Міністерство освіти і науки України
Національний університет «Львівська політехніка»
кафедра інформаційних систем та мереж

Григорович Віктор

Алгоритмізація та програмування
Переліки, структури та об’єднання

Навчальний посібник

2020

Григорович Віктор Геннадійович

Алгоритмізація та програмування. Переліки, структури та об’єднання. Навчальний посібник.

Дисципліна «Алгоритмізація та програмування» вивчається на першому курсі, – з цієї дисципліни розпочинається цикл предметів, що стосуються програмування та розробки програмного забезпечення.

У посібнику містяться теоретичні відомості, приклади, методичні вказівки з їх розв’язування, варіанти лабораторних завдань та питання і завдання з контролю знань з теми «Переліки, структури та об’єднання».

Розглядаються наступні роботи лабораторного практикуму:

Лабораторна робота № 9.1.

Послідовний пошук в масиві структур

Лабораторна робота № 9.2.

Впорядкування та бінарний пошук в масиві структур

Лабораторна робота № 9.3.

Опрацювання масиву структур

В посібнику наведено рекомендований набір лабораторних робіт та приклади їх виконання.

Відповідальний за випуск – Григорович В.Г.

Стислий зміст

Зміст

Вступ

Дисципліна «Алгоритмізація та програмування» вивчається на першому курсі, – з цієї дисципліни розпочинається цикл предметів, що стосуються програмування та розробки програмного забезпечення.

В посібнику містяться методичні вказівки та варіанти лабораторних завдань з основ програмування мовою C/C++ в середовищі Microsoft Visual Studio C++.

В посібнику наведено рекомендований набір лабораторних робіт з теми «Переліки, структури та об’єднання», приклади їх виконання та прийнятий стиль оформлення C/C++ програм.

Тема 9. Переліки, структури та об’єднання

Стисло та головне про переліки, структури та об’єднання

Переліки

Визначення перелічуваного типу

Перелічуваний тип – це тип, що визначається користувачем та складається із набору цілих констант, які називаються *нумераторами* (або *елементами переліку*).

Загальний вигляд команди визначення перелічуваного типу та змінних – переліків (квадратні дужки використовуються для позначення необов’язкового елементу синтаксичної конструкції):

```
enum [ім’я-перелічуваного-типу]
{
    список-елементів-переліку
} [список-змінних-переліків];
```

де

ім’я-перелічуваного-типу – ім’я перелічуваного типу, необов’язкове. Якщо ім’я перелічуваного типу не вказане, то визначається анонімний перелік;

список-елементів-переліку – список ідентифікаторів (нумераторів, елементів переліку), відокремлених комами;

список-змінних-переліків – оголошення змінних-переліків розглядається в наступному параграфі.

Команда визначення перелічуваного типу – це команда визначення даних, яка містить блок визначення даних (елементів переліку). Кожний блок визначення даних (на відміну від блоку опису команд) має закінчуватися символом «;» (крапка з комою).

Кожний ідентифікатор (елемент переліку) в своїй області видимості має бути унікальним, проте їх значення можуть повторюватися. Областю видимості нумераторів є охоплюючий блок (область видимості самого переліку), тобто елементи переліку видимі в тій області, в якій визначено перелік. Наступний фрагмент – не відкомпілюється:

```
enum A {a, b, c};
enum B {a, x, y};
```

– `error C2365: 'a' : redefinition; previous definition was 'enumerator'`, причина помилки – нумератор `a` повторно визначений і в переліку `A`, і в переліку `B`.

Кожному елементу присвоюється ціле значення, яке відповідає певному місцю розташування цього елемента в переліку. За умовчанням, значення присвоюються за порядком, починаючи з нуля: першому елементу присвоюється значення `0`, наступному – `1`, і т.д.:

```
enum Color { RED, YELLOW, GREEN };
```

– визначили тип `Color`, який складається із трьох елементів. Елемент `RED` має значення `0`, `YELLOW` – значення `1`, `GREEN` – значення `2`.

Оголошення переліків – змінних перелічуваних типів

Змінні-переліки (змінні перелічуваних типів) можна оголосити в команді визначення перелічуваного типу:

```
enum Color { RED, YELLOW, GREEN } x, y;
```

– визначається перелічуваний тип `Color` та переліки (змінні) `x`, `y`, які можуть набувати значень `RED`, `YELLOW` та `GREEN`.

В мові C ім'я перелічуваного типу є тегом (міткою), яке потрібно використовувати лише разом з ключовим словом `enum`. Оголошення змінних-переліків окремо від визначення перелічуваного типу в мові C виглядає так:

```
enum Color { RED, YELLOW, GREEN };
enum Color x, y;
```

В мові C++ перелічуваний тип – це справжній тип і його ім'я є повноправним ідентифікатором (не тегом), тобто ім'я перелічуваного типу можна використовувати при оголошенні змінних-переліків без ключового слова `enum`:

```
enum Color { RED, YELLOW, GREEN };
Color x, y;
```

Можна оголошувати переліки неіменованих (анонімних) типів: ім'я типу не вказується, але можна оголошувати змінні:

```
enum { RED, YELLOW, GREEN } x, y;
```

– змінні `x` та `y` – переліки неіменованого перелічуваного типу, які можуть набувати значень `RED`, `YELLOW` та `GREEN`.

Структури

Визначення структурного типу

Структурний тип – це тип, що визначається користувачем та складається із набору елементів, які називаються *полями*.

Загальний вигляд команди визначення структурного типу та змінних – структур (квадратні дужки використовуються для позначення необов'язкового елементу синтаксичної конструкції):

```
struct [ім'я-структурного-типу]
{
    список-полів
} [список-змінних-структур];
```

де

ім'я-структурного-типу – ім'я структурного типу, необов'язкове. Якщо ім'я структурного типу не вказане, то визначається анонімна структура;

список-полів – список оголошень полів; оголошення поля аналогічне до оголошення змінних; *список-полів* має наступний загальний вигляд (квадратні дужки використовуються для позначення необов'язкового елементу синтаксичної конструкції):

ім'я-типу ім'я-поля [, ..., *ім'я-поля*];

[...

ім'я-типу ім'я-поля [, ..., *ім'я-поля*];]

В структурі має бути по крайній мірі одне поле.

Імена полів в структурі мають бути унікальними, але ім'я поля цієї структури можна використовувати як ідентифікатор іншого ресурсу – змінної, типу, функції, поля іншої структури чи об'єднання. В різних структурах можна використовувати однакові імена полів.

список-змінних-структур – оголошення змінних-структур розглядається в наступному параграфі.

Команда визначення структурного типу – це команда визначення даних, яка містить блок визначення даних (елементів структури). Кожний блок визначення даних (на відміну від блоку опису команд) має закінчуватися символом «;» (крапка з комою).

Приклад:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
};
```

Оголошення та розподіл пам'яті для структур – змінних структурних типів

Оголошення структур

Як і для переліків, змінні-структури (змінні структурних типів) можна оголосити в команді визначення структурного типу:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
} s1, s2;
```

– визначається структурний тип `Student` та структури (змінні) `s1` та `s2`.

Стилістично кращим вважається оголошувати змінні окремо від визначення типу.

В мові С ім'я структурного типу є тегом (міткою), яке потрібно використовувати лише разом з ключовим словом `struct`. Оголошення змінних-структур окремо від визначення структурного типу в мові С виглядає так:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
};

struct Student s1, s2;
```

В мові С++ структурний тип – це справжній тип і його ім'я є повноправним ідентифікатором (не тегом), тобто ім'я структурного типу можна використовувати при оголошенні змінних-структур без ключового слова `struct`:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
};

Student s1, s2;
```

Можна оголошувати структури неіменованих (анонімних) типів: ім'я типу не вказується, але можна оголошувати змінні:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
} s1, s2;
```

– змінні `s1` та `s2` – структури неіменованого структурного типу.

Як і для інших типів, для структурного можна оголошувати масиви структур та вказівники на структури:

```
Student s3[25], *ps;
```

– оголосили масив `s3` із `25` елементів типу `Student` та вказівник `ps` на структуру типу `Student`.

Розподіл пам'яті для структур

В пам'яті для структури виділяється неперервна область, розмір якої дорівнює сумі розмірів всіх полів – але є один нюанс, який називається вирівнюванням. Розмір області пам'яті, виділеної для структури, може бути більшим за суму розмірів полів цієї структури.

Поля, вказані при визначенні структури першими, розміщуються в комірках з молодшими адресами.

Розглянемо, що таке вирівнювання і як воно працює:

Для структури

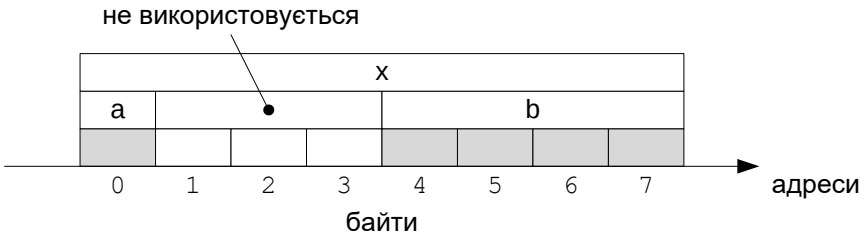
```
struct A
{
    char a;
    int b;
} x;
```

її розмір `sizeof(A)` (чи `sizeof(x)`) не завжди буде дорівнювати `5` – все залежить від налаштування компілятора та директив в тексті програми.

Правило: поля в пам'яті вирівнюються по границі, кратній своєму розміру.

Тобто, 1-байтові поля не вирівнюються, 2-байтові – вирівнюються на позиції парних адрес, 4-байтові – на позиції адрес, кратних чотирьом і т.д.

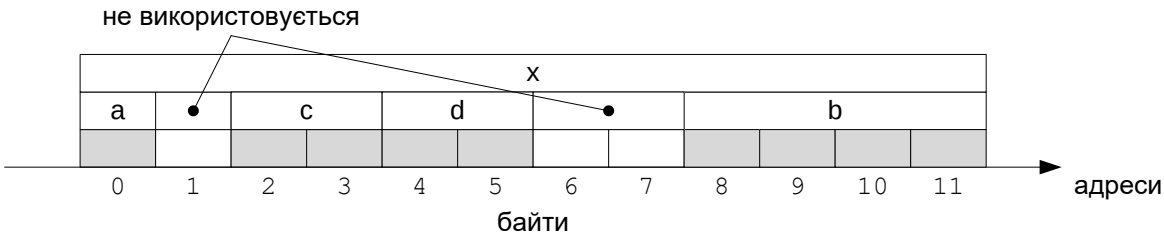
У більшості випадків за умовчанням вирівнювання розміру структури в пам'яті становить 4 байти. Таким чином, `sizeof(A) == 8`:



Добавимо ще поля:

```
struct A
{
    char  a;
    short c;
    short d;
    int   b;
} x;
```

Тепер поля розміщуються в пам’яті наступним чином:



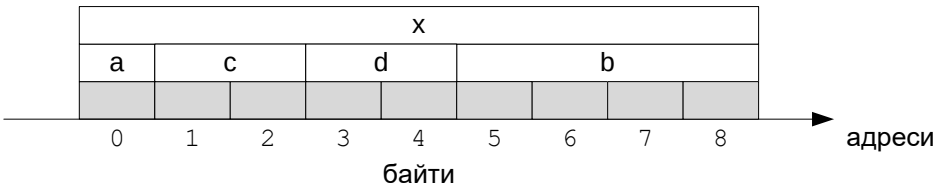
Є можливість керувати вирівнюванням безпосередньо в програмі:

```
#pragma pack(push, 1)

struct A
{
    char  a;
    short c;
    short d;
    int   b;
} x;

#pragma pack(pop)
```

– встановили вирівнювання в 1 байт, визначили структуру та вернули попередні налаштування.

Повертати попередні налаштування – категорично рекомендується!

Тепер розподіл пам’яті виглядає так:



Ініціалізація структури

Ініціалізація структури виконується аналогічно до ініціалізації масиву:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
};

Student s = { "Іваненко", 1997, I, KN }; // ініціалізація
```

– створюємо структуру – змінну типу `Student` та присвоюємо значення всім її полям. Порядок дуже важливий при ініціалізації структури, бо компілятор встановлює позиційну відповідність між полями при визначенні структури та константами в ініціалізаторі – порядок значень в ініціалізаторі має збігатися з порядком полів при визначенні структури.

Якщо якесь поле залишилося не заповненим, то воно автоматично заповнюється значенням `0` – для цілих типів, `NULL` – для вказівників, `'\0'` (нуль-символ) – для літерних рядків.

Доступ до елементів структури

До елементів структури можна звертатися двома способами:

1) за допомогою імені структури;
2) за допомогою вказівника на структуру.

Доступ за іменем структури

Доступ до елементів структури за допомогою імені цієї структури здійснюється через операцію `.` (крапка), загальний вигляд – наступний:

```
структура.поле
```

Приклад:

```
Student s;

s.prizv = "Іваненко";
s.rikNar = 1997;
s.kurs = I;
s.spec = KN;
```

Доступ за вказівником на структуру

Є два способи доступу до елементів структури за допомогою вказівника на цю структуру.

Перший спосіб – традиційний, що використовує операцію роз-іменування вказівника `*` (зірочка), загальний вигляд – наступний:

```
(*вказівник-на-структуру).поле
```

Приклад:

```
Student *p = new Student;

(*p).prizv = "Іваненко";
(*p).rikNar = 1997;
(*p).kurs = I;
(*p).spec = KN;
```

Другий спосіб – більш зручний – використовує операцію -> (стрілка), яка записується двома послідовними символами: - (мінус) та > (більше). Загальний вигляд:

```
вказівник-на-структуру -> поле
```

Приклад:

```
Student *p = new Student;

p->prizv = "Іваненко";
p->rikNar = 1997;
p->kurs = I;
p->spec = KN;
```

Об’єднання

Визначення об’єднуваного типу

Як і структурний, об’єднуваний тип – це тип, що визначається користувачем та складається із набору елементів, які називаються *полями*.

Загальний вигляд команди визначення об’єднуваного типу та змінних – об’єднань (квадратні дужки використовуються для позначення необов’язкового елементу синтаксичної конструкції):

```
union [ім’я-об’єднуваного-типу]
{
    список-полів
} [список-змінних-об’єднань];
```

де

ім’я-об’єднуваного-типу – ім’я об’єднуваного типу, необов’язкове. Якщо ім’я об’єднуваного типу не вказане, то визначається анонімне об’єднання;

список-полів – список оголошень полів; оголошення поля аналогічне до оголошення змінних; *список-полів* має наступний загальний вигляд (квадратні дужки використовуються для позначення необов’язкового елементу синтаксичної конструкції):

```
ім’я-типу ім’я-поля [ , ..., ім’я-поля ];
[...
ім’я-типу ім’я-поля [ , ..., ім’я-поля ];]
```

В об’єднанні має бути по крайній мірі одне поле.

Імена полів в об’єднанні мають бути унікальними, але ім’я поля цього об’єднання можна використовувати як ідентифікатор іншого ресурсу – змінної, типу, функції, поля іншого об’єднання чи структури. В різних об’єднаннях можна використовувати однакові імена полів.

список-змінних-об’єднань – оголошення змінних-об’єднань розглядається в далі.

Команда визначення об’єднуваного типу – це команда визначення даних, яка містить блок визначення даних (елементів об’єднання). Кожний блок визначення даних (на відміну від блоку опису команд) має закінчуватися символом «;» (крапка з комою).

Приклад:

```
union Oplata
{
    int    stavka;
    double taryf;
};
```

Оголошення та розподіл пам’яті для об’єднань – змінних об’єднуваних типів

Оголошення об’єднань

Як і для переліків та структур, змінні-об’єднання (змінні об’єднуваних типів) можна оголосити в команді визначення об’єднуваного типу:

```
union Oplata
{
    int    stavka;
    double taryf;
} o1, o2;
```

– визначається об’єднуваний тип `Oplata` та об’єднання (змінні) `o1` та `o2`.

Стилістично кращим вважається оголошувати змінні окремо від визначення типу.

В мові C ім’я об’єднуваного типу є тегом (міткою), яке потрібно використовувати лише разом з ключовим словом `union`. Оголошення змінних-об’єднань окремо від визначення об’єднуваного типу в мові C виглядає так:

```
union Oplata
{
    int    stavka;
    double taryf;
};

union Oplata o1, o2;
```

В мові C++ об’єднуваний тип – це справжній тип і його ім’я є повноправним ідентифікатором (не тегом), тобто ім’я об’єднуваного типу можна використовувати при оголошенні змінних-об’єднань без ключового слова `union`:

```
union Oplata
{
    int    stavka;
    double taryf;
};

Oplata o1, o2;
```

Можна оголошувати об’єднання неіменованих (анонімних) типів: ім’я типу не вказується, але можна оголошувати змінні:

```
union
{
    int    stavka;
    double taryf;
} o1, o2;
```

– змінні `o1` та `o2` – об’єднання неіменованого об’єднуваного типу.

Як і для інших типів, для об’єднуваного можна оголошувати масиви об’єднань та вказівники на об’єднання:

```
Oplata o3[25], *po;
```

– оголосили масив `o3` із `25` елементів типу `Oplata` та вказівник `po` на об’єднання типу `Oplata`.

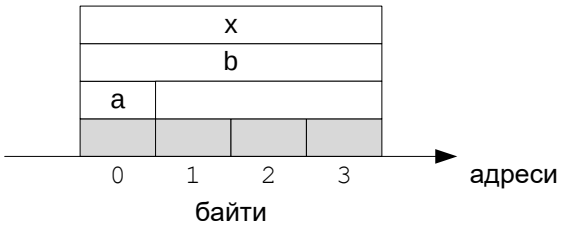
Розподіл пам’яті для об’єднань

В пам’яті для об’єднання виділяється неперервна область, розмір якої дорівнює розміру найбільшого поля.

Для об’єднання

```
union A
{
    char a;
    int  b;
} x;
```

його розмір `sizeof(A)` (чи `sizeof(x)`) буде дорівнювати `4`:



– поле `b` об’єднання `x` типу `A` дає доступ до всіх його `4`-х байт і трактує вміст цієї області як ціле число; поле `a` об’єднання `x` дає доступ лише до самого першого байту (інші байти об’єднання цим полем ігноруються) і трактує вміст цього байту як код символу.

Ініціалізація об’єднання

Ініціалізація об’єднання виконується аналогічно до ініціалізації масиву чи структури, проте в команді ініціалізації можна надати значення лише першому полю об’єднання:

```
union A
{
    char a;
    int  b;
};

A x = { 'a' };

cout << x.a << endl;
cout << x.b << endl;
```

Вивід:

```
a
97
```

– ініціалізуємо об’єднання символом `'a'`.

За допомогою поля `b` ми трактуємо символ як ціле число (код цього символу), тому друга команда `cout` виведе значення `97`.

Наступний спосіб ініціалізації – хибний (деякі системи видадуть попередження, а деякі – повідомлення про помилку):

```
union A
{
    char a;
    int  b;
};

A x = { 10000 };
```

– значення `10000` буде «обрізане», щоб помістити в один байт.

Доступ до елементів об’єднання

Доступ до елементів об’єднання аналогічний доступу до елементів структури.

Теоретичні відомості

Переліки

Вступ

Припустимо, потрібно написати програму, яка буде керувати світлофором. Виникає питання – як подати значення кольорів? Можливо, просто цілими числами: 0 – буде позначати червоний колір, 1 – жовтий, а 2 – зелений. Такий спосіб хороший тим, що значення цілих чисел легко опрацьовуються, але використовувати так звані «магічні числа» – поганий стиль програмування («магічними» такі числа називаються тому, що ніхто не може пояснити, чому 0 означає саме червоний колір) – коли в програмі багато «магічних чисел», зрозуміти її стає дуже важко.

Інший спосіб подати кольори значеннями літерних констант `"червоний"`, `"жовтий"` та `"зелений"`. Літерні константи – змістовні, проте їх важко опрацьовувати.

Ще один спосіб подання кольорів – визначити цілі константи:

```
const int RED    = 0;
const int YELLOW = 1;
const int GREEN  = 2;
```

Імена констант – значущі, цілі числа легко опрацьовувати. Проте цей спосіб не може запобігти помилки, пов’язаної з тим, що буде введено інше ціле число як значення кольору – а це приведе до збою в роботі світлофору.

