATA,
FILE_SHARE_READ,
NULL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
NULL);
if (INVALID_HANDLE_VALUE != hSourceFile)
{
cout << "\x1b[32m \n The source encrypted file, %s, is open. \n \x1b[30m";
}
else
{
cout << "\x1b[31m \n Error opening source plaintext file! \n \x1b[30m";
}
if (CryptAcquireContext(
&hCryptProv,
NULL,
MS_ENHANCED_PROV,
PROV_RSA_FULL,
0))
{
cout << "\x1b[32m \n A cryptographic provider has been acquired \n \x1b[30m";
}
else
{
cout << "\x1b[31m \n Error during CryptAcquireContext! \n \x1b[30m";
}
// Decrypt the file with the saved session key.

DWORD dwKeyBlobLen;
PBYTE pbKeyBlob = NULL;

// Read the key BLOB length from the source file.
if (!ReadFile(
hSourceFile,
&dwKeyBlobLen,
sizeof(DWORD),
0,
NULL))
{
cout << "\x1b[31m \n Error reading key BLOB length! \n \x1b[30m";
}
if (!(pbKeyBlob = (PBYTE)malloc(dwKeyBlobLen)))
{
cout << "\x1b[31m \n Memory allocation error. \n \x1b[30m";
}
if (!ReadFile(
hSourceFile,
pbKeyBlob,
dwKeyBlobLen,
0,
NULL))
{
cout << "\x1b[31m \n Error reading key BLOB length! \n \x1b[30m";
}
if (!CryptImportKey(
hCryptProv,
pbKeyBlob,
dwKeyBlobLen,
0,
0,
&hKey))
{
cout << "\x1b[31m \n Error during CryptImportKey! \n \x1b[30m";
}
if (pbKeyBlob)
{
free(pbKeyBlob);
}
if (!CryptDecrypt(
hKey,
0,
fEOF,
0,
(BYTE*)nd->fam,
&count))
{
cout << "\x1b[31m \n Error during CryptDecrypt! \n \x1b[30m";
}
```