Виходом може служити використання перелічуваних типів та переліків.

Перелічуваний тип (скор. **перелік**, англ. *enumeration, enumerated type*) – тип даних, множиною значень якого є обмежений список ідентифікаторів – констант цього типу.

Перелічуваний тип визначається як набір ідентифікаторів, які виконують ту саму роль, що і звичайні іменовані константи, але ідентифікатори – константи перелічуваного типу – будуть пов’язані з цим типом, що дає можливість контролю за «правильним» використанням таких констант.

Використання переліків дозволяє зробити текст програми більш зрозумілим, оскільки дає можливість замінити «магічні числа», які кодують певні значення, значущими іменами.

При компіляції значення переліків представляються у вигляді цілих чисел.

Визначення перелічуваного типу

Перелічуваний тип – це тип, що визначається користувачем та складається із набору цілих констант, які називаються *нумераторами* (або *елементами переліку*).

Загальний вигляд команди визначення перелічуваного типу та змінних – переліків (квадратні дужки використовуються для позначення необов'язкового елементу синтаксичної конструкції):

```
enum [ім'я-перелічуваного-типу]
{
    список-елементів-переліку
} [список-змінних-переліків];
```

де

ім'я-перелічуваного-типу – ім'я перелічуваного типу, необов'язкове. Якщо ім'я перелічуваного типу не вказане, то визначається анонімний перелік;

список-елементів-переліку – список ідентифікаторів (нумераторів, елементів переліку), відокремлених комами;

список-змінних-переліків – оголошення змінних-переліків розглядається в наступному параграфі.

Команда визначення перелічуваного типу – це команда визначення даних, яка містить блок визначення даних (елементів переліку). Кожний блок визначення даних (на відміну від блоку опису команд) має закінчуватися символом «;» (крапка з комою).

Кожний ідентифікатор (елемент переліку) в своїй області видимості має бути унікальним, проте їх значення можуть повторюватися. Областю видимості нумераторів є блок, який їх охоплює (область видимості самого переліку), тобто елементи переліку видимі в тій області, в якій визначено перелік. Наступний фрагмент – не відкомпілюється:

```
enum A {a, b, c};
enum B {a, x, y};
```

– `error C2365: 'a' : redefinition; previous definition was 'enumerator'`, причина помилки – нумератор `a` повторно визначений і в переліку `A`, і в переліку `B`.

Кожний елемент переліку в загальному має наступний вигляд (квадратні дужки використовуються для позначення необов'язкового елементу синтаксичної конструкції):

ім'я-елементу [= *ціла-константа*]

Кожному елементу присвоюється ціле значення, яке відповідає певному місцю розташування цього елемента в переліку. За умовчанням, значення присвоюються в порядку зростання, починаючи з нуля: першому елементу присвоюється значення `0`, наступному – `1`, і т.д.:

```
enum Color { RED, YELLOW, GREEN };
```

– визначили тип `Color`, який складається із трьох елементів. Елемент `RED` має значення `0`, `YELLOW` – значення `1`, `GREEN` – значення `2`.

Значення для елементу переліку можна надати явно:

```
enum Color { RED = 1, YELLOW, GREEN };
```

Елемент `RED` тепер має значення `1`. Якщо наступним елементам переліку не присвоюються явні значення, то вони отримують наступні за порядком значення; – значення попереднього елемента, збільшене на одиницю, тобто елемент `YELLOW` отримує значення `2`, а елемент `GREEN` – значення `3`.

Кожний елемент переліку опрацьовується як константа, він має мати унікальне ім'я в тій області, в якій визначено цей перелік.

Значення, які визначаються елементами переліку, можуть бути неунікальними, наприклад:

```
enum Color { RED = 2, YELLOW = 1, GREEN };
```

– значення елементів `RED`, `YELLOW` та `GREEN` дорівнюють `2`, `1` та `2` відповідно. Значення `2` використовується кілька разів, це допускається.

Оголошення переліків – змінних перелічуваних типів

Змінні-переліки (змінні перелічуваних типів) можна оголосити в команді визначення перелічуваного типу:

```
enum Color { RED, YELLOW, GREEN } x, y;
```

– визначається перелічуваний тип `Color` та переліки (змінні) `x`, `y`, які можуть набувати значень `RED`, `YELLOW` та `GREEN`.

В мові C ім'я перелічуваного типу є тегом (міткою), яке потрібно використовувати лише разом з ключовим словом `enum`. Оголошення змінних-переліків окремо від визначення перелічуваного типу в мові C виглядає так:

```
enum Color { RED, YELLOW, GREEN };
enum Color x, y;
```

В мові C++ перелічуваний тип – це справжній тип і його ім'я є повноправним ідентифікатором (не тегом), тобто ім'я перелічуваного типу можна використовувати при оголошенні змінних-переліків без ключового слова `enum`:

```
enum Color { RED, YELLOW, GREEN };
Color x, y;
```

Можна оголошувати переліки неіменованих (анонімних) типів: ім'я типу не вказується, але можна оголошувати змінні:

```
enum { RED, YELLOW, GREEN } x, y;
```

– змінні `x` та `y` – переліки неіменованого перелічуваного типу, які можуть набувати значень `RED`, `YELLOW` та `GREEN`.

Якщо при опрацюванні функцією параметра-переліку деяке значення не опрацьовується (наприклад, один із елементів переліку забули опрацювати в команді `switch`) – як в наступному фрагменті програми:

```
enum Color { RED, YELLOW, GREEN };

void f(Color c)
{
    switch (c)
    {
        case RED:
            cout << RED << endl;
            break;
        case YELLOW:
            cout << YELLOW << endl;
            break;
    }
}
```

– тоді компілятор може вивести попередження про пропущене значення.

Елементи переліку неявно перетворюються до типу `int`:

```
int red = RED;
```

Основні проблеми при використанні переліків

Найчастіше зустрічаються наступні проблеми:

1) Відображення значення елемента переліку в літерний рядок, який містить ім'я цього елемента, тобто `RED` → `"RED"`.
2) Ітерація по елементах переліку та контроль виходу за межі: скільки б ми не додавали елементів до переліку, завжди є константа, яка на одиницю перевищує значення останнього елемента.

Кожну із цих задач має розв'язувати програміст самостійно – компілятор за нас цю роботу не зробить!

Відображення елементів переліку у літерні рядки

Можливий спосіб розв'язку:

```
#include <iostream>

using namespace std;

enum Color { RED, YELLOW, GREEN };

char* strColors[] = { "RED", "YELLOW", "GREEN" };

char* convert(Color c)
{
    return strColors[c];
}
```

```
int main()
{
    Color c = RED;
    cout << convert( c ) << endl;

    return 0;
}
```

Недоліком вказаного способу є те, що він – статичний (прив’язаний до конкретного коду):

- вручну необхідно перевіряти, щоб кількість елементів переліку збігалася із кількістю елементів в масиві літерних рядків;
- елементи переліку мають нумеруватися за порядком, починаючи з нуля, без пропусків номерів.

Перевагою є те, що літерні рядки можуть не збігатися з назвами елементів переліку:

```
char* strs[] = {
                    "Red as an egypt rose",
                    "Yellow submarine",
                    "Green peace"
                };
```

Структури

Вступ

Коли потрібно поєднати в єдине ціле сукупність однотипних даних – використовуються масиви. Але часто виникає задача описати певну сутність, яка має характеристики різних типів: наприклад сутність Студент характеризується такими даними: прізвище (літерний рядок), рік народження (коротке без-знакове ціле число), курс (перелік), спеціальність (перелік) тощо. Очевидно, для опису такої сутності тип даних «масив» – не підходить. Використовувати для такого опису незв'язані між собою змінні – недоцільно, оскільки втрачається цілісність подання інформації та унеможливлюється контроль за її правильним опрацюванням.

Вирішити вказану проблему можна за допомогою **структурних типів** та **структур**, які дозволяють поєднати в єдине ціле сукупність різнотипних даних.

Структури – це об'єднані дані різних типів, які характеризують певну сутність та за цим змістом пов'язані між собою.

Визначення структурного типу

Структурний тип – це тип, що визначається користувачем та складається із набору елементів, які називаються *полями*.

Загальний вигляд команди визначення структурного типу та змінних – структур (квадратні дужки використовуються для позначення необов'язкового елементу синтаксичної конструкції):

```
struct [ім'я-структурного-типу]
{
    список-полів
} [список-змінних-структур];
```

де

ім'я-структурного-типу – ім'я структурного типу, необов'язкове. Якщо ім'я структурного типу не вказане, то визначається анонімна структура;

список-полів – список оголошень полів; оголошення поля аналогічне до оголошення змінних; *список-полів* має наступний загальний вигляд (квадратні дужки використовуються для позначення необов'язкового елементу синтаксичної конструкції):

```
ім'я-типу ім'я-поля [ , ..., ім'я-поля ];
[...
ім'я-типу ім'я-поля [ , ..., ім'я-поля ];]
```

В структурі має бути по крайній мірі одне поле.

Імена полів в структурі мають бути унікальними, але ім'я поля цієї структури можна використовувати як ідентифікатор іншого ресурсу – змінної, типу, функції, поля іншої структури чи об'єднання. В різних структурах можна використовувати однакові імена полів.

список-змінних-структур – оголошення змінних-структур розглядається в наступному параграфі.

Команда визначення структурного типу – це команда визначення даних, яка містить блок визначення даних (елементів структури). Кожний блок визначення даних (на відміну від блоку опису команд) має закінчуватися символом «;» (крапка з комою).

Приклад:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
};
```

Оголошення та розподіл пам'яті для структур – змінних структурних типів

Оголошення структур

Як і для переліків, змінні-структури (змінні структурних типів) можна оголосити в команді визначення структурного типу:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
} s1, s2;
```

– визначається структурний тип `Student` та структури (змінні) `s1` та `s2`.

Стилістично кращим вважається оголошувати змінні окремо від визначення типу.

В мові С ім'я структурного типу є тегом (міткою), яке потрібно використовувати лише разом з ключовим словом `struct`. Оголошення змінних-структур окремо від визначення структурного типу в мові С виглядає так:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };
```

```
struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
};

struct Student s1, s2;
```

В мові C++ структурний тип – це справжній тип і його ім'я є повноправним ідентифікатором (не тегом), тобто ім'я структурного типу можна використовувати при оголошенні змінних-структур без ключового слова `struct`:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
};

Student s1, s2;
```

Можна оголошувати структури неіменованих (анонімних) типів: ім'я типу не вказується, але можна оголошувати змінні:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
} s1, s2;
```

– змінні `s1` та `s2` – структури неіменованого структурного типу.

Як і для інших типів, для структурного можна оголошувати масиви структур та вказівники на структури:

```
Student s3[25], *ps;
```

– оголосили масив `s3` із `25` елементів типу `Student` та вказівник `ps` на структуру типу `Student`.

Вкладені структури

Одне (чи більше) з полів структури, в свою чергу може бути структурою – такі структури називаються вкладеними.

Для доступу до вкладеної структури необхідно вказати, що вкладена внутрішня структура належить до області видимості зовнішньої структури за допомогою операції :: (дві двокрапки) – операції видимості. Наприклад:

```
struct A
{
    int x;
    struct B
    {
        int y;
    } v2;
} v1;

A v3;
A::B v4;
```

Обмеження при визначенні структур

Єдиним обмеженням при визначенні структур є те, що не допускається рекурсія при визначенні структурних типів, – ні безпосередньо, ні через інші типи – не можна при визначенні поля використовувати ім'я цієї ж структури.

```
struct A
{
    A x;  // помилка
};
```

– помилка: поле x структури A не може мати тип цієї ж структури.

Єдиний виняток з цього правила – коли поле є вказівником на таку структуру:

```
struct A
{
    A *y;  // допускається
};
```

Розподіл пам'яті для структур

В пам'яті для структури виділяється неперервна область, розмір якої більший або дорівнює сумі розмірів всіх полів: є нюанс, який називається вирівнюванням – розмір області пам'яті, виділеної для структури, може бути більшим за суму розмірів полів цієї структури.

Поля, вказані при визначенні структури першими, розміщуються в комірках з молодшими адресами.

Розглянемо, що таке вирівнювання і як воно працює:

Для структури

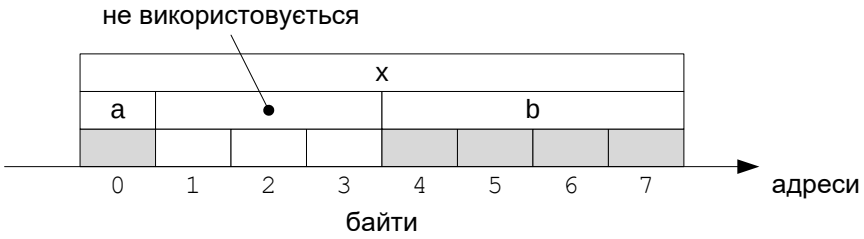
```
struct A
{
    char a;
    int  b;
} x;
```

її розмір `sizeof(A)` (чи `sizeof(x)`) не завжди буде дорівнювати 5 – все залежить від налаштування компілятора та директив в тексті програми.

Правило: поля в пам’яті вирівнюються по границі, кратній своєму розміру.

Тобто, 1-байтові поля не вирівнюються, 2-байтові – вирівнюються на позиції парних адрес, 4-байтові – на позиції адрес, кратних чотирьом і т.д.

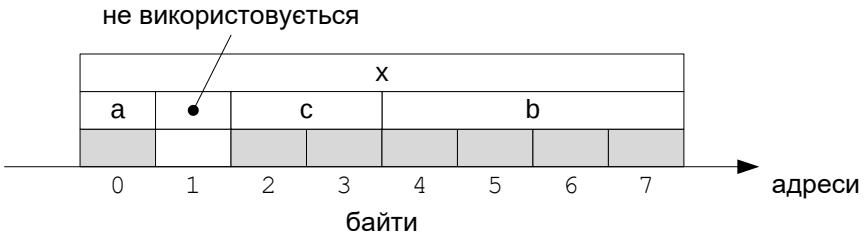
У більшості випадків за умовчанням вирівнювання розміру структури в пам’яті становить 4 байти. Таким чином, `sizeof(A) == 8`:



Розглянемо тепер розподіл пам’яті для наступної структури:

```
struct A
{
    char  a;
    short c;
    int   b;
} x;
```

– він виглядає так:



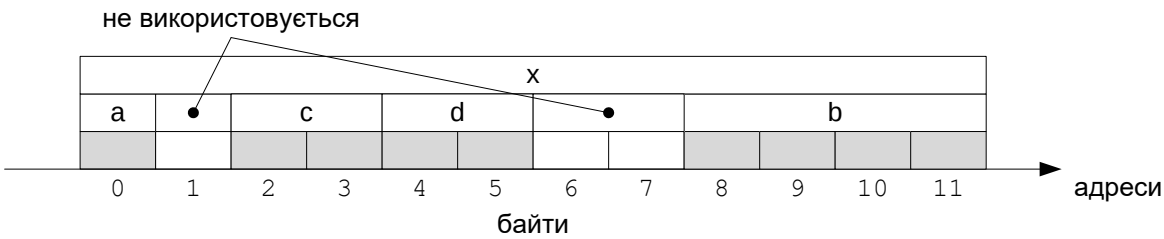
– нове поле не привело до збільшення розміру структури!

Те поле, яке можна розмістити до границі вирівнювання по адресі, кратній 4 байти – не приводить до збільшення розміру структури в пам’яті.

Добавимо ще одне поле:

```
struct A
{
    char  a;
    short c;
    short d;
    int   b;
} x;
```

Тепер поля розміщуються в пам’яті наступним чином:



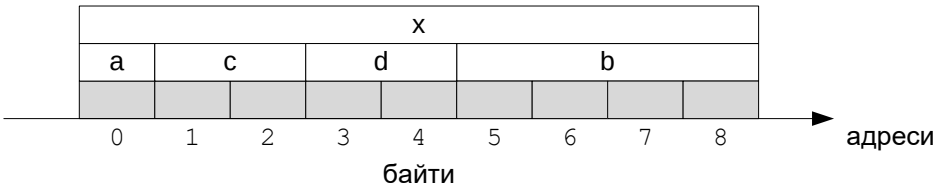
Є можливість керувати вирівнюванням безпосередньо в програмі:

```
#pragma pack(push, 1)

    struct A
    {
        char a;
        short c;
        short d;
        int b;
    } x;

#pragma pack(pop)
```

– встановили вирівнювання в 1 байт, визначили структуру та вернули попередні налаштування.

Повертати попередні налаштування – категорично рекомендується!

Тепер розподіл пам'яті виглядає так:



Ініціалізація структури

Ініціалізація структури виконується аналогічно до ініціалізації масиву:

```
enum Kurs { I = 1, II, III, IV, V, VI };
enum Spec { ME, MF, FI, IN, KN };

struct Student
{
    string   prizv;
    unsigned rikNar;
    Kurs     kurs;
    Spec     spec;
};

Student s = { "Іваненко", 1997, I, KN }; // ініціалізація
```

– створюємо структуру – змінну типу `Student` та присвоюємо значення всім її полям. Порядок дуже важливий при ініціалізації структури, бо компілятор встановлює позиційну відповідність між полями при визначенні структури та константами в ініціалізаторі – порядок значень в ініціалізаторі має збігатися з порядком полів при визначенні структури.

Якщо якесь поле залишилося не заповненим, то воно автоматично заповнюється значенням `0` – для цілих типів, `NULL` – для вказівників, `'\0'` (нуль-символ) – для літерних рядків.

Доступ до елементів структури

До елементів структури можна звертатися двома способами:

1) за допомогою імені структури;
2) за допомогою вказівника на структуру.

Доступ за іменем структури

Доступ до елементів структури за допомогою імені цієї структури здійснюється через операцію . (крапка), загальний вигляд – наступний:

```
структура.поле
```

Приклад:

```
#include <iostream>
#include <string>

using namespace std;

    enum Kurs { I = 1, II, III, IV, V, VI };
    enum Spec { ME, MF, FI, IN, KN };

    struct Student
    {
        string   prizv;
        unsigned rikNar;
        Kurs     kurs;
        Spec     spec;
    };

int main()
{
    Student s;

    s.prizv = "Іваненко";
    s.rikNar = 1997;
    s.kurs = I;
    s.spec = KN;

    return 0;
}
```

Доступ за вказівником на структуру

Є два способи доступу до елементів структури за допомогою вказівника на цю структуру.

Перший спосіб – традиційний, що використовує операцію роз-іменування вказівника * (зірочка), загальний вигляд – наступний:

```
(*вказвник-на-структуру).поле
```

Приклад:

```
#include <iostream>
#include <string>

using namespace std;
```

```
    enum Kurs { I = 1, II, III, IV, V, VI };
    enum Spec { ME, MF, FI, IN, KN };

    struct Student
    {
        string   prizv;
        unsigned rikNar;
        Kurs     kurs;
        Spec     spec;
    };

int main()
{
    Student *p = new Student;

    (*p).prizv = "Іваненко";
    (*p).rikNar = 1997;
    (*p).kurs = I;
    (*p).spec = KN;

    return 0;
}
```

– дужки тут обов'язкові, бо згідно правилу «суфікс важливіший префіксу», операція доступу `.` (крапка) має більш високий пріоритет, ніж операція роз'іменування `*` (зірочка).

Другий спосіб – більш зручний – використовує операцію `->` (стрілка), яка записується двома послідовними символами: `-` (мінус) та `>` (більше). Загальний вигляд:

```
вказівник-на-структуру -> поле
```

Приклад:

```
#include <iostream>
#include <string>

using namespace std;

    enum Kurs { I = 1, II, III, IV, V, VI };
    enum Spec { ME, MF, FI, IN, KN };

    struct Student
    {
        string   prizv;
        unsigned rikNar;
        Kurs     kurs;
        Spec     spec;
    };

int main()
{
    Student *p = new Student;

    p->prizv = "Іваненко";
    p->rikNar = 1997;
    p->kurs = I;
    p->spec = KN;

    return 0;
}
```

Операції над структурами

Є 5 операцій, які можна виконувати над структурами:

1) присвоювати полю структури значення відповідного типу;
2) присвоювати всій структурі значення того самого структурного типу;
3) отримувати адресу структури за допомогою операції `&`;
4) отримувати доступ до будь-якого поля структури;
5) визначати розмір структури та її полів за допомогою операції `sizeof()`.

Операції доступу розглядалися в попередньому параграфі.

Присвоєння.

Імена структурних змінних можна використовувати в якості операндів операції присвоєння. При цьому обидві структурні змінні (перед та після знаку присвоєння `=`) мають бути оголошені за допомогою одного і того самого структурного типу:

```
#include <iostream>
#include <string>

using namespace std;

    enum Kurs { I = 1, II, III, IV, V, VI };
    enum Spec { ME, MF, FI, IN, KN };

    struct Student
    {
        string   prizv;
        unsigned rikNar;
        Kurs     kurs;
        Spec     spec;
    };

int main()
{
    Student s1 = { "Іваненко", 1997, I, KN };
    Student s2 = s1;

    return 0;
}
```

Передавання структур у функції

Оскільки структурний тип – повноправний тип даних, лише створений нами, то з ним можна виконувати всі ті дії при передаванні у функції, що з іншими типами:

1) всю структуру можна передати у функцію:

```
    void f1( Student s )
    {
        cout << s.prizv << "  "
             << s.rikNar << "  "
             << s.kurs << "  "
```

```
            << s.spec << endl;
}
```

2) можна передати вказівник на структуру:

```
void f2( Student *s )
{
    cout << s->prizv << "  "
         << s->rikNar << "  "
         << s->kurs << "  "
         << s->spec << endl;
}
```

3) можна передавати елементи структури:

```
void f3( string p, unsigned rn, Kurs k, Spec s )
{
    cout << p << "  "
         << rn << "  "
         << k << "  "
         << s << endl;
}
```

4) можна повертати структури як результат функції:

```
Student f4( Student s )
{
    return s;
}
```

5) можна описати параметри-посилання на структуру – це забезпечить можливість передачі структур із функцій не як результат функції, а як вихідний (результуючий) параметр:

```
void f5( Student &s )
{
    // ...
}
```

Бітові поля

Багато задач потребують великої економії пам'яті. Одним із засобів економити пам'ять при використанні структурних типів є бітові поля.

Бітові поля представляють дані цілого типу і описуються наступним чином:

тип [*назва-поля*] : *розмір-в-бітах* ;

де

тип – один з цілих типів;

назва-поля – необов'язковий елемент. Якщо ім'я поля не вказане, до нього не можна звертатися. Такі неіменовані поля використовуються для того, щоб заповнити місце – для реалізації вирівнювання;

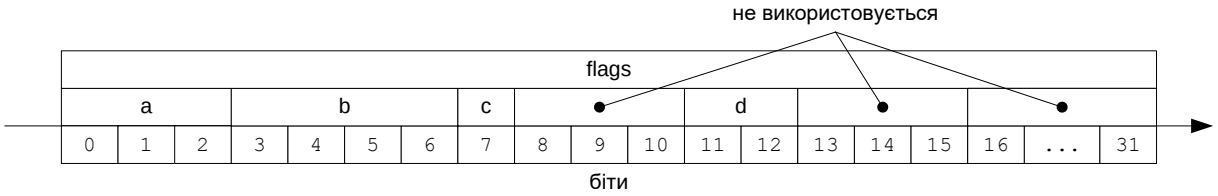
розмір-в-бітах – кількість біт, які виділяються для вказаного бітового поля. Якщо вказати значення `0`, то це гарантує, що наступне поле структури буде вирівняне таким чином, що його границя буде починатися з адреси, кратній розміру `int`.

Бітові поля описуються лише як складові частини структурного типу і розміщуються в напрямку від молодших до старших бітів.

Приклад:

```
struct BitFieldsInt
{
    int      a:3;
    unsigned b:4;
    int      c:1;
    int       :3;
    unsigned d:2;
} flagsInt;
```

– вказана структура забезпечує розміщення в пам'яті змінної `flagsInt`, яка містить бітові поля `a`, `b`, `c`, `d` – як показано на рисунку:



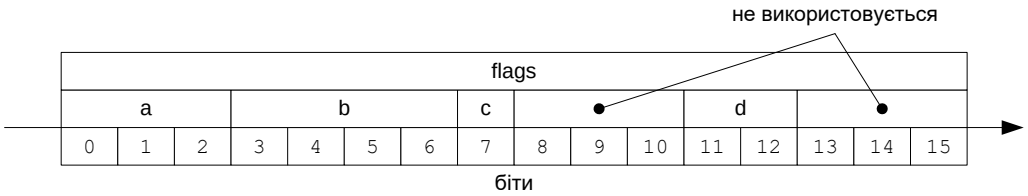
– оскільки бітові поля визначені на базі 4-байтових (32-розрядних) типів `int` та `unsigned` (скорочення від `unsigned int`), то базовим типом для таких бітових полів є 32-розрядний цілий тип.

Для примусового вирівнювання бітових полів до границі наступного базового цілого числа використовуються неіменовані поля довжиною `0`. Якщо в наведеній вище структурі `BitFieldsInt` описати неіменоване поле `int :3;` як `int :0;`, то поле `d` буде розміщено починаючи з біту 32 – в наступній групі з чотирьох байт, а вся структура `flagsInt` буде займати в пам'яті 8 байт.

Якщо оголосити структуру з бітовими полями на базі короткого цілого типу:

```
struct BitFieldsShort
{
    short a:3;
    short b:4;
    short c:1;
    short  :3;
    short d:2;
} flagsShort;
```

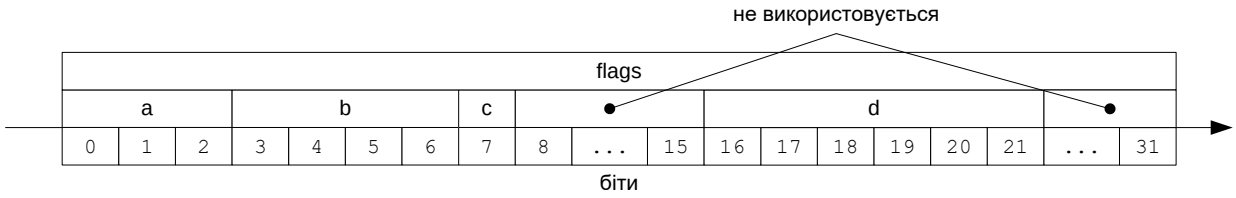
– то вона буде займати в пам'яті 2 байти:



– оскільки тепер базовим для таких бітових полів є 16-розрядний цілий тип.

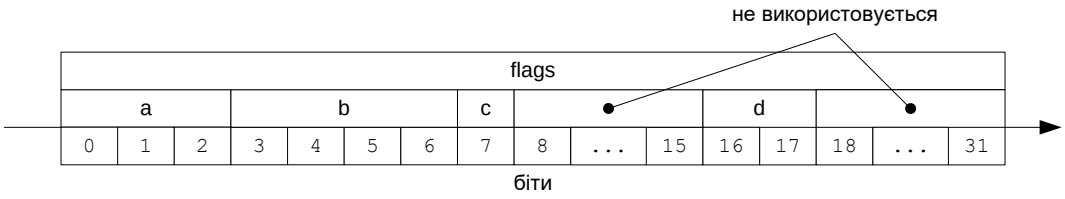
Бітові поля не можуть переходити через границі пам'яті цілого числа. Наприклад, якби поле `d` мало розмір понад `5` бітів, то воно було би розміщене в наступному слові (групі з двох байт):

```
struct BitFieldsShort
{
    short a:3;
    short b:4;
    short c:1;
    short  :3;
    short d:6;
} flagsShort;
```



Аналогічно, якщо описати неіменоване поле `int :3;` як `int :0;`, то поле `d` буде розміщено починаючи з наступного слова:

```
struct BitFieldsShort
{
    short a:3;
    short b:4;
    short c:1;
    short  :0;
    short d:2;
} flagsShort;
```



Бітові поля опрацьовуються як цілі поля базового типу.

Їх можна використовувати в будь-яких виразах і з будь-якими операціями, за винятком операції отримання адреси `&` – цю операцію до бітових полів застосовувати не можна!

Звертання до бітових полів здійснюється аналогічно звертанню до інших елементів структури:

```
flagsShort.a = 1;

if ( flagsShort.b == flagsShort.d )
    ...
```

Бітове поле не може бути масивом, проте допускаються масиви структур, які містять бітові поля.

Найчастіше бітові поля використовуються в якості набору перемикачів, які описують стан певного об’єкта. Наприклад, структура

```
struct
{
    unsigned Shift:1;
    unsigned Ins  :1;
    unsigned Caps :1;
} key;
```

дозволяє зберігати інформацію про стан клавіатури комп’ютера. При цьому вмикання і вимикання при знаків здійснюється зрозумілими командами присвоєння (а не логічними операціями):

```
key.Ins  = 1;
key.Caps = 0;
```

а перевірка стану:

```
if ( key.Shift==1 && key.Caps==1 )
    ...
```

Базові типи мають бути достатньо великими, щоб помістити бітове поле. Наприклад, наступні два поля оголошені не вірно:

```
struct Illegal
{
    short a:17;    // Illegal!
    long  b:33;    // Illegal!

} illegalFields;
```

Об’єднання

Вступ

Часто буває потрібно описати в програмі об’єкти, які мають один набір характеристик для об’єктів одної категорії та зовсім інший набір характеристик – для об’єктів іншої категорії, наприклад: юридичні та фізичні особи характеризуються різними наборами атрибутів, студенти різних спеціальностей вивчають різні предмети, працівники на різних посадах мають різні характеристики оплати праці (фіксований ставка чи погодинний тариф) тощо.

В таких випадках недоцільно використовувати структури для подання вказаних характеристик – бо деякі поля будуть використовуватися в одних випадках і ніколи не використовуватимуться – в інших, що приведе до неоправданих затрат пам’яті.

Зазначену проблему дозволяють усунути **об’єднувані типи** та **об’єднання**, які дозволяють розміщувати різні поля в одній і тій самій області пам’яті.

Визначення об’єднуваного типу

Як і структурний, об’єднуваний тип – це тип, що визначається користувачем та складається із набору елементів, які називаються *полями*.

Загальний вигляд команди визначення об’єднуваного типу та змінних – об’єднань (квадратні дужки використовуються для позначення необов’язкового елементу синтаксичної конструкції):

```
union [ім’я-об’єднуваного-типу]
{
    список-полів
} [список-змінних-об’єднань];
```

де

ім’я-об’єднуваного-типу – ім’я об’єднуваного типу, необов’язкове. Якщо ім’я об’єднуваного типу не вказане, то визначається анонімне об’єднання;

список-полів – список оголошень полів; оголошення поля аналогічне до оголошення змінних; *список-полів* має наступний загальний вигляд (квадратні дужки використовуються для позначення необов’язкового елементу синтаксичної конструкції):

```
ім’я-типу ім’я-поля [ , ..., ім’я-поля ];
[...
ім’я-типу ім’я-поля [ , ..., ім’я-поля ];]
```

В об’єднанні має бути по крайній мірі одне поле.

Імена полів в об’єднанні мають бути унікальними, але ім’я поля цього об’єднання можна використовувати як ідентифікатор іншого ресурсу – змінної, типу, функції, поля іншого об’єднання чи структури. В різних об’єднаннях можна використовувати однакові імена полів.

список-змінних-об’єднань – оголошення змінних-об’єднань розглядається в наступному параграфі.

Команда визначення об’єднуваного типу – це команда визначення даних, яка містить блок визначення даних (елементів об’єднання). Кожний блок визначення даних (на відміну від блоку опису команд) має закінчуватися символом «;» (крапка з комою).

Приклад:

```
union Oplata
{
    int    stavka;
    double taryf;
};
```

Оголошення та розподіл пам’яті для об’єднань – змінних об’єднуваних типів

Оголошення об’єднань

Як і для переліків та структур, змінні-об’єднання (змінні об’єднуваних типів) можна оголосити в команді визначення об’єднуваного типу:

```
union Oplata
{
    int    stavka;
    double taryf;
} o1, o2;
```

– визначається об’єднуваний тип `Oplata` та об’єднання (змінні) `o1` та `o2`.

Стилістично кращим вважається оголошувати змінні окремо від визначення типу.

В мові С ім’я об’єднуваного типу є тегом (міткою), яке потрібно використовувати лише разом з ключовим словом `union`. Оголошення змінних-об’єднань окремо від визначення об’єднуваного типу в мові С виглядає так:

```
union Oplata
{
    int    stavka;
    double taryf;
};

union Oplata o1, o2;
```

В мові С++ об’єднуваний тип – це справжній тип і його ім’я є повноправним ідентифікатором (не тегом), тобто ім’я об’єднуваного типу можна використовувати при оголошенні змінних-об’єднань без ключового слова `union`:

```
union Oplata
{
    int    stavka;
    double taryf;
};

Oplata o1, o2;
```

Можна оголошувати об’єднання неіменованих (анонімних) типів: ім’я типу не вказується, але можна оголошувати змінні:

```
union
{
    int    stavka;
    double taryf;
} o1, o2;
```

– змінні `o1` та `o2` – об’єднання неіменованого об’єднуваного типу.

Як і для інших типів, для об’єднуваного можна оголошувати масиви об’єднань та вказівники на об’єднання:

```
Oplata o3[25], *po;
```

– оголосили масив `o3` із `25` елементів типу `Oplata` та вказівник `po` на об’єднання типу `Oplata`.

Вкладені об’єднання

Як і для структури, так і для об’єднання, одне (чи більше) з полів певної структури (певного об’єднання), в свою чергу може бути структурою чи об’єднанням – такі структури чи об’єднання називаються вкладеними.

Все, що стосувалося вкладених структур, стосується і вкладених об’єднань.

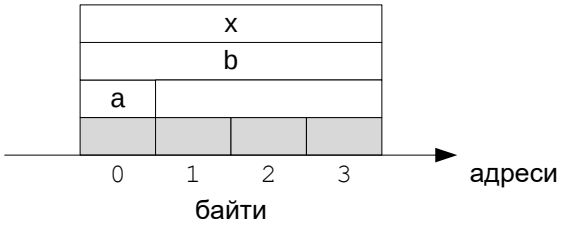
Розподіл пам’яті для об’єднань

В пам’яті для об’єднання виділяється неперервна область, розмір якої дорівнює розміру найбільшого поля.

Для об’єднання

```
union A
{
    char a;
    int  b;
} x;
```

його розмір `sizeof(A)` (чи `sizeof(x)`) буде дорівнювати `4`:



– поле `b` об’єднання `x` типу `A` дає доступ до всіх його 4-х байт і трактує вміст цієї області як ціле число; поле `a` об’єднання `x` дає доступ лише до самого першого байту (інші байти об’єднання цим полем ігноруються) і трактує вміст цього байту як код символу.

Ініціалізація об’єднання

Ініціалізація об’єднання виконується аналогічно до ініціалізації масиву чи структури, проте в команді ініціалізації можна надати значення лише першому полю об’єднання:

```
union A
{
    char a;
    int  b;
};

A x = { 'a' };

cout << x.a << endl;
cout << x.b << endl;
```

Вивід:

```
a
97
```

– ініціалізуємо об’єднання символом `'a'`.

За допомогою поля `b` ми трактуємо символ як ціле число (код цього символу), тому друга команда `cout` виведе значення `97`.

Наступний спосіб ініціалізації – хибний (деякі системи виведуть попередження, а деякі – повідомлення про помилку):

```
union A
{
    char a;
    int  b;
};

A x = { 10000 };
```

– значення `10000` буде «обрізане», щоб помістити в один байт.

Доступ до елементів об’єднання

Доступ до елементів об’єднання аналогічний доступу до елементів структури.

Операції над об’єднаннями

Над об’єднаннями можна виконувати ті самі операції, що і над структурами.

Передавання об’єднань у функції

Об’єднання передаються у / із функції аналогічно структурам.

Визначення та еквівалентність типів

Визначення синоніму типу – команда `typedef`

Команда **typedef** визначає синонім типу. Її загальний вигляд:

```
typedef визначення-типу нове-ім'я-типу;
```

де

визначення-типу – будь-яке допустиме в C++ визначення типу;

нове-ім'я-типу – синонім (нове ім'я), що надається цьому типу.

Приклад:

```
int a[10];
```

– оголошує змінну `a` як масив із `10` елементів цілого типу.

Використаємо команду визначення синоніму типу `typedef`:

```
typedef int A[10];
A a;
```

– визначає тип `A` як сукупність масивів із `10` елементів цілого типу та оголошує змінну `a` цього типу.

Таким чином, ключове слово `typedef` перетворює команду оголошення змінної в команду визначення синоніму типу.

Команда визначення синоніму типу `typedef` використовується для того, щоб замінити складні та громіздкі оголошення простішими. Зокрема, в мові C імена переліків, структур та об'єднань – не повноправні імена типів, а лише теги, які в оголошенні змінних, описі параметрів та результату функцій слід використовувати лише разом із ключовими словами `enum`, `struct` та `union` відповідно. В таких випадках і використовувалася команда `typedef` для визначення більш компактного імені.

На відміну від оголошень `enum`, `struct` та `union`, команда `typedef` не вводить нового типу, а лише призначає нове ім'я вже визначеному типу.

Еквівалентність типів

Існує кілька схем для визначення, чи еквівалентні типи двох об'єктів. Найчастіше використовуються схеми, що називаються **структурна еквівалентність типів** та **іменна еквівалентність типів** (в англомовній термінології *structural type system* та *nominative type system* відповідно). Відповідно до схеми структурної еквівалентності типів два об'єкти належать до одного типу, якщо їх компоненти мають однакові типи. Відповідно до схеми іменної еквівалентності типів два об'єкти мають один і той самий тип лише тоді, коли вони визначені за допомогою імені цього ж типу.

Таким чином, відповідно до схеми іменної еквівалентності типів, два структурні типи будуть різними навіть тоді, коли вони мають одні і ті самі компоненти. Наприклад:

```
struct s1 { int a; };
struct s2 { int a; };
```

– визначено два різні типи. Змінні цих типів – несумісні:

```
s1 x;
s2 y = x;    // помилка: невідповідність типів
```

Структурні типи – відрізняються від простих, тому наступне присвоєння – помилкове:

```
s1 x;
int i = x;    // помилка: невідповідність типів
```

В мові C / C++ діє правило:

Окремі визначення типів визначають різні типи (навіть якщо ці визначення буквально ідентичні).

Мови C та C++ побудовані на жорсткій іменній еквівалентності типів, хоча окремі елементи структурної еквівалентності типів є в мові C.

Команда визначення синоніму типу `typedef` реалізує схему іменної еквівалентності типів – вона дозволяє задати нове ім’я типу, не визначаючи новий тип: в оголошенні, яке починається ключовим словом `typedef`, описується не змінна, а вводиться нове ім’я для типу.

Приклади

Оголошення структури з об’єднанням

Описати тип *Працівник*, який характеризується наступними даними:

прізвище – літерний рядок;

рік_народження – без-знакове ціле;

посада – перелік із значень: *робітник*, *інженер*, *службовець*;

оплата – для посади *робітник* – погодинний *тариф*, – дійсне число;

для посад *інженер* та *службовець* – фіксована *ставка* – ціле число.

Створити динамічну змінну цього типу та присвоїти її полям відповідні значення.

```
// гірший спосіб
#include <iostream>
#include <string>
using namespace std;

enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };

struct Pracivnyk
{
    string   prizv;
    unsigned rik_nar;
    Posada   posada;
    union Oplata
    {
        double taryf;
        int    stavka;
    } oplata;
};

int main()
{
    Pracivnyk* p;
    p = new Pracivnyk;

    p->prizv = "Іваненко";
    p->rik_nar = 1995;
    p->posada = ROBITNYK;
    p->oplata.taryf = 3.14;

    return 0;
}
```

– як бачимо, створення окремого поля `oplata` приводить до того, що спосіб доступу до поля `taryf` відрізняється від доступу до інших полів – цей спосіб занадто громіздкий.

Проблему можна усунути, якщо не надавати об’єднанню імені (використовувати анонімне об’єднання) та не створювати окремого поля об’єднуваного типу:

```
// кращий спосіб
#include <iostream>
#include <string>
```

```
using namespace std;

enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };

struct Pracivnyk
{
    string   prizv;
    unsigned rik_nar;
    Posada   posada;
    union
    {
        double taryf;
        int    stavka;
    };
};

int main()
{
    Pracivnyk* p;
    p = new Pracivnyk;

    p->prizv = "Іваненко";
    p->rik_nar = 1995;
    p->posada = ROBITNYK;
    p->taryf = 3.14;

    return 0;
}
```

– в цьому випадку неіменоване об’єднання в складі структури `Pracivnyk` означає, що поля об’єднання `taryf` та `stavka` спільно використовують одну і ту саму область пам’яті.

Формування динамічного масиву структур з об’єднаннями

Сформувати масив структур з об’єднаннями, що містять інформацію про працівників, які характеризуються наступними даними:

прізвище – літерний рядок;

рік_народження – беззнакове ціле;

посада – перелік із значень: *робітник*, *інженер*, *службовець*;

оплата – для посади *робітник* – погодинний *тариф*, – дійсне число;

для посад *інженер* та *службовець* – фіксована *ставка* – ціле число.

Використовувати динамічний масив структур – кількість працівників вводить користувач при виконанні програми.

Продовжимо попередню програму, змінимо функцію `main()` та додамо функцію `Create()`:

```
#include <iostream>
#include <string>
#include <Windows.h>            // забезпечення відображення кирилиці

using namespace std;
```

```
    enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };

    struct Pracivnyk
    {
        string   prizv;
        unsigned rik_nar;
        Posada   posada;
        union
        {
            double taryf;
            int    stavka;
        };
    };

void Create(Pracivnyk *p, const int N);

int main()
{                               // забезпечення відображення кирилиці:
    SetConsoleCP(1251);         // встановлення сторінки win-cp1251 в потік вводу
    SetConsoleOutputCP(1251); // встановлення сторінки win-cp1251 в потік виводу

    int N;
    cout << "Введіть N: "; cin >> N;

    Pracivnyk *p = new Pracivnyk[N];

    Create(p, N);

    return 0;
}

void Create(Pracivnyk *p, const int N)
{
    int posada;
    for (int i=0; i<N; i++)
    {
        cout << "Працівник № " << i+1 << ":" << endl;

        cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
        cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

        cout << "    прізвище: "; getline(cin, p[i].prizv);
        cout << "    рік нар.: "; cin >> p[i].rik_nar;
        cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
        cin >> posada;
        p[i].posada = (Posada)posada;
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "    тариф  : "; cin >> p[i].taryf;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "    ставка : "; cin >> p[i].stavka;
            break;
        }
        cout << endl;
    }
}
```

Вивід масиву структур з об’єднаннями у вигляді таблиці

Вивести на екран у вигляді таблиці масив структур з об’єднаннями, що містять наступні дані про працівників:

прізвище – літерний рядок;

рік_народження – беззнакове ціле;

посада – перелік із значень: *робітник*, *інженер*, *службовець*;

оплата – для посади *робітник* – погодинний *тариф*, – дійсне число;

для посад *інженер* та *службовець* – фіксована *ставка* – ціле число.

Використовувати динамічний масив структур – кількість працівників вводить користувач при виконанні програми.

Продовжимо попередню програму, добавимо функцію `Print()`:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <Windows.h>            // забезпечення відображення кирилиці
using namespace std;

    enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };

    string posadaStr[] = { "робітник", "інженер", "службовець" };

    struct Pracivnyk
    {
        string   prizv;
        unsigned rik_nar;
        Posada   posada;
        union
        {
            double taryf;
            int    stavka;
        };
    };

void Create(Pracivnyk *p, const int N);
void Print(Pracivnyk *p, const int N);

int main()
{                               // забезпечення відображення кирилиці:
    SetConsoleCP(1251);         // встановлення сторінки win-cp1251 в потік вводу
    SetConsoleOutputCP(1251); // встановлення сторінки win-cp1251 в потік виводу

    int N;
    cout << "Введіть N: "; cin >> N;

    Pracivnyk *p = new Pracivnyk[N];

    Create(p, N);
    Print(p, N);

    return 0;
}
```

```
void Create(Pracivnyk *p, const int N)
{
    int posada;
    for (int i=0; i<N; i++)
    {
        cout << "Працівник № " << i+1 << ":" << endl;

        cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
        cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

        cout << "    прізвище: "; getline(cin, p[i].prizv);
        cout << "    рік нар.: "; cin >> p[i].rik_nar;
        cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
        cin >> posada;
        p[i].posada = (Posada)posada;
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "    тариф  : "; cin >> p[i].taryf;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "    ставка : "; cin >> p[i].stavka;
            break;
        }
        cout << endl;
    }
}

void Print(Pracivnyk *p, const int N)
{
    cout << "========================================================================="
         << endl;
    cout << "|  №  |   Прізвище   |  Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
         << endl;
    cout << "-------------------------------------------------------------------------"
         << endl;

    for (int i=0; i<N; i++)
    {
        cout << "| "    << setw(3)  << right << i+1 << " ";
        cout << "| "    << setw(13) << left <<  p[i].prizv
             << "|    " << setw(4)  << right << p[i].rik_nar << "    "
             << "| "    << setw(11) << left <<  posadaStr[ p[i].posada ];

        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "| " << setw(9) << setprecision(2) << fixed << right
                 << p[i].taryf << " |" << setw(13) << right << "|" << endl;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                 << p[i].stavka << "  |" << endl;
           break;
        }
    }
    cout << "========================================================================="
         << endl;
    cout << endl;
}
```

Послідовний пошук в масиві структур з об’єднаннями

В динамічному масиві структур з об’єднаннями, які містять наступну інформацію про працівників:

прізвище – літерний рядок;

рік_народження – беззнакове ціле;

посада – перелік із значень: *робітник*, *інженер*, *службовець*;

оплата – для посади *робітник* – погодинний *тариф*, – дійсне число;

для посад *інженер* та *службовець* – фіксована *ставка* – ціле число

знайти процент робітників, чий тариф перевищує 3,14 та вивести їх прізвища.

Продовжимо попередню програму, добавимо функцію `LineSearch()`, яка реалізує послідовний (лінійний) пошук в масиві:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <Windows.h>            // забезпечення відображення кирилиці
using namespace std;

    enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };

    string posadaStr[] = { "робітник", "інженер", "службовець" };

    struct Pracivnyk
    {
        string   prizv;
        unsigned rik_nar;
        Posada   posada;
        union
        {
            double taryf;
            int    stavka;
        };
    };

void Create(Pracivnyk *p, const int N);
void Print(Pracivnyk *p, const int N);
double LineSearch(Pracivnyk *p, const int N);

int main()
{                               // забезпечення відображення кирилиці:
    SetConsoleCP(1251);         // встановлення сторінки win-cp1251 в потік вводу
    SetConsoleOutputCP(1251); // встановлення сторінки win-cp1251 в потік виводу

    int N;
    cout << "Введіть N: "; cin >> N;

    Pracivnyk *p = new Pracivnyk[N];

    Create(p, N);
    Print(p, N);

    double proc = LineSearch(p, N);
    cout << "Процент робітників, чий тариф перевищує 3,14:" << endl;
```

```
    cout << proc << "%" << endl;
    return 0;
}

void Create(Pracivnyk *p, const int N)
{
    int posada;
    for (int i=0; i<N; i++)
    {
        cout << "Працівник № " << i+1 << ":" << endl;

        cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
        cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

        cout << "    прізвище: "; getline(cin, p[i].prizv);
        cout << "    рік нар.: "; cin >> p[i].rik_nar;
        cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
        cin >> posada;
        p[i].posada = (Posada)posada;
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "    тариф  : "; cin >> p[i].taryf;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "    ставка : "; cin >> p[i].stavka;
            break;
        }
        cout << endl;
    }
}

void Print(Pracivnyk *p, const int N)
{
    cout << "========================================================================="
         << endl;
    cout << "|  №  |   Прізвище   |  Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
         << endl;
    cout << "-------------------------------------------------------------------------"
         << endl;
    for (int i=0; i<N; i++)
    {
        cout << "| "    << setw(3)  << right << i+1 << " ";
        cout << "| "    << setw(13) << left <<  p[i].prizv
             << "|    " << setw(4)  << right << p[i].rik_nar << "    "
             << "| "    << setw(11) << left <<  posadaStr[ p[i].posada ];
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "| " << setw(9) << setprecision(2) << fixed << right
                 << p[i].taryf << " |" << setw(13) << right << "|" << endl;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                 << p[i].stavka << "  |" << endl;
           break;
        }
    }
    cout << "========================================================================="
         << endl;
```

```
    cout << endl;
}

double LineSearch(Pracivnyk *p, const int N)
{
    cout << "Прізвища робітників, чий тариф перевищує 3,14:" << endl;
    int k=0, n=0;
    for (int i=0; i<N; i++)
    {
        if ( p[i].posada == ROBITNYK )
        {
            n++;
            if ( p[i].taryf > 3.14 )
            {
                k++;
                cout << setw(3) << right << k
                     << " " << p[i].prizv << endl;
            }
        }
    }
    return 100.0*k/n;
}
```

Організація меню в текстовому режимі

Переробимо попередню програму, добавивши меню у функцію `main()`:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <Windows.h>            // забезпечення відображення кирилиці
using namespace std;

    enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };
    string posadaStr[] = { "робітник", "інженер", "службовець" };

    struct Pracivnyk
    {
        string   prizv;
        unsigned rik_nar;
        Posada   posada;
        union
        {
            double taryf;
            int    stavka;
        };
    };

void Create(Pracivnyk *p, const int N);
void Print(Pracivnyk *p, const int N);
double LineSearch(Pracivnyk *p, const int N);

int main()
{                               // забезпечення відображення кирилиці:
    SetConsoleCP(1251);         // встановлення сторінки win-cp1251 в потік вводу
    SetConsoleOutputCP(1251); // встановлення сторінки win-cp1251 в потік виводу

    int N;
    cout << "Введіть кількість працівників N: "; cin >> N;

    Pracivnyk *p = new Pracivnyk[N];
```

```
    double proc;

    int menuItem;
    do {
        cout << endl << endl << endl;
        cout << "Виберіть дію:" << endl << endl;
        cout << "  [1] - введення даних з клавіатури" << endl;
        cout << "  [2] - вивід даних на екран" << endl;
        cout << "  [3] - вивід прізвищ та проценту робітників," << endl;
        cout << "        чий тариф перевищує 3,14" << endl << endl;
        cout << "  [0] - вихід та завершення роботи програми" << endl << endl;
        cout << "Введіть значення: "; cin >> menuItem;
        cout << endl << endl << endl;

        switch ( menuItem )
        {
        case 1:
            Create(p, N);
            break;
        case 2:
            Print(p, N);
            break;
        case 3:
            proc = LineSearch(p, N);
            cout << "Процент робітників, чий тариф перевищує 3,14:" << endl;
            cout << proc << "%" << endl;
            break;
        case 0:
            break;
        default:
            cout << "Ви ввели помилкове значення! "
                    "Слід ввести число - номер вибраного пункту меню" << endl;
        }
    } while ( menuItem != 0 );

    return 0;
}

void Create(Pracivnyk *p, const int N)
{
    int posada;
    for (int i=0; i<N; i++)
    {
        cout << "Працівник № " << i+1 << ":" << endl;

        cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
        cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

        cout << "    прізвище: "; getline(cin, p[i].prizv);
        cout << "    рік нар.: "; cin >> p[i].rik_nar;
        cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
        cin >> posada;
        p[i].posada = (Posada)posada;
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "    тариф  : "; cin >> p[i].taryf;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "    ставка : "; cin >> p[i].stavka;
```

```
            break;
        }
        cout << endl;
    }
}

void Print(Pracivnyk *p, const int N)
{
    cout << "========================================================================="
         << endl;
    cout << "|  №  |   Прізвище   |  Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
         << endl;
    cout << "-------------------------------------------------------------------------"
         << endl;
    for (int i=0; i<N; i++)
    {
        cout << "| "    << setw(3)  << right << i+1 << " ";
        cout << "| "    << setw(13) << left <<  p[i].prizv
             << "|    " << setw(4)  << right << p[i].rik_nar << "    "
             << "| "    << setw(11) << left <<  posadaStr[ p[i].posada ];
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "| " << setw(9) << setprecision(2) << fixed << right
                 << p[i].taryf << " |" << setw(13) << right << "|" << endl;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                 << p[i].stavka << "  |" << endl;
           break;
        }
    }
    cout << "========================================================================="
         << endl;
    cout << endl;
}

double LineSearch(Pracivnyk *p, const int N)
{
    cout << "Прізвища робітників, чий тариф перевищує 3,14:" << endl;
    int k=0, n=0;
    for (int i=0; i<N; i++)
    {
        if ( p[i].posada == ROBITNYK )
        {
            n++;
            if ( p[i].taryf > 3.14 )
            {
                k++;
                cout << setw(3) << right << k
                     << " " << p[i].prizv << endl;
            }
        }
    }
    return 100.0*k/n;
}
```

Фізичне впорядкування масиву структур з об’єднаннями

Створити динамічний масив структур з об’єднаннями, які описують наступні дані про працівників:

прізвище – літерний рядок;

рік_народження – беззнакове ціле;

посада – перелік із значень: *робітник*, *інженер*, *службовець*;

оплата – для посади *робітник* – погодинний *тариф*, – дійсне число;

для посад *інженер* та *службовець* – фіксована *ставка* – ціле число.

Реалізувати фізичне впорядкування вказаного масиву за наступними ключами: в першу чергу – за посадою (в порядку: спочатку – робітники, потім – інженери, в кінці – службовці); в другу чергу (для однакових посад) – за прізвищем за зростанням – в алфавітному порядку.

Фізичне впорядкування означає фактичну (реальну) зміну порядку слідування елементів – всі способи сортування масивів, які розглядалися при вивченні тем «Одновимірні масиви» та «Багатовимірні масиви», реалізовували фізичне впорядкування.

Продовжимо попередню програму, добавимо новий пункт меню та нову функцію `Sort()`, яка реалізує фізичне впорядкування:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <Windows.h>            // забезпечення відображення кирилиці
using namespace std;

    enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };
    string posadaStr[] = { "робітник", "інженер", "службовець" };

    struct Pracivnyk
    {
        string   prizv;
        unsigned rik_nar;
        Posada   posada;
        union
        {
            double taryf;
            int    stavka;
        };
    };

void Create(Pracivnyk *p, const int N);
void Print(Pracivnyk *p, const int N);
double LineSearch(Pracivnyk *p, const int N);
void Sort(Pracivnyk *p, const int N);

int main()
{                               // забезпечення відображення кирилиці:
    SetConsoleCP(1251);         // встановлення сторінки win-cp1251 в потік вводу
    SetConsoleOutputCP(1251); // встановлення сторінки win-cp1251 в потік виводу
```

```
    int N;
    cout << "Введіть кількість працівників N: "; cin >> N;

    Pracivnyk *p = new Pracivnyk[N];
    double proc;

    int menuItem;
    do {
        cout << endl << endl << endl;
        cout << "Виберіть дію:" << endl << endl;
        cout << "  [1] - введення даних з клавіатури" << endl;
        cout << "  [2] - вивід даних на екран" << endl;
        cout << "  [3] - вивід прізвищ та проценту робітників," << endl;
        cout << "        чий тариф перевищує 3,14" << endl;
        cout << "  [4] - фізичне впорядкування даних" << endl << endl;
        cout << "  [0] - вихід та завершення роботи програми" << endl << endl;
        cout << "Введіть значення: "; cin >> menuItem;
        cout << endl << endl << endl;

        switch ( menuItem )
        {
        case 1:
            Create(p, N);
            break;
        case 2:
            Print(p, N);
            break;
        case 3:
            proc = LineSearch(p, N);
            cout << "Процент робітників, чий тариф перевищує 3,14:" << endl;
            cout << proc << "%" << endl;
            break;
        case 4:
            Sort(p, N);
            break;
        case 0:
            break;
        default:
            cout << "Ви ввели помилкове значення! "
                    "Слід ввести число - номер вибраного пункту меню" << endl;
        }
    } while ( menuItem != 0 );

    return 0;
}

void Create(Pracivnyk *p, const int N)
{
    int posada;
    for (int i=0; i<N; i++)
    {
        cout << "Працівник № " << i+1 << ":" << endl;

        cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
        cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

        cout << "    прізвище: "; getline(cin, p[i].prizv);
        cout << "    рік нар.: "; cin >> p[i].rik_nar;
        cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
        cin >> posada;
        p[i].posada = (Posada)posada;
        switch ( p[i].posada )
```

```
        {
        case ROBITNYK:
            cout << "    тариф  : "; cin >> p[i].taryf;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "    ставка : "; cin >> p[i].stavka;
            break;
        }
        cout << endl;
    }
}

void Print(Pracivnyk *p, const int N)
{
    cout << "========================================================================="
         << endl;
    cout << "|  №  |   Прізвище   | Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
         << endl;
    cout << "-------------------------------------------------------------------------"
         << endl;
    for (int i=0; i<N; i++)
    {
        cout << "| "    << setw(3)  << right << i+1 << " ";
        cout << "| "    << setw(13) << left <<  p[i].prizv
             << "|    " << setw(4)  << right << p[i].rik_nar << "    "
             << "| "    << setw(11) << left <<  posadaStr[ p[i].posada ];
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "| " << setw(9) << setprecision(2) << fixed << right
                 << p[i].taryf << " |" << setw(13) << right << "|" << endl;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                 << p[i].stavka << "  |" << endl;
           break;
        }
    }
    cout << "========================================================================="
         << endl;
    cout << endl;
}

double LineSearch(Pracivnyk *p, const int N)
{
    cout << "Прізвища робітників, чий тариф перевищує 3,14:" << endl;
    int k=0, n=0;
    for (int i=0; i<N; i++)
    {
        if ( p[i].posada == ROBITNYK )
        {
            n++;
            if ( p[i].taryf > 3.14 )
            {
                k++;
                cout << setw(3) << right << k
                     << " " << p[i].prizv << endl;
            }
        }
    }
```

```
    return 100.0*k/n;
}

void Sort(Pracivnyk *p, const int N)
{
    Pracivnyk tmp;
    for (int i0=0; i0<N-1; i0++)        // метод "бульбашки"
    for (int i1=0; i1<N-i0-1; i1++)
        if (( p[i1].posada >  p[i1+1].posada )
            ||
            ( p[i1].posada == p[i1+1].posada &&
              p[i1].prizv >  p[i1+1].prizv ))
        {
            tmp = p[i1];
            p[i1] = p[i1+1];
            p[i1+1] = tmp;
        }
}
```

Бінарний пошук в масиві структур з об’єднаннями

Створити динамічний масив структур з об’єднаннями, які описують наступні дані про працівників:

прізвище – літерний рядок;

рік_народження – беззнакове ціле;

посада – перелік із значень: *робітник*, *інженер*, *службовець*;

оплата – для посади *робітник* – погодинний *тариф*, – дійсне число;

для посад *інженер* та *службовець* – фіксована *ставка* – ціле число.

Реалізувати бінарний пошук у попередньо впорядкованому вказаному масиві структур з об’єднаннями за парою ключів (посада, прізвище), – тобто, за допомогою бінарного пошуку визначити, чи міститься в масиві інформація про працівника з вказаним прізвищем, який перебуває на вказаній посаді.

Продовжимо попередню програму, добавимо новий пункт меню та нову функцію `BinSearch()`, яка реалізує бінарний пошук:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <Windows.h>            // забезпечення відображення кирилиці
using namespace std;

    enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };
    string posadaStr[] = { "робітник", "інженер", "службовець" };

    struct Pracivnyk
    {
        string   prizv;
        unsigned rik_nar;
        Posada   posada;
        union
        {
            double taryf;
```

```
            int    stavka;
        };
    };

void Create(Pracivnyk *p, const int N);
void Print(Pracivnyk *p, const int N);
double LineSearch(Pracivnyk *p, const int N);
void Sort(Pracivnyk *p, const int N);
int BinSearch(Pracivnyk *p, const int N, const string prizv, const Posada posada);

int main()
{                              // забезпечення відображення кирилиці:
    SetConsoleCP(1251);        // встановлення сторінки win-cp1251 в потік вводу
    SetConsoleOutputCP(1251); // встановлення сторінки win-cp1251 в потік виводу

    int N;
    cout << "Введіть кількість працівників N: "; cin >> N;

    Pracivnyk *p = new Pracivnyk[N];

    double proc;
    int iposada;
    Posada posada;
    string prizv;
    int found;

    int menuItem;
    do {
        cout << endl << endl << endl;
        cout << "Виберіть дію:" << endl << endl;
        cout << "  [1] - введення даних з клавіатури" << endl;
        cout << "  [2] - вивід даних на екран" << endl;
        cout << "  [3] - вивід прізвищ та проценту робітників," << endl;
        cout << "        чий тариф перевищує 3,14" << endl;
        cout << "  [4] - фізичне впорядкування даних" << endl;
        cout << "  [5] - бінарний пошук працівника за посадою та прізвищем"
             << endl << endl;
        cout << "  [0] - вихід та завершення роботи програми" << endl << endl;
        cout << "Введіть значення: "; cin >> menuItem;
        cout << endl << endl << endl;

        switch ( menuItem )
        {
        case 1:
            Create(p, N);
            break;
        case 2:
            Print(p, N);
            break;
        case 3:
            proc = LineSearch(p, N);
            cout << "Процент робітників, чий тариф перевищує 3,14:" << endl;
            cout << proc << "%" << endl;
            break;
        case 4:
            Sort(p, N);
            break;
        case 5:
            cout << "Введіть ключі пошуку:" << endl;
            cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
            cin >> iposada;
            posada = (Posada)iposada;
```

```
            cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
            cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

            cout << "    прізвище: "; getline(cin, prizv);
            cout << endl;
            if ( (found = BinSearch(p, N, prizv, posada)) != -1 )
                cout << "Знайдено працівника в позиції " << found+1 << endl;
            else
                cout << "Шуканого працівника не знайдено" << endl;
            break;
        case 0:
            break;
        default:
            cout << "Ви ввели помилкове значення! "
                    "Слід ввести число - номер вибраного пункту меню" << endl;
        }
    } while ( menuItem != 0 );

    return 0;
}

void Create(Pracivnyk *p, const int N)
{
    int posada;
    for (int i=0; i<N; i++)
    {
        cout << "Працівник № " << i+1 << ":" << endl;

        cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
        cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

        cout << "    прізвище: "; getline(cin, p[i].prizv);
        cout << "    рік нар.: "; cin >> p[i].rik_nar;
        cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
        cin >> posada;
        p[i].posada = (Posada)posada;
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "    тариф  : "; cin >> p[i].taryf;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "    ставка : "; cin >> p[i].stavka;
            break;
        }
        cout << endl;
    }
}

void Print(Pracivnyk *p, const int N)
{
    cout << "========================================================================="
         << endl;
    cout << "|  №  |   Прізвище   |  Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
         << endl;
    cout << "-------------------------------------------------------------------------"
         << endl;
    for (int i=0; i<N; i++)
    {
        cout << "| "    << setw(3)  << right << i+1 << " ";
        cout << "| "    << setw(13) << left <<  p[i].prizv
```

```
             << "|    " << setw(4)  << right << p[i].rik_nar << "    "
             << "| "    << setw(11) << left <<  posadaStr[ p[i].posada ];
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "| " << setw(9) << setprecision(2) << fixed << right
                 << p[i].taryf << " |" << setw(13) << right << "|" << endl;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                 << p[i].stavka << "  |" << endl;
           break;
        }
    }
    cout << "========================================================================="
         << endl;
    cout << endl;
}

double LineSearch(Pracivnyk *p, const int N)
{
    cout << "Прізвища робітників, чий тариф перевищує 3,14:" << endl;
    int k=0, n=0;
    for (int i=0; i<N; i++)
    {
        if ( p[i].posada == ROBITNYK )
        {
            n++;
            if ( p[i].taryf > 3.14 )
            {
                k++;
                cout << setw(3) << right << k
                     << " " << p[i].prizv << endl;
            }
        }
    }
    return 100.0*k/n;
}

void Sort(Pracivnyk *p, const int N)
{
    Pracivnyk tmp;
    for (int i0=0; i0<N-1; i0++)        // метод "бульбашки"
    for (int i1=0; i1<N-i0-1; i1++)
        if (( p[i1].posada >  p[i1+1].posada )
            ||
            ( p[i1].posada == p[i1+1].posada &&
              p[i1].prizv >  p[i1+1].prizv ))
        {
            tmp = p[i1];
            p[i1] = p[i1+1];
            p[i1+1] = tmp;
        }
}

int BinSearch(Pracivnyk *p, const int N, const string prizv, const Posada posada)
{ // повертає індекс знайденого елемента або -1, якщо шуканий елемент відсутній
    int L=0, R=N-1, m;
    do {
        m = (L+R)/2;
        if ( p[m].prizv == prizv && p[m].posada == posada )
```

```
            return m;
        if (( p[m].posada <  posada )
            ||
            ( p[m].posada == posada &&
              p[m].prizv  <  prizv))
        {
            L = m+1;
        }
        else
        {
            R = m-1;
        }
    } while ( L <= R );
    return -1;
}
```

Індексне впорядкування масиву структур з об’єднаннями

Створити динамічний масив структур з об’єднаннями, які описують наступні дані про працівників:

прізвище – літерний рядок;

рік_народження – беззнакове ціле;

посада – перелік із значень: *робітник*, *інженер*, *службовець*;

оплата – для посади *робітник* – погодинний *тариф*, – дійсне число;

для посад *інженер* та *службовець* – фіксована *ставка* – ціле число.

Реалізувати індексне впорядкування вказаного масиву за наступними ключами: в першу чергу – за посадою (в порядку: спочатку – робітники, потім – інженери, в кінці – службовці); в другу чергу (для однакових посад) – за прізвищем за зростанням – в алфавітному порядку.

Індексне впорядкування означає логічну, а не фактичну (реальну) зміну порядку слідування елементів.

Продовжимо попередню програму, добавимо новий пункт меню та нові функції `IndexSort()`, яка будує індексний масив для реалізації логічного впорядкування, та `PrintIndexSorted()`, яка виводить елементи масиву структур з об’єднаннями в порядку, визначеним індексним масивом:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <Windows.h>            // забезпечення відображення кирилиці
using namespace std;

    enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };
    string posadaStr[] = { "робітник", "інженер", "службовець" };

    struct Pracivnyk
    {
        string   prizv;
        unsigned rik_nar;
```

```
        Posada   posada;
        union
        {
            double taryf;
            int    stavka;
        };
    };

void Create(Pracivnyk *p, const int N);
void Print(Pracivnyk *p, const int N);
double LineSearch(Pracivnyk *p, const int N);
void Sort(Pracivnyk *p, const int N);
int BinSearch(Pracivnyk *p, const int N, const string prizv, const Posada posada);
int *IndexSort(Pracivnyk *p, const int N);
void PrintIndexSorted(Pracivnyk *p, int *I, const int N);

int main()
{                               // забезпечення відображення кирилиці:
    SetConsoleCP(1251);         // встановлення сторінки win-cp1251 в потік вводу
    SetConsoleOutputCP(1251); // встановлення сторінки win-cp1251 в потік виводу

    int N;
    cout << "Введіть кількість працівників N: "; cin >> N;

    Pracivnyk *p = new Pracivnyk[N];

    double proc;
    int iposada;
    Posada posada;
    string prizv;
    int found;

    int menuItem;
    do {
        cout << endl << endl << endl;
        cout << "Виберіть дію:" << endl << endl;
        cout << "  [1] - введення даних з клавіатури" << endl;
        cout << "  [2] - вивід даних на екран" << endl;
        cout << "  [3] - вивід прізвищ та проценту робітників," << endl;
        cout << "        чий тариф перевищує 3,14" << endl;
        cout << "  [4] - фізичне впорядкування даних" << endl;
        cout << "  [5] - бінарний пошук працівника за посадою та прізвищем" << endl;
        cout << "  [6] - індексне впорядкування та вивід даних"
             << endl << endl;
        cout << "  [0] - вихід та завершення роботи програми" << endl << endl;
        cout << "Введіть значення: "; cin >> menuItem;
        cout << endl << endl << endl;

        switch ( menuItem )
        {
        case 1:
            Create(p, N);
            break;
        case 2:
            Print(p, N);
            break;
        case 3:
            proc = LineSearch(p, N);
            cout << "Процент робітників, чий тариф перевищує 3,14:" << endl;
            cout << proc << "%" << endl;
            break;
        case 4:
```

```
            Sort(p, N);
            break;
        case 5:
            cout << "Введіть ключі пошуку:" << endl;
            cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
            cin >> iposada;
            posada = (Posada)iposada;

            cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
            cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

            cout << "    прізвище: "; getline(cin, prizv);
            cout << endl;
            if ( (found = BinSearch(p, N, prizv, posada)) != -1 )
                cout << "Знайдено працівника в позиції " << found+1 << endl;
            else
                cout << "Шуканого працівника не знайдено" << endl;
            break;
        case 6:
            PrintIndexSorted(p, IndexSort(p, N), N);
            break;
        case 0:
            break;
        default:
            cout << "Ви ввели помилкове значення! "
                    "Слід ввести число - номер вибраного пункту меню" << endl;
        }
    } while ( menuItem != 0 );

    return 0;
}

void Create(Pracivnyk *p, const int N)
{
    int posada;
    for (int i=0; i<N; i++)
    {
        cout << "Працівник № " << i+1 << ":" << endl;

        cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
        cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

        cout << "    прізвище: "; getline(cin, p[i].prizv);
        cout << "    рік нар.: "; cin >> p[i].rik_nar;
        cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
        cin >> posada;
        p[i].posada = (Posada)posada;
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "    тариф  : "; cin >> p[i].taryf;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "    ставка : "; cin >> p[i].stavka;
            break;
        }
        cout << endl;
    }
}

void Print(Pracivnyk *p, const int N)
```

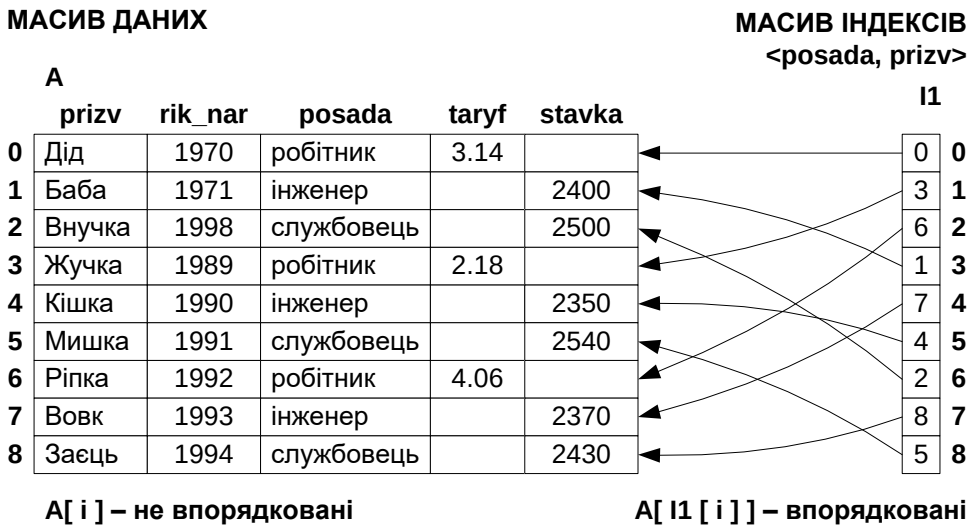
```
{
    cout << "========================================================================="
         << endl;
    cout << "|  №  |   Прізвище   |  Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
         << endl;
    cout << "-------------------------------------------------------------------------"
         << endl;
    for (int i=0; i<N; i++)
    {
        cout << "| "    << setw(3)  << right << i+1 << " ";
        cout << "| "    << setw(13) << left <<  p[i].prizv
             << "|    " << setw(4)  << right << p[i].rik_nar << "    "
             << "| "    << setw(11) << left <<  posadaStr[ p[i].posada ];
        switch ( p[i].posada )
        {
        case ROBITNYK:
            cout << "| " << setw(9) << setprecision(2) << fixed << right
                 << p[i].taryf << " |" << setw(13) << right << "|" << endl;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                 << p[i].stavka << "  |" << endl;
           break;
        }
    }
    cout << "========================================================================="
         << endl;
    cout << endl;
}

double LineSearch(Pracivnyk *p, const int N)
{
    cout << "Прізвища робітників, чий тариф перевищує 3,14:" << endl;
    int k=0, n=0;
    for (int i=0; i<N; i++)
    {
        if ( p[i].posada == ROBITNYK )
        {
            n++;
            if ( p[i].taryf > 3.14 )
            {
                k++;
                cout << setw(3) << right << k
                     << " " << p[i].prizv << endl;
            }
        }
    }
    return 100.0*k/n;
}

void Sort(Pracivnyk *p, const int N)
{
    Pracivnyk tmp;
    for (int i0=0; i0<N-1; i0++)        // метод "бульбашки"
    for (int i1=0; i1<N-i0-1; i1++)
        if (( p[i1].posada >  p[i1+1].posada )
            ||
            ( p[i1].posada == p[i1+1].posada &&
              p[i1].prizv >  p[i1+1].prizv ))
        {
            tmp = p[i1];
```

```
            p[i1] = p[i1+1];
            p[i1+1] = tmp;
        }
}

int BinSearch(Pracivnyk *p, const int N, const string prizv, const Posada posada)
{ // повертає індекс знайденого елемента або -1, якщо шуканий елемент відсутній
    int L=0, R=N-1, m;
    do {
        m = (L+R)/2;
        if ( p[m].prizv == prizv && p[m].posada == posada )
            return m;
        if (( p[m].posada <  posada )
            ||
            ( p[m].posada == posada &&
              p[m].prizv  <  prizv))
        {
            L = m+1;
        }
        else
        {
            R = m-1;
        }
    } while ( L <= R );
    return -1;
}

int *IndexSort(Pracivnyk *p, const int N)
{ //  використовуємо метод вставки для формування індексного масиву
  //
  //   int i, j, value;
  //   for (i = 1; i < length; i++) {
  //       value = a[i];
  //       for (j = i - 1; j >= 0 && a[j] > value; j--) {
  //           a[j + 1] = a[j];
  //       }
  //       a[j + 1] = value;
  //   }

    int *I = new int[N];    // створили індексний масив

    for (int i=0; i<N; i++)
        I[i]=i;             // заповнили його початковими даними

    int i, j, value;        // починаємо сортувати масив індексів
    for (i = 1; i < N; i++)
    {
        value = I[i];
        for (j = i - 1;
             j >= 0 && (( p[I[j]].posada >  p[value].posada ) ||
                        ( p[I[j]].posada == p[value].posada &&
                          p[I[j]].prizv  >  p[value].prizv)) ;
             j--)
        {
            I[j + 1] = I[j];
        }
        I[j + 1] = value;
    }

    return I;
}
```

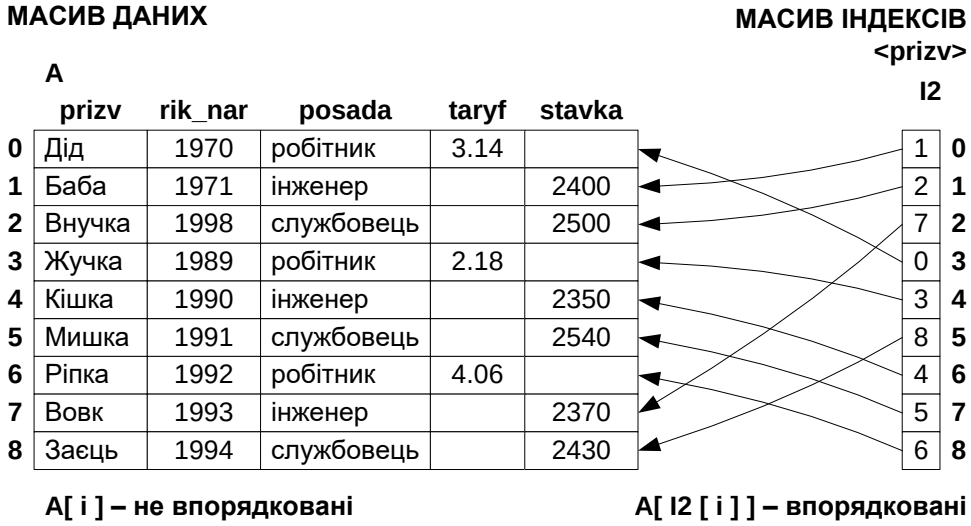
```
void PrintIndexSorted(Pracivnyk *p, int *I, const int N)
{ //  аналогічно функції Print(), але замість звертання p[i]...
  //  використовуємо доступ за допомогою індексного масиву I: p[I[i]]...

    cout << "========================================================================="
         << endl;
    cout << "|  №  |   Прізвище   | Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
         << endl;
    cout << "-------------------------------------------------------------------------"
         << endl;
    for (int i=0; i<N; i++)
    {
        cout << "| "    << setw(3)  << right << i+1 << " ";
        cout << "| "    << setw(13) << left <<  p[I[i]].prizv
             << "|    " << setw(4)  << right << p[I[i]].rik_nar << "    "
             << "| "    << setw(11) << left <<  posadaStr[ p[I[i]].posada ];
        switch ( p[I[i]].posada )
        {
        case ROBITNYK:
            cout << "| " << setw(9) << setprecision(2) << fixed << right
                 << p[I[i]].taryf << " |"
                 << setw(13) << right << "|"
                 << endl;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                 << p[I[i]].stavka << "  |"
                 << endl;
           break;
        }
    }
    cout << "========================================================================="
         << endl;
    cout << endl;
}
```

Пояснення методу індексного впорядкування

Індексне впорядкування не змінює порядку розташування елементів в основному масиві даних A – замість цього будується так званий індексний масив I, який містить значення номерів (індексів) елементів основного масиву даних, записаних таким чином, що елементи індексного масиву відповідають елементам основного масиву даних, впорядкованих за певним критерієм.



Іншими словами, відсутність змін в порядку розташування елементів основного масиву даних означає, що переглядаючи елементи A[i] при зміні індексу i за правилом i=0; i<N; i++ – ніякого порядку не отримаємо; а логічне (індексне) впорядкування означає, що при перегляді елементів основного масиву даних A за допомогою масиву індексів I1 – тобто, A[I1 [i]] при зміні індексу i за правилом i=0; i<N; i++ – будемо вибирати елементи у певному порядку.



Перевагою індексного впорядкування є те, що для одного основного масиву даних можна побудувати багато індексних масивів, які реалізують різні критерії впорядкування. Наприклад, індексний масив I1 реалізує логічне впорядкування за наступними ключами: в першу чергу – за посадою (в порядку: спочатку – робітники, потім – інженери, в кінці – службовці); в другу чергу (для однакових посад) – за прізвищем за зростанням – в алфавітному порядку. Можна побудувати індексний масив I2, який буде реалізувати логічне впорядкування лише за прізвищем – в алфавітному порядку.

Алгоритм побудови індексного масиву реалізований у функції `IndexSort()`, яка отримує вказівник `p` на основний масив даних, кількість елементів в цьому масиві `N` та повертає результат – вказівник на створений та сформований індексний масив.

Тіло функції починається коментарем, який пояснює алгоритм методу вставки (цей метод буде використовуватися при формуванні індексного масиву):

```
//   використовуємо метод вставки для формування індексного масиву
//
//   int i, j, value;
//   for (i = 1; i < length; i++) {
//       value = a[i];
//       for (j = i - 1; j >= 0 && a[j] > value; j--) {
//           a[j + 1] = a[j];
//       }
//       a[j + 1] = value;
//   }
```

– в цьому фрагменті методом вставки впорядковується масив `a`, який складається із `length` елементів цілого типу.

Метод полягає в тому, що в кожний момент часу масив розділений на дві частини: ліву (впорядковану) та праву (ще не впорядковану).

Вважаємо, що елемент `a[0]` вже впорядкований – тобто належить до лівої, впорядкованої частини.

В циклі `for (i = 1; i < length; i++)` будемо впорядковувати наступні елементи – а саме: на кожній ітерації черговий елемент `a[i]` будемо вставляти в потрібну позицію лівої, впорядкованої частини масиву – поки не вичерпаємо всі елементи правої, невпорядкованої частини.

Внутрішній цикл `for (j = i - 1; j >= 0 && a[j] > value; j--)` призначений для пошуку позиції вставки елементу `a[i]` у відсортовану частину масиву.

Перед початком внутрішнього циклу значення елемента `a[i]` зберігаємо у тимчасовій змінній `value`. В тілі циклу переглядаємо елементи лівої впорядкованої частини «справа наліво» (`j = i - 1; j >= 0`), «зсуваємо» елементи «праворуч» командою `a[j + 1] = a[j];`, після завершення циклу записуємо значення `value` в позицію вставки: `a[j + 1] = value;`.

Заголовок цього циклу містить умову `a[j] > value`, яка описує порушення необхідного порядку.

Код функції починається командами створення індексного масиву та його заповнення початковими значеннями:

```
int *I = new int[N];     // створили індексний масив

for (int i=0; i<N; i++)
    I[i]=i;              // заповнили його початковими даними
```

– в якості початкових значень елементів виступають їх індекси, – це відповідає такому тлумаченню: при перегляді масиву `p` за допомогою звертання `p[ I [ i ] ]` при зміні `int i=0; i<N; i++`, ми отримаємо той самий порядок, що і при перегляді масиву `p` за допомогою звертання `p[ i ]`.

Далі (в наступному циклі) починаємо впорядковувати індексний масив `I`, – замість впорядкування масиву `a`.

Умова порушення порядку `a[j] > value` тепер стане такою:

```
(( p[I[j]].posada >  p[value].posada ) ||
 ( p[I[j]].posada == p[value].posada &&
   p[I[j]].prizv  >  p[value].prizv))
```

Для демонстрації результатів індексного впорядкування призначена функція `PrintIndexSorted()`, яка працює аналогічно функції `Print()`, приймає додатковий параметр `I` – вказівник на індексний масив та відрізняється від функції `Print()` лише тим, що в тілі функції `PrintIndexSorted()` використовується звертання до елементів основного масиву даних за допомогою індексного масиву: `p[I[i]]...`, а в тілі функції `Print()` – «безпосереднє» звертання `p[i]...`.

Запис у файл та зчитування із файлу масиву структур

Створити динамічний масив структур з об'єднаннями, які описують наступні дані про працівників:

прізвище – літерний рядок;

рік_народження – беззнакове ціле;

посада – перелік із значень: *робітник*, *інженер*, *службовець*;

оплата – для посади *робітник* – погодинний *тариф*, – дійсне число;

для посад *інженер* та *службовець* – фіксована *ставка* – ціле число.

Реалізувати запис у файл та зчитування із файлу вказаного масиву.

Продовжимо попередню програму, добавимо нові пункти меню та нові функції `SaveToFile()`, яка записує масив структур у файл, та `LoadFromFile()`, яка зчитує елементи масиву структур із файлу:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <Windows.h>            // забезпечення відображення кирилиці
using namespace std;

    enum Posada { ROBITNYK, INZHENER, SLUZHBOVETS };
    string posadaStr[] = { "робітник", "інженер", "службовець" };
```

```
    struct Pracivnyk
    {
        string   prizv;
        unsigned rik_nar;
        Posada   posada;
        union
        {
            double taryf;
            int    stavka;
        };
    };

void Create(Pracivnyk *p, const int N);
void Print(Pracivnyk *p, const int N);
double LineSearch(Pracivnyk *p, const int N);
void Sort(Pracivnyk *p, const int N);
int BinSearch(Pracivnyk *p, const int N, const string prizv, const Posada posada);
int *IndexSort(Pracivnyk *p, const int N);
void PrintIndexSorted(Pracivnyk *p, int *I, const int N);
void SaveToFile(Pracivnyk *p, const int N, const char *filename);
void LoadFromFile(Pracivnyk *&p, int &N, const char *filename);

int main()
{                              // забезпечення відображення кирилиці:
    SetConsoleCP(1251);        // встановлення сторінки win-cp1251 в потік вводу
    SetConsoleOutputCP(1251); // встановлення сторінки win-cp1251 в потік виводу

    int N;
    cout << "Введіть кількість працівників N: "; cin >> N;

    Pracivnyk *p = new Pracivnyk[N];

    double proc;
    int iposada;
    Posada posada;
    string prizv;
    int found;
    char filename[100];

    int menuItem;
    do {
        cout << endl << endl << endl;
        cout << "Виберіть дію:" << endl << endl;
        cout << "  [1] - введення даних з клавіатури" << endl;
        cout << "  [2] - вивід даних на екран" << endl;
        cout << "  [3] - вивід прізвищ та проценту робітників," << endl;
        cout << "        чий тариф перевищує 3,14" << endl;
        cout << "  [4] - фізичне впорядкування даних" << endl;
        cout << "  [5] - бінарний пошук працівника за посадою та прізвищем" << endl;
        cout << "  [6] - індексне впорядкування та вивід даних" << endl;
        cout << "  [7] - запис даних у файл" << endl;
        cout << "  [8] - зчитування даних із файлу" << endl << endl;
        cout << "  [0] - вихід та завершення роботи програми" << endl << endl;
        cout << "Введіть значення: "; cin >> menuItem;
        cout << endl << endl << endl;

        switch ( menuItem )
        {
        case 1:
            Create(p, N);
            break;
```

```
        case 2:
            Print(p, N);
            break;
        case 3:
            proc = LineSearch(p, N);
            cout << "Процент робітників, чий тариф перевищує 3,14:" << endl;
            cout << proc << "%" << endl;
            break;
        case 4:
            Sort(p, N);
            break;
        case 5:
            cout << "Введіть ключі пошуку:" << endl;
            cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
            cin >> iposada;
            posada = (Posada)iposada;

            cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
            cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

            cout << "    прізвище: "; getline(cin, prizv);
            cout << endl;
            if ( (found = BinSearch(p, N, prizv, posada)) != -1 )
                cout << "Знайдено працівника в позиції " << found+1 << endl;
            else
                cout << "Шуканого працівника не знайдено" << endl;
            break;
        case 6:
            PrintIndexSorted(p, IndexSort(p, N), N);
            break;
        case 7:
            cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
            cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

            cout << "Введіть ім'я файлу: "; cin.getline(filename, 99);
            SaveToFile(p, N, filename);
            break;
        case 8:
            cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
            cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

            cout << "Введіть ім'я файлу: "; cin.getline(filename, 99);
            LoadFromFile(p, N, filename);
            break;
        case 0:
            break;
        default:
            cout << "Ви ввели помилкове значення! "
                    "Слід ввести число - номер вибраного пункту меню" << endl;
        }
    } while ( menuItem != 0 );

    return 0;
}

void Create(Pracivnyk *p, const int N)
{
    int posada;
    for (int i=0; i<N; i++)
    {
        cout << "Працівник № " << i+1 << ":" << endl;
```

```
            cin.get();  // очищуємо буфер клавіатури – бо залишаються символи
            cin.sync(); // "кінець рядка", які не дають ввести наступний літерний рядок

            cout << "    прізвище: "; getline(cin, p[i].prizv);
            cout << "    рік нар.: "; cin >> p[i].rik_nar;
            cout << "    посада (0 - робітник, 1 - інженер, 2 - службовець): ";
            cin >> posada;
            p[i].posada = (Posada)posada;
            switch ( p[i].posada )
            {
            case ROBITNYK:
                cout << "    тариф  : "; cin >> p[i].taryf;
                break;
            case INZHENER:
            case SLUZHBOVETS:
                cout << "    ставка : "; cin >> p[i].stavka;
                break;
            }
            cout << endl;
        }
    }

    void Print(Pracivnyk *p, const int N)
    {
        cout << "=============================================================================="
             << endl;
        cout << "|  №  |   Прізвище   |  Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
             << endl;
        cout << "------------------------------------------------------------------------------"
             << endl;

        for (int i=0; i<N; i++)
        {
            cout << "| "    << setw(3)  << right << i+1 << " ";
            cout << "| "    << setw(13) << left <<  p[i].prizv
                 << "|    " << setw(4)  << right << p[i].rik_nar << "    "
                 << "| "    << setw(11) << left <<  posadaStr[ p[i].posada ];
            switch ( p[i].posada )
            {
            case ROBITNYK:
                cout << "| " << setw(9) << setprecision(2) << fixed << right
                     << p[i].taryf << " |" << setw(13) << right << "|" << endl;
                break;
            case INZHENER:
            case SLUZHBOVETS:
                cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                     << p[i].stavka << "  |" << endl;
               break;
            }
        }
        cout << "=============================================================================="
             << endl;
        cout << endl;
    }

    double LineSearch(Pracivnyk *p, const int N)
    {
        cout << "Прізвища робітників, чий тариф перевищує 3,14:" << endl;
        int k=0, n=0;
        for (int i=0; i<N; i++)
        {
            if ( p[i].posada == ROBITNYK )
```

```
        {
            n++;
            if ( p[i].taryf > 3.14 )
            {
                k++;
                cout << setw(3) << right << k
                     << " " << p[i].prizv << endl;
            }
        }
    }
    return 100.0*k/n;
}

void Sort(Pracivnyk *p, const int N)
{
    Pracivnyk tmp;
    for (int i0=0; i0<N-1; i0++)        // метод "бульбашки"
    for (int i1=0; i1<N-i0-1; i1++)
        if (( p[i1].posada >  p[i1+1].posada )
            ||
            ( p[i1].posada == p[i1+1].posada &&
              p[i1].prizv >  p[i1+1].prizv ))
        {
            tmp = p[i1];
            p[i1] = p[i1+1];
            p[i1+1] = tmp;
        }
}

int BinSearch(Pracivnyk *p, const int N, const string prizv, const Posada posada)
{ // повертає індекс знайденого елемента або -1, якщо шуканий елемент відсутній
    int L=0, R=N-1, m;
    do {
        m = (L+R)/2;
        if ( p[m].prizv == prizv && p[m].posada == posada )
            return m;
        if (( p[m].posada <  posada )
            ||
            ( p[m].posada == posada &&
              p[m].prizv  <  prizv))
        {
            L = m+1;
        }
        else
        {
            R = m-1;
        }
    } while ( L <= R );
    return -1;
}

int *IndexSort(Pracivnyk *p, const int N)
{ //   використовуємо метод вставки для формування індексного масиву
  //
  //   int i, j, value;
  //   for (i = 1; i < length; i++) {
  //       value = a[i];
  //       for (j = i - 1; j >= 0 && a[j] > value; j--) {
  //           a[j + 1] = a[j];
  //       }
  //       a[j + 1] = value;
  //   }
```

```
    int *I = new int[N];    // створили індексний масив

    for (int i=0; i<N; i++)
        I[i]=i;             // заповнили його початковими даними

    int i, j, value;        // починаємо сортувати масив індексів
    for (i = 1; i < N; i++)
    {
        value = I[i];
        for (j = i - 1;
             j >= 0 && (( p[I[j]].posada >  p[value].posada ) ||
                        ( p[I[j]].posada == p[value].posada &&
                          p[I[j]].prizv  >  p[value].prizv)) ;
             j--)
        {
            I[j + 1] = I[j];
        }
        I[j + 1] = value;
    }

    return I;
}

void PrintIndexSorted(Pracivnyk *p, int *I, const int N)
{ //  аналогічно функції Print(), але замість звертання p[i]...
  //  використовуємо доступ за допомогою індексного масиву I: p[I[i]]...

    cout << "=========================================================================="
         << endl;
    cout << "|  №  |   Прізвище   |  Рік.нар.  |   Посада   |   Тариф   |   Ставка   |"
         << endl;
    cout << "--------------------------------------------------------------------------"
         << endl;
    for (int i=0; i<N; i++)
    {
        cout << "| "    << setw(3)  << right << i+1 << " ";
        cout << "| "    << setw(13) << left <<  p[I[i]].prizv
             << "|    " << setw(4)  << right << p[I[i]].rik_nar << "    "
             << "| "    << setw(11) << left <<  posadaStr[ p[I[i]].posada ];
        switch ( p[I[i]].posada )
        {
        case ROBITNYK:
            cout << "| " << setw(9) << setprecision(2) << fixed << right
                 << p[I[i]].taryf << " |"
                 << setw(13) << right << "|"
                 << endl;
            break;
        case INZHENER:
        case SLUZHBOVETS:
            cout << "| " << setw(11) << right << "|" << "  " << setw(8) << right
                 << p[I[i]].stavka << "  |"
                 << endl;
           break;
        }
    }
    cout << "=========================================================================="
         << endl;
    cout << endl;
}

void SaveToFile(Pracivnyk *p, const int N, const char *filename)
```

```
{
    ofstream fout(filename, ios::binary);            // відкрили бінарний файл запису

    fout.write((char*)&N, sizeof(N));                // записали кількість елементів

    for (int i=0; i<N; i++)
        fout.write((char*)&p[i], sizeof(Pracivnyk)); // записали елементи масиву

    fout.close();                                    // закрили файл
}

void LoadFromFile(Pracivnyk *&p, int &N, const char *filename)
{
    delete [] p;                                    // знищили попередні дані

    ifstream fin(filename, ios::binary);            // відкрили бінарний файл зчитування

    fin.read((char*)&N, sizeof(N));                 // прочитали кількість елементів

    p = new Pracivnyk[N];                           // створили динамічний масив

    for (int i=0; i<N; i++)
        fin.read((char*)&p[i], sizeof(Pracivnyk)); // прочитали елементи масиву

    fin.close();                                    // закрили файл
}
```

Пояснення функцій файлового вводу / виводу

Розглядаємо лише засоби файлового потокового вводу / виводу (тобто, засоби мови C++). Засоби, які надає мова C стосовно опрацювання файлів – в цьому посібнику розглядатися не будуть (як і в попередніх темах, коли розглядалися лише засоби мови C++ організації консольного вводу / виводу).

Для доступу до необхідних ресурсів потрібно підключити бібліотеку `<fstream>`, яка описує засоби файлового потокового вводу / виводу. В цій бібліотеці описано (серед багатьох інших ресурсів) два файлових потокових типи: `ofstream` (output file stream – англ.: потік файлового виводу) та `ifstream` (input file stream – англ.: потік файлового вводу). Терміну «файловий вивід» відповідає «запис у файл», а «файловий ввід» – «зчитування з файлу».

При оголошенні файлової змінної (тобто, змінної, яка належить до типу «потік файлового вводу» чи «потік файлового виводу») слід задати по-перше: ім'я файлу в нотації операційної системи – нуль-термінальний літерний рядок `char []`, та по-друге: признак бінарного файлу `ios::binary` – вказує, що дані у файл будуть записуватися як двійкові коди, а не як текст, і так само будуть зчитуватися з файлу у вигляді двійкових кодів, а не тексту (тобто, не так, як організовано консольний ввід/вивід).

Запис у файл

Розглянемо алгоритм функції `SaveToFile()`, яка реалізує запис даних у файл.

Функція приймає параметри: `p` – вказівник на динамічний масив структур з об'єднаннями, який містить інформацію про працівників; `N` – кількість елементів у цьому масиві; `filename` – вказівник на нуль-термінальний літерний рядок, який містить ім'я файлу.

Перша команда в тілі функції

```
ofstream fout(filename, ios::binary);
```

«відкриває» файл `filename` для запису (output file stream – потік файлового виводу) та створює файлову змінну `fout` для доступу до компонентів цього файлу. Вибір імені змінної – справа смаку програміста; ім'я `fout` (file output – файловий вивід) вибрано за аналогією до `cout` (console output – консольний вивід).

Наступна команда

```
fout.write((char*)&N, sizeof(N));
```

записує у файл `fout` (тобто, у файл, пов'язаний із файловою змінною `fout`), значення змінної `N`. Параметри функції `write()`: адреса початку області пам'яті, звідки слід брати дані для запису у файл, та розмір цієї області пам'яті.

Оскільки потік – це набір символів, які передаються від одного (довільного) пристрою до іншого (теж довільного) пристрою, то адресу змінної `&N` слід перетворити до типу «вказівник на символ»: `(char*)&N` – перший параметр функції `write()`. Для обчислення розміру області пам'яті, відведеної для змінної `N`, традиційно використовуємо операцію `sizeof(N)`.

Далі в циклі записуємо у файл значення елементів масиву структур:

```
for (int i=0; i<N; i++)           // записали елементи масиву
    fout.write((char*)&p[i], sizeof(Pracivnyk));
```

– адресу чергового елемента масиву `&p[i]` перетворюємо до типу «вказівник на символ»: `(char*)&p[i]` – перший параметр та вказуємо розмір структури «працівник»: `sizeof(Pracivnyk)` – другий параметр функції `write()`.

Остання команда

```
fout.close();
```

«закриває» файл `fout`, тобто завершує всі дії з файлом і гарантує, що всі дані, які записувалися у файл за допомогою файлової змінної `fout` будуть збережені на диску у файлі операційної системи з іменем, вказаним в змінній `filename`.

Для розуміння: `filename` – нуль-термінальний літерний рядок, що містить ім'я «фізичного» файлу, тобто – файлу операційної системи. Файлова змінна `fout` – це «програмний» або «логічний» файл, тобто – представник «фізичного» файлу в програмі.

Зчитування з файлу

Розглянемо алгоритм функції `LoadFromFile()`, яка реалізує зчитування даних із файлу.

Функція приймає параметри: `&p` – вказівник на динамічний масив структур з об'єднаннями, який містить інформацію про працівників; `&N` – кількість елементів у цьому масиві; `filename` – вказівник на нуль-термінальний літерний рядок, який містить ім'я файлу.

Оскільки в результаті зчитування можуть змінитися кількість елементів (прочитали іншу кількість, ніж була до того) та буде змінено вказівник на масив – то параметри `&p` та `&N` приймаються як посилання, тим самим забезпечується можливість передачі інформації за допомогою цих параметрів як у функцію так і з функції.

Перша команда в тілі функції

```
delete [] p;
```

знищує попередні дані – звільняє пам'ять, виділену для динамічного масиву структур з об'єднаннями, що містить інформацію про працівників.

Команда

```
ifstream fin(filename, ios::binary);
```

«відкриває» файл `filename` для зчитування (input file stream – потік файлового вводу) та створює файлову змінну `fin` для доступу до компонентів цього файлу. Вибір імені змінної – справа смаку програміста; ім'я `fin` (file input – файловий ввід) вибрано за аналогією до `cin` (console input – консольний ввід).

Наступна команда

```
fin.read((char*)&N, sizeof(N));
```

зчитує з файлу `fin` (тобто, з файлу, пов'язаного з файловою змінною `fin`), значення змінної `N`. Параметри функції `read()`: адреса початку області пам'яті, куди слід записувати прочитані з файлу дані, та розмір цієї області пам'яті.

Оскільки потік – це набір символів, які передаються від одного (довільного) пристрою до іншого (теж довільного) пристрою, то адресу змінної `&N` слід перетворити до типу «вказівник на символ»: `(char*)&N` – перший параметр функції `read()`. Для обчислення розміру області пам'яті, відведеної для змінної `N`, традиційно використовуємо операцію `sizeof(N)`.

Потім командою

```
p = new Pracivnyk[N];
```

створюємо новий динамічний масив і в циклі

```
for (int i=0; i<N; i++)          // прочитали елементи масиву
    fin.read((char*)&p[i], sizeof(Pracivnyk));
```

зчитуємо значення його елементів: адресу чергового елемента масиву `&p[i]` перетворюємо до типу «вказівник на символ»: `(char*)&p[i]` – перший параметр, та вказуємо розмір структури «працівник»: `sizeof(Pracivnyk)` – другий параметр функції `read()`.

Остання команда

```
fin.close();
```

«закриває» файл `fin`, тобто завершує всі дії з відповідним файлом операційної системи з іменем, вказаним в змінній `filename`.

Семантика імен – аналогічна до тої, яка використовувалася в попередньому параграфі: `filename` – нуль-термінальний літерний рядок, що містить ім'я «фізичного» файлу, тобто – файлу операційної системи. Файлова змінна `fin` – це «програмний» або «логічний» файл, тобто – представник «фізичного» файлу в програмі.

Вилучення елементів масиву

Лабораторна робота № 9.3 вимагає скласти програму, яка в тому числі буде містити функцію, що вилучає певні елементи масиву. Принципи створення такої функції розглядаються в цьому параграфі.

Напишемо програму, яка формує за допомогою генератора випадкових чисел масив цілих чисел із `N` елементів з діапазону `[-10; 10]`, виводить його на екран, вилучає всі від'ємні елементи і виводить на екран модифікований масив.

Функції, що реалізують формування та вивід масиву на екран, – розглядалися раніше.

Функція `Remove()`, яка реалізує вилучення елементів масиву, пояснена в коментарях:

```
#include <iostream>
#include <iomanip>
#include <time.h>
using namespace std;

void Create(int* a, const int N);
void Print(int* a, const int N);
void Remove(int* &a, int &N);

int main(int argc, char* argv[])
{
    srand((unsigned) time(NULL));

    int N;
    cout << "Enter size: "; cin >> N;

    int *a = new int[N];

    Create(a, N);
    Print(a, N);

    Remove(a, N);
    Print(a, N);
```

```
    return 0;
}

void Create(int* a, const int N)
{
    for (int i=0; i<N; i++)
        a[i] = -10 + rand()%21;
}

void Print(int* a, const int N)
{
    for (int i=0; i<N; i++)
        cout << "a[" << setw(2) << i << "] = " << a[i] << endl;
    cout << endl;
}

void Remove(int* &a, int &N)
{
    int K=0;                // кількість елементів, які потрібно залишити
    for (int i=0; i<N; i++) // скануємо заданий масив і
        if (a[i] >= 0)      // обчислюємо кількість елементів,
            K++;            // які потрібно залишити

    int* t = new int[K];    // тимчасовий масив

    int j=0;                // індекс в тимчасовому масиві
    for (int i=0; i<N; i++) // скануємо заданий масив
        if (a[i] >= 0)      // і копіюємо елементи,
            t[j++] = a[i];  // які потрібно залишити

    delete [] a;            // знищуємо заданий масив
    a = t;                  // налаштовуємо вказівник на тимчасовий масив
    N = K;                  // змінюємо значення кількості елементів
}
```

Лабораторний практикум

Оформлення звітів про виконання лабораторних робіт

Вимоги до оформлення звіту про виконання лабораторних робіт №№ 9.1–9.3

Звіти про виконання лабораторних робіт №№ 9.1–9.3 мають містити наступні елементи:

1) заголовок;
2) умову завдання;

   Умова завдання має бути вставлена у звіт як фрагмент зображення (скрін) сторінки посібника.
3) структурну схему програми;

   Структурна схема програми зображує взаємозв'язки програми та всіх її програмних одиниць: схему вкладеності та охоплення підпрограм, програми та модулів; а також схему звертання одних програмних одиниць до інших.
4) текст програми;

   Текст програми має бути правильно відформатований: відступами і порожніми рядками слід відображати логічну структуру програми; програма має містити необхідні коментарі – про призначення підпрограм, змінних та параметрів – якщо їх імена не значущі, та про призначення окремих змістовних фрагментів програми. Текст програми слід подавати моноширинним шрифтом (`Courier New` *розміром 10 пт. або* `Consolas` *розміром 9,5 пт.*) *з одинарним міжрядковим інтервалом;*
5) посилання на git-репозиторій з проектом;
6) хоча б для одної функції, яка повертає результат (як результат функції чи як параметр-посилання) – результати unit-тесту: текст програми unit-тесту та скрін результатів її виконання (див. хід виконання Лабораторної роботи № 5.6);
7) висновки.

Зразок оформлення звіту про виконання лабораторних робіт №№ 9.1–9.3

ЗВІТ

про виконання лабораторної роботи № < *номер* >

« *назва теми лабораторної роботи* »

з дисципліни

«Алгоритмізація та програмування»

студента(ки) групи КН-16

< Прізвище Ім'я По_батькові >

Умова завдання:

…

Структурна схема програми:

…

Текст програми:

…

Посилання на git-репозиторій з проектом:

…

Результати unit-тесту:

…

Висновки:

…

Лабораторна робота № 9.1. Послідовний пошук в масиві структур

Мета роботи

Навчитися опрацьовувати масиви структур з об’єднаннями.

Питання, які необхідно вивчити та пояснити на захисті

1) Загальний синтаксис оголошення масивів та переліків, структур і об’єднань.
2) Доступ до елементів масивів та переліків, структур і об’єднань.
3) Дії з масивами та переліками, структурами і об’єднаннями.
4) Ініціалізація масивів та переліків, структур і об’єднань.
5) Внутрішня реалізація масивів та переліків, структур і об’єднань.
6) Передавання та опрацювання масивів та структур у функціях.
7) Загальний синтаксис оголошення та формування динамічних масивів.
8) Загальна схема послідовного пошуку в масиві.

Оформлення звіту

Вимоги та зразок оформлення звіту наведені у вступі до лабораторного практикуму.

Приклади розв’язання лабораторних завдань

Взірці виконання лабораторних завдань – див. в розділі Приклади: Оголошення структури з об’єднаннями, Формування динамічного масиву структур з об’єднаннями, Вивід масиву структур з об’єднаннями у вигляді таблиці, Послідовний пошук в масиві структур з об’єднаннями.

Варіанти лабораторних завдань

Рівень А). Сформувати масив структур, що містять інформацію про: прізвище студента, курс, спеціальність (для представлення спеціальності використовувати переліки, а для представлення курсу – цілі числа) та оцінки з фізики, математики, інформатики.

Рівень В). Сформувати масив структур з об’єднаннями, що містять інформацію про: прізвище студента, курс, спеціальність (для представлення спеціальності використовувати переліки, а для представлення курсу – цілі числа) та оцінки з фізики, математики; якщо спеціальність – «Комп’ютерні науки», то третя оцінка – з програмування; якщо спеціальність – «Інформатика», то третя оцінка – з чисельних методів; для всіх інших

спеціальностей: «Математика та економіка», «Фізика та інформатика», «Трудове навчання» – третя оцінка – з педагогіки. Для представлення третьої оцінки використовувати об’єднання.

Використовувати динамічний масив структур – кількість студентів вводить користувач при виконанні програми.

Сформований масив вивести на екран у вигляді таблиці, кожний рядок якої містить:

Рівень А) – сім клітинок: (1) – порядковий номер студента у групі, (2) – прізвище, (3) – курс, (4) – спеціальність, оцінки з (5) – фізики, (6) – математики, (7) – інформатики;

Рівень В) – дев’ять клітинок: (1) – порядковий номер студента у групі, (2) – прізвище, (3) – курс, (4) – спеціальність, оцінки з (5) – фізики, (6) – математики (клітинки цих оцінок заповнені для студентів всіх спеціальностей), оцінки з (7) – програмування, (8) – чисельних методів, (9) – педагогіки (заповнена лише одна з клітинок цих оцінок – залежно від спеціальності такого студента).

Таблиця має містити заголовок і шапку.

Всі дії – формування, опрацювання масиву даних та вивід результатів – слід реалізувати окремими функціями, які всю необхідну інформацію отримують за допомогою параметрів. Використання глобальних змінних – не допускається.

Кожна функція має виконувати лише одну роль, і ця роль має бути відображена у назві функції.

«Функція, яка повертає / обчислює / шукає ...» – має не виводити ці значення, а повернути їх у місце виклику як результат функції або як відповідний вихідний параметр.

Для зарахування лабораторної роботи достатньо виконати одне з двох завдань, кожне з яких оцінюється у 50% від максимальної кількості балів.

Варіант 1.

1. Вивести прізвища студентів, які вчаться на “відмінно”.
2. Обчислити процент студентів, у яких середній бал більший за 4,5.

Варіант 2.

1. Обчислити кількість студентів, які вчаться без трійок (на “відмінно” і “добре”).
2. Обчислити процент студентів, у яких середній бал менший 4.

Варіант 3.

1. Вивести прізвища студентів, які вчаться без трійок (на “відмінно” і “добре”).

2. Обчислити кількість студентів, які отримали з фізики оцінку “5”.

Варіант 4.

1. Обчислити кількість студентів, які вчаться на “відмінно”.
2. Обчислити процент студентів, які отримали з фізики оцінку “5”.

Варіант 5.

1. Обчислити процент студентів, які вчаться на “відмінно”.
2. Вивести прізвища студентів, які отримали з фізики оцінку “5”.

Варіант 6.

1. Обчислити процент студентів, які вчаться без трійок (на “відмінно” і “добре”).
2. Вивести прізвища студентів, які отримали з фізики оцінки “5” або “4”.

Варіант 7.

1. Для кожного студента вивести: прізвище і середній бал.
2. Обчислити процент студентів, які отримали з фізики оцінки “5” або “4”.

Варіант 8.

1. Для кожного предмету обчислити середній бал.
2. Обчислити кількість студентів, які отримали з фізики оцінки “5” або “4”.

Варіант 9.

1. Обчислити кількість оцінок «відмінно» з кожного предмету.
2. Обчислити кількість студентів, які отримали з фізики і математики оцінки «5».

Варіант 10.

1. Обчислити кількість оцінок «добре» з кожного предмету.
2. Обчислити процент студентів, які отримали і з фізики і з математики оцінку «5».

Варіант 11.

1. Обчислити кількість оцінок «задовільно» з кожного предмету.
2. Вивести прізвища студентів, які отримали і з фізики і з математики оцінки «5» або «4».

Варіант 12.

1. Обчислити кількість кожної з оцінок «5», «4», «3» з фізики.

2. Обчислити кількість студентів, які отримали і з фізики і з математики оцінки «4» або «5».

Варіант 13.

1. Обчислити кількість кожної з оцінок «5», «4», «3» з математики.
2. Вивести прізвища студентів, які отримали і з фізики і з математики оцінку «5».

Варіант 14.

1. Обчислити кількість кожної з оцінок «5», «4», «3» з програмування.
2. Обчислити процент студентів, які отримали і з фізики і з математики оцінки «4» або «5».

Варіант 15.

1. Обчислити найбільший середній бал (порівнюючи середні бали для кожного студента).
2. Обчислити процент студентів, які отримали з фізики оцінки “5” або “4”.

Варіант 16.

1. Обчислити кількість студентів, середній бал яких вищий за 4,5.
2. Порівнюючи середні бали для кожного предмету, визначити предмет, середній бал якого найбільший.

Варіант 17.

1. Обчислити кількість студентів, середній бал яких менший 4.
2. Порівнюючи середні бали для кожного предмету, визначити предмет, середній бал якого найменший.

Варіант 18.

1. Обчислити найменший середній бал (порівнюючи середні бали для кожного студента).
2. Обчислити кількість оцінок «добре» з кожного предмету.

Варіант 19.

1. Вивести прізвище студента, у якого найбільший середній бал.
2. Обчислити процент студентів, які вчаться на «відмінно».

Варіант 20.

1. Вивести прізвище студента, у якого найменший середній бал.
2. Обчислити кількість оцінок «задовільно» з кожного предмету.

Варіант 21.

1. Вивести прізвища студентів, які вчаться на “відмінно”.
2. Обчислити процент студентів, у яких середній бал більший за 4,5.

Варіант 22.

1. Обчислити кількість студентів, які вчаться без трійок (на “відмінно” і “добре”).
2. Обчислити процент студентів, у яких середній бал менший 4.

Варіант 23.

1. Вивести прізвища студентів, які вчаться без трійок (на “відмінно” і “добре”).
2. Обчислити кількість студентів, які отримали з фізики оцінку “5”.

Варіант 24.

1. Обчислити кількість студентів, які вчаться на “відмінно”.
2. Обчислити процент студентів, які отримали з фізики оцінку “5”.

Варіант 25.

1. Обчислити процент студентів, які вчаться на “відмінно”.
2. Вивести прізвища студентів, які отримали з фізики оцінку “5”.

Варіант 26.

1. Обчислити процент студентів, які вчаться без трійок (на “відмінно” і “добре”).
2. Вивести прізвища студентів, які отримали з фізики оцінки “5” або “4”.

Варіант 27.

1. Для кожного студента вивести: прізвище і середній бал.
2. Обчислити процент студентів, які отримали з фізики оцінки “5” або “4”.

Варіант 28.

1. Для кожного предмету обчислити середній бал.
2. Обчислити кількість студентів, які отримали з фізики оцінки “5” або “4”.

Варіант 29.

1. Обчислити кількість оцінок «відмінно» з кожного предмету.
2. Обчислити кількість студентів, які отримали з фізики і математики оцінки «5».

Варіант 30.

1. Обчислити кількість оцінок «добре» з кожного предмету.
2. Обчислити процент студентів, які отримали і з фізики і з математики оцінку «5».

Варіант 31.

1. Обчислити кількість оцінок «задовільно» з кожного предмету.
2. Вивести прізвища студентів, які отримали і з фізики і з математики оцінки «5» або «4».

Варіант 32.

1. Обчислити кількість кожної з оцінок «5», «4», «3» з фізики.
2. Обчислити кількість студентів, які отримали і з фізики і з математики оцінки «4» або «5».

Варіант 33.

1. Обчислити кількість кожної з оцінок «5», «4», «3» з математики.
2. Вивести прізвища студентів, які отримали і з фізики і з математики оцінку «5».

Варіант 34.

1. Обчислити кількість кожної з оцінок «5», «4», «3» з програмування.
2. Обчислити процент студентів, які отримали і з фізики і з математики оцінки «4» або «5».

Варіант 35.

1. Обчислити найбільший середній бал (порівнюючи середні бали для кожного студента).
2. Обчислити процент студентів, які отримали з фізики оцінки “5” або “4”.

Варіант 36.

1. Обчислити кількість студентів, середній бал яких вищий за 4,5.
2. Порівнюючи середні бали для кожного предмету, визначити предмет, середній бал якого найбільший.

Варіант 37.

1. Обчислити кількість студентів, середній бал яких менший 4.
2. Порівнюючи середні бали для кожного предмету, визначити предмет, середній бал якого найменший.

Варіант 38.

1. Обчислити найменший середній бал (порівнюючи середні бали для кожного студента).
2. Обчислити кількість оцінок «добре» з кожного предмету.

Варіант 39.

1. Вивести прізвище студента, у якого найбільший середній бал.
2. Обчислити процент студентів, які вчаться на «відмінно».

Варіант 40.

1. Вивести прізвище студента, у якого найменший середній бал.
2. Обчислити кількість оцінок «задовільно» з кожного предмету.

Лабораторна робота № 9.2. Впорядкування та бінарний пошук в масиві структур

Мета роботи

Навчитися впорядковувати масив структур з об'єднаннями. Навчитися здійснювати фізичне та індексне впорядкування. Навчитися здійснювати бінарний пошук у фізично чи індексно впорядкованому масиві.

Питання, які необхідно вивчити та пояснити на захисті

1) *Загальний синтаксис оголошення масивів та переліків, структур і об'єднань.*
2) *Доступ до елементів масивів та переліків, структур і об'єднань.*
3) *Дії з масивами та переліками, структурами і об'єднаннями.*
4) *Ініціалізація масивів та переліків, структур і об'єднань.*
5) *Внутрішня реалізація масивів та переліків, структур і об'єднань.*
6) *Передавання та опрацювання масивів та структур у функціях.*
7) *Загальний синтаксис оголошення та формування динамічних масивів.*
8) Загальна схема фізичного впорядкування масиву.
9) Загальна схема індексного впорядкування масиву.
10) Загальна схема бінарного пошуку в масиві.

Оформлення звіту

Вимоги та зразок оформлення звіту наведені у вступі до лабораторного практикуму.

Приклади розв'язання лабораторних завдань

Взірці виконання лабораторних завдань – див. в розділі Приклади: Оголошення структури з об'єднаннями, Формування динамічного масиву структур з об'єднаннями, Вивід масиву структур з об'єднаннями у вигляді таблиці, Організація меню в текстовому режимі, Фізичне впорядкування масиву структур з об'єднаннями, Бінарний пошук в масиві структур з об'єднаннями, Індексне впорядкування масиву структур з об'єднаннями.

Варіанти лабораторних завдань

Рівень А). Сформувати масив структур, що містять інформацію про: прізвище студента, курс, спеціальність (для представлення спеціальності використовувати переліки, а для представлення курсу – цілі числа) та оцінки з фізики, математики, інформатики.

Рівень В). Сформувати масив структур з об'єднаннями, що містять інформацію про: прізвище студента, курс, спеціальність (для представлення спеціальності використовувати переліки, а для представлення курсу – цілі числа) та оцінки з фізики, математики; якщо спеціальність – «Комп'ютерні науки», то третя оцінка – з програмування; якщо спеціальність – «Інформатика», то третя оцінка – з чисельних методів; для всіх інших спеціальностей: «Математика та економіка», «Фізика та інформатика», «Трудове навчання» – третя оцінка – з педагогіки. Для представлення третьої оцінки використовувати об'єднання.

Використовувати динамічний масив структур – кількість студентів вводить користувач при виконанні програми.

Сформований масив вивести на екран у вигляді таблиці, кожний рядок якої містить:

Рівень А) – сім клітинок: (1) – порядковий номер студента у групі, (2) – прізвище, (3) – курс, (4) – спеціальність, оцінки з (5) – фізики, (6) – математики, (7) – інформатики;

Рівень В) – дев'ять клітинок: (1) – порядковий номер студента у групі, (2) – прізвище, (3) – курс, (4) – спеціальність, оцінки з (5) – фізики, (6) – математики (клітинки цих оцінок заповнені для студентів всіх спеціальностей), оцінки з (7) – програмування, (8) – чисельних методів, (9) – педагогіки (заповнена лише одна з клітинок цих оцінок – залежно від спеціальності такого студента).

Таблиця має містити заголовок і шапку.

Програма має містити меню. Необхідно передбачити контроль помилок користувача при введенні даних.

Всі дії – формування, опрацювання масиву даних та вивід результатів – слід реалізувати окремими функціями, які всю необхідну інформацію отримують за допомогою параметрів. Використання глобальних змінних – не допускається.

Кожна функція має виконувати лише одну роль, і ця роль має бути відображена у назві функції.

«Функція, яка повертає / обчислює / шукає ...» – має не виводити ці значення, а повернути їх у місце виклику як результат функції або як відповідний вихідний параметр.

Для зарахування лабораторної роботи достатньо виконати одне з трьох завдань, кожне з яких оцінюється незалежно від інших.

Варіант 1.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за

номером курсу, в останню чергу (для однакових спеціальностей і курсів) – за прізвищем за зростанням – в алфавітному порядку.

2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за номером курсу, в останню чергу (для однакових спеціальностей і курсів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності на вказаному курсі.

Варіант 2.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за назвою спеціальності, в останню чергу (для однакових курсів і спеціальностей) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за назвою спеціальності, в останню чергу (для однакових курсів і спеціальностей) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі на вказаній спеціальності.

Варіант 3.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням середнього балу, в останню чергу (для однакових спеціальностей і середніх балів) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням середнього балу, в останню чергу (для однакових спеціальностей і середніх балів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаним середнім балом.

Варіант 4.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням

середнього балу, в останню чергу (для однакових курсів і середніх балів) – за прізвищем за спаданням – в зворотному до алфавітного порядку.

2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням середнього балу, в останню чергу (для однакових курсів і середніх балів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаним середнім балом.

Варіант 5.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням оцінки з математики, в останню чергу (для однакових спеціальностей і оцінок з математики) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням оцінки з математики, в останню чергу (для однакових спеціальностей і оцінок з математики) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаною оцінкою з математики.

Варіант 6.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням оцінки з математики, в останню чергу (для однакових курсів і оцінок з математики) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням оцінки з математики, в останню чергу (для однакових курсів і оцінок з математики) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаною оцінкою з математики.

Варіант 7.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням оцінки з третього - профільного - предмету, в останню чергу (для однакових спеціальностей і оцінок з третього предмету) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням оцінки з третього – профільного – предмету, в останню чергу (для однакових спеціальностей і оцінок з третього предмету) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаною оцінкою з третього предмету.

Варіант 8.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням оцінки з третього – профільного – предмету, в останню чергу (для однакових курсів і оцінок з третього предмету) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням оцінки з третього – профільного – предмету, в останню чергу (для однакових курсів і оцінок з третього предмету) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаною оцінкою з третього предмету.

Варіант 9.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за назвою спеціальності, в останню чергу (для однакових середніх балів і спеціальностей) – за прізвищем за зростанням – в алфавітному порядку.

2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за назвою спеціальності, в останню чергу (для однакових середніх балів і спеціальностей) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаним середнім балом.

Варіант 10.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за номером курсу, в останню чергу (для однакових середніх балів і курсів) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за номером курсу, в останню чергу (для однакових середніх балів і курсів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаним середнім балом.

Варіант 11.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з фізики, в другу чергу (для однакових оцінок з фізики) – за назвою спеціальності, в останню чергу (для однакових оцінок з фізики і спеціальностей) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з фізики, в другу чергу (для однакових оцінок з фізики) – за назвою спеціальності, в останню чергу (для однакових оцінок з фізики і спеціальностей) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаною оцінкою з фізики.

Варіант 12.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з фізики, в другу чергу (для однакових оцінок з фізики) – за номером курсу, в останню чергу (для однакових оцінок з фізики і курсів) – за прізвищем за спаданням – в зворотному до алфавітного порядку.

2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з фізики, в другу чергу (для однакових оцінок з фізики) – за номером курсу, в останню чергу (для однакових оцінок з фізики і курсів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаною оцінкою з фізики.

Варіант 13.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за назвою спеціальності, в останню чергу (для однакових оцінок з третього предмету і спеціальностей) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за назвою спеціальності, в останню чергу (для однакових оцінок з третього предмету і спеціальностей) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаною оцінкою з третього предмету.

Варіант 14.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за номером курсу, в останню чергу (для однакових оцінок з третього предмету і курсів) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за номером курсу, в останню чергу (для однакових оцінок з третього предмету і курсів) – за прізвищем.

3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаною оцінкою з третього предмету.

Варіант 15.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за значенням оцінки з математики, в останню чергу (для однакових середніх балів і оцінок з математики) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за значенням оцінки з математики, в останню чергу (для однакових середніх балів і оцінок з математики) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем та з вказаним середнім балом та вказаною оцінкою з математики.

Варіант 16.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за значенням оцінки з третього предмету, в останню чергу (для однакових середніх балів і оцінок з третього предмету) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за значенням оцінки з третього предмету, в останню чергу (для однакових середніх балів і оцінок з третього предмету) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем та з вказаним середнім балом і вказаною оцінкою з третього предмету.

Варіант 17.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за значенням середнього балу, в останню

чергу (для однакових оцінок з третього предмету і середніх балів) – за прізвищем за зростанням – в алфавітному порядку.

2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за значенням середнього балу, в останню чергу (для однакових оцінок з третього предмету і середніх балів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем та з вказаним середнім балом і вказаною оцінкою з третього предмету.

Варіант 18.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за прізвищем за спаданням – в зворотному до алфавітного порядку, в останню чергу (для однакових спеціальностей і прізвищ) – за номером курсу.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за прізвищем, в останню чергу (для однакових спеціальностей і прізвищ) – за номером курсу.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та на вказаній спеціальності.

Варіант 19.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за прізвищем за зростанням – в алфавітному порядку, в останню чергу (для однакових курсів і прізвищ) – за назвою спеціальності.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за прізвищем, в останню чергу (для однакових курсів і прізвищ) – за назвою спеціальності.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та на вказаній спеціальності.

Варіант 20.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за прізвищем за спаданням – в зворотному до алфавітного порядку, в останню чергу (для однакових середніх балів і прізвищ) – за номером курсу.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за прізвищем, в останню чергу (для однакових середніх балів і прізвищ) – за номером курсу.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та з вказаним середнім балом.

Варіант 21.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за номером курсу, в останню чергу (для однакових спеціальностей і курсів) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за номером курсу, в останню чергу (для однакових спеціальностей і курсів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності на вказаному курсі.

Варіант 22.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за назвою спеціальності, в останню чергу (для однакових курсів і спеціальностей) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за назвою спеціальності, в останню чергу (для однакових курсів і спеціальностей) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі на вказаній спеціальності.

Варіант 23.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням середнього балу, в останню чергу (для однакових спеціальностей і середніх балів) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням середнього балу, в останню чергу (для однакових спеціальностей і середніх балів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаним середнім балом.

Варіант 24.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням середнього балу, в останню чергу (для однакових курсів і середніх балів) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням середнього балу, в останню чергу (для однакових курсів і середніх балів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаним середнім балом.

Варіант 25.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням оцінки з математики, в останню чергу (для однакових спеціальностей і оцінок з математики) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням оцінки з математики, в останню чергу (для однакових спеціальностей і оцінок з математики) – за прізвищем.

3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаною оцінкою з математики.

Варіант 26.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням оцінки з математики, в останню чергу (для однакових курсів і оцінок з математики) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням оцінки з математики, в останню чергу (для однакових курсів і оцінок з математики) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаною оцінкою з математики.

Варіант 27.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням оцінки з третього - профільного - предмету, в останню чергу (для однакових спеціальностей і оцінок з третього предмету) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за значенням оцінки з третього – профільного – предмету, в останню чергу (для однакових спеціальностей і оцінок з третього предмету) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаною оцінкою з третього предмету.

Варіант 28.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням оцінки з третього – профільного – предмету, в останню чергу (для однакових курсів і оцінок з

третього предмету) – за прізвищем за спаданням – в зворотному до алфавітного порядку.

2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за значенням оцінки з третього – профільного – предмету, в останню чергу (для однакових курсів і оцінок з третього предмету) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаною оцінкою з третього предмету.

Варіант 29.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за назвою спеціальності, в останню чергу (для однакових середніх балів і спеціальностей) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за назвою спеціальності, в останню чергу (для однакових середніх балів і спеціальностей) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаним середнім балом.

Варіант 30.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за номером курсу, в останню чергу (для однакових середніх балів і курсів) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за номером курсу, в останню чергу (для однакових середніх балів і курсів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаним середнім балом.

Варіант 31.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з фізики, в другу чергу (для однакових оцінок з фізики) – за назвою спеціальності, в останню чергу (для однакових оцінок з фізики і спеціальностей) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з фізики, в другу чергу (для однакових оцінок з фізики) – за назвою спеціальності, в останню чергу (для однакових оцінок з фізики і спеціальностей) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаною оцінкою з фізики.

Варіант 32.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з фізики, в другу чергу (для однакових оцінок з фізики) – за номером курсу, в останню чергу (для однакових оцінок з фізики і курсів) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з фізики, в другу чергу (для однакових оцінок з фізики) – за номером курсу, в останню чергу (для однакових оцінок з фізики і курсів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаною оцінкою з фізики.

Варіант 33.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за назвою спеціальності, в останню чергу (для однакових оцінок з третього предмету і спеціальностей) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за назвою спеціальності, в останню чергу (для однакових оцінок з третього предмету і спеціальностей) – за прізвищем.

3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаній спеціальності та вказаною оцінкою з третього предмету.

Варіант 34.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за номером курсу, в останню чергу (для однакових оцінок з третього предмету і курсів) – за прізвищем за спаданням – в зворотному до алфавітного порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за номером курсу, в останню чергу (для однакових оцінок з третього предмету і курсів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та вказаною оцінкою з третього предмету.

Варіант 35.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за значенням оцінки з математики, в останню чергу (для однакових середніх балів і оцінок з математики) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за значенням оцінки з математики, в останню чергу (для однакових середніх балів і оцінок з математики) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем та з вказаним середнім балом та вказаною оцінкою з математики.

Варіант 36.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за значенням оцінки з третього предмету, в останню чергу (для однакових середніх

балів і оцінок з третього предмету) – за прізвищем за спаданням – в зворотному до алфавітного порядку.

2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за значенням оцінки з третього предмету, в останню чергу (для однакових середніх балів і оцінок з третього предмету) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем та з вказаним середнім балом і вказаною оцінкою з третього предмету.

Варіант 37.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за значенням середнього балу, в останню чергу (для однакових оцінок з третього предмету і середніх балів) – за прізвищем за зростанням – в алфавітному порядку.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням оцінки з третього – профільного – предмету, в другу чергу (для однакових оцінок з третього предмету) – за значенням середнього балу, в останню чергу (для однакових оцінок з третього предмету і середніх балів) – за прізвищем.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем та з вказаним середнім балом і вказаною оцінкою з третього предмету.

Варіант 38.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за прізвищем за спаданням – в зворотному до алфавітного порядку, в останню чергу (для однакових спеціальностей і прізвищ) – за номером курсу.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за назвою спеціальності, в другу чергу (для однакових спеціальностей) – за прізвищем, в останню чергу (для однакових спеціальностей і прізвищ) – за номером курсу.

3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та на вказаній спеціальності.

Варіант 39.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за прізвищем за зростанням – в алфавітному порядку, в останню чергу (для однакових курсів і прізвищ) – за назвою спеціальності.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за номером курсу, в другу чергу (для однакових курсів) – за прізвищем, в останню чергу (для однакових курсів і прізвищ) – за назвою спеціальності.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та на вказаній спеціальності.

Варіант 40.

1. Програма має дати користувачеві можливість фізично впорядкувати масив в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за прізвищем за спаданням – в зворотному до алфавітного порядку, в останню чергу (для однакових середніх балів і прізвищ) – за номером курсу.
2. Програма має будувати індексний масив, який забезпечує наступне індексне впорядкування: в першу чергу – за значенням середнього балу, в другу чергу (для однакових середніх балів) – за прізвищем, в останню чергу (для однакових середніх балів і прізвищ) – за номером курсу.
3. За допомогою бінарного пошуку визначити, чи навчається студент із вказаним користувачем прізвищем на вказаному курсі та з вказаним середнім балом.

Лабораторна робота № 9.3. Опрацювання масиву структур

Мета роботи

Навчитися опрацьовувати масив структур з об’єднаннями.

Питання, які необхідно вивчити та пояснити на захисті

1) *Загальний синтаксис оголошення масивів та переліків, структур і об’єднань.*
2) *Доступ до елементів масивів та переліків, структур і об’єднань.*
3) *Дії з масивами та переліками, структурами і об’єднаннями.*
4) *Ініціалізація масивів та переліків, структур і об’єднань.*
5) *Внутрішня реалізація масивів та переліків, структур і об’єднань.*
6) *Передавання та опрацювання масивів та структур у функціях.*
7) *Загальний синтаксис оголошення та формування динамічних масивів.*
8) *Загальна схема фізичного впорядкування масиву.*
9) *Загальна схема індексного впорядкування масиву.*
10) *Загальна схема послідовного пошуку.*
11) *Загальна схема бінарного пошуку.*

Оформлення звіту

Вимоги та зразок оформлення звіту наведені у вступі до лабораторного практикуму.

Приклади розв’язання лабораторних завдань

Взірці виконання лабораторних завдань – див. в розділі Приклади.

Варіанти лабораторних завдань

Кожна програма має містити меню. Необхідно передбачити контроль помилок користувача при введенні даних.

Необхідні програмі дані слід реалізувати за допомогою динамічного масиву структур, кількість елементів у якому визначає користувач. Програма має забезпечувати можливість зчитування всіх даних із файлу та запису всього масиву у файл. Ім’я файлу має вводитися з клавіатури.

При розробці програми застосувати технологію низхідного проектування. Логічно закінчені фрагменти оформити у вигляді функцій, всі необхідні дані яким передаються через список параметрів. Використовувати глобальні змінні – не можна.

Кожна функція має виконувати лише одну роль, і ця роль має бути відображена у назві функції.

«Функція, яка повертає / обчислює / шукає ...» – має не виводити ці значення, а повернути їх у місце виклику як результат функції або як відповідний вихідний параметр.

Варіант 1.

Звітна відомість результатів екзаменаційної сесії студентської групи для кожного студента містить прізвище, ініціали і оцінки з п’яти предметів.

Скласти програму, за допомогою якої можна корегувати список (добавляти, вилучати, редагувати інформацію) і отримувати:

- список всіх студентів;
- список студентів, що склали іспити тільки на «5»;
- список студентів, що мають трійки;
- список студентів, що мають двійки. При цьому студент, що має більш ніж одну двійку, виключається із списку.

Варіант 2.

Підприємство має місцеву телефонну станцію. Телефонний довідник цього підприємства для кожного номера телефону містить номер приміщення і список службовців, що сидять в цьому приміщенні.

Скласти програму, яка:

- корегує базу (добавляє, вилучає, редагує інформацію);
- за номером телефону видає номер приміщення і список людей, що сидять в ньому;
- за номером приміщення видає номер телефону;
- за прізвищем службовця видає номер телефону і номер приміщення.

Номер телефону – двозначний. У одному приміщенні може знаходитися від одного до чотирьох службовців.

Варіант 3.

У готелі є 15 номерів, з них 5 одномісних і 10 двомісних. Скласти програму, яка заповнює та корегує дані про мешканців (добавляє, вилучає, редагує інформацію) і за прізвищем визначає номер, де проживає мешканець.

Програма запрошує прізвище мешканця.

- Якщо мешканця з таким прізвищем немає, про це видається повідомлення.

- Якщо мешканець з таким прізвищем в готелі єдиний, програма видає прізвище мешканця і номер помешкання.
- Якщо в готелі проживає два або більше мешканців з таким прізвищем, програма додатково запрошує ініціали.

Варіант 4.

Є список службовців. Для кожного службовця вказано прізвище і ініціали, назву посади, рік прийому на роботу і оклад (величину заробітної плати).

Написати програму, що виконує наступні дії:

- корегування списку з клавіатури (добавлення, вилучення, редагування інформації);
- сортування за прізвищем, окладом або роком прийому на роботу;
- вивід на екран інформації про службовця, прізвище якого введене з клавіатури.

Варіант 5.

Розклад електричок зберігається у вигляді сукупності записів. Кожен запис містить назву пункту призначення, позначки типу «звичайний», «підвищеного комфорту», «швидкісний експрес» та час відправлення.

Написати програму, що виконує наступні дії:

- корегування розкладу з клавіатури (добавлення, вилучення, редагування інформації);
- сортування за станцією призначення або за часом відправлення;
- вивід на екран інформації про поїзди, що відходять після введеного часу.

Варіант 6.

Є список товарів. Для кожного товару вказані його назва, вартість одиниці товару в гривнях, кількість і одиниця вимірювання (наприклад, кількість: 100 шт., одиниця вимірювання: упаковка по 20 кг.).

Написати програму, що виконує наступні дії:

- корегування списку з клавіатури (добавлення, вилучення, редагування інформації);
- сортування по назві товару або за загальною вартістю;
- вивід на екран інформації про товар, назва якого введена з клавіатури;
- вивід на екран інформації про товари із заданого з клавіатури діапазону вартості.

Варіант 7.

Є список товарів. Для кожного товару вказані його назва, назва магазину, в якому продається товар, вартість одиниці товару в гривнях і його кількість з вказівкою одиниці вимірювання (наприклад, кількість: 100 шт., одиниця вимірювання: упаковка по 20 кг.).

Написати програму, що виконує наступні дії:

- корегування списку з клавіатури (добавлення, вилучення, редагування інформації);
- сортування за назвою товару або за назвою магазину;
- вивід на екран інформації про товар, назва якого введена з клавіатури;
- вивід на екран інформації про товари, що продаються в магазині, назва якого введена з клавіатури.

Варіант 8.

Є список студентської групи. Кожен елемент списку містить прізвище студента і три екзаменаційні оцінки, причому список ніяк не впорядкований.

Скласти програму, яка корегує список (добавляє, вилучає, редагує інформацію) і сортує його залежно від вибору користувача: або за середнім балом, або за прізвищами – в алфавітному порядку, або за оцінками із заданого предмету.

Варіант 9.

Є список товарів. Для кожного товару вказані його назва, назва магазину, в якому продається товар, вартість одиниці товару в гривнях і його кількість з вказівкою одиниці вимірювання (наприклад, кількість: 100 шт., одиниця вимірювання: упаковка по 20 кг.).

Написати програму, що виконує наступні дії:

- корегування списку з клавіатури (добавлення, вилучення, редагування інформації);
- сортування за назвою магазину або за загальною вартістю;
- вивід на екран інформації про товари, що продаються в магазині, назва якого введена з клавіатури;
- вивід на екран інформації про товари із заданого з клавіатури діапазону вартості.

Варіант 10.

Описати структуру з іменем Route, що містить наступні поля:

- назву початкового пункту маршруту;
- назву кінцевого пункту маршруту;

- номер маршруту.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Route;
- впорядкування масиву структур за номерами маршрутів;
- вивід на екран інформації про маршрут, номер якого введений з клавіатури; якщо такого маршруту немає, вивести на екран відповідне повідомлення.

Варіант 11.

Описати структуру з іменем Route, що містить наступні поля:

- назву початкового пункту маршруту;
- назву кінцевого пункту маршруту;
- номер маршруту.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Route;
- впорядкування масиву структур за номерами маршрутів;
- вивід на екран інформації про маршрути, які починаються або закінчуються в пункті, назва якого введена з клавіатури; якщо таких маршрутів немає, вивести на екран відповідне повідомлення.

Варіант 12.

Описати структуру з іменем Note, що містить наступні поля:

- прізвище, ім’я;
- номер телефону;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Note;
- впорядкування масиву структур за датами днів народження;
- вивід на екран інформації про людину, номер телефону якої введений з клавіатури; якщо такої людини немає, вивести на екран відповідне повідомлення.

Варіант 13.

Описати структуру з іменем Note, що містить наступні поля:

- прізвище, ім’я;
- номер телефону;

- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Note;
- впорядкування масиву структур за прізвищами (в алфавітному порядку);
- вивід на екран інформації про людей, що народилися в тому місяці, значення якого введене з клавіатури; якщо таких немає, вивести на екран відповідне повідомлення.

Варіант 14.

Описати структуру з іменем Note, що містить наступні поля:

- прізвище, ім'я;
- номер телефону;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Note;
- впорядкування масиву структур за телефонними номерами;
- вивід на екран інформації про людину, прізвище якої введене з клавіатури; якщо такої немає, вивести на екран відповідне повідомлення.

Варіант 15.

Описати структуру з іменем Zodiac, що містить наступні поля:

- прізвище, ім’я;
- знак Зодіаку;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Zodiac;
- впорядкування масиву структур за датами днів народження.
- вивід на екран інформації про людину, чиє прізвище введене з клавіатури; якщо такої людини немає, вивести на екран відповідне повідомлення.

Варіант 16.

Описати структуру з іменем Zodiac, що містить наступні поля:

- прізвище, ім’я;
- знак Зодіаку;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Zodiac;
- впорядкування масиву структур за прізвищами (в алфавітному порядку).
- вивід на екран інформації про людей, що народилися під знаком, найменування якого введене з клавіатури; якщо таких немає, вивести на екран відповідне повідомлення.

Варіант 17.

Описати структуру з іменем Zodiac, що містить наступні поля:

- прізвище, ім'я;
- знак Зодіаку;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Zodiac;
- впорядкування масиву структур за знаками Зодіаку;
- вивід на екран інформації про людей, що народилися в тому місяці, значення якого введене з клавіатури; якщо таких немає, вивести на екран відповідне повідомлення.

Варіант 18.

Описати структуру з іменем Price, що містить наступні поля:

- назва товару;
- назва магазину, в якому продається товар;
- вартість товару в гривнях.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Price;
- впорядкування масиву структур за алфавітним порядком назв магазинів;
- вивід на екран інформації про товар, назва якого введена з клавіатури; якщо таких товарів немає, вивести на екран відповідне повідомлення.

Варіант 19.

Описати структуру з іменем Price, що містить наступні поля:

- назва товару;
- назва магазину, в якому продається товар;
- вартість товару в гривнях.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Price;
- впорядкування масиву структур за алфавітним порядком назв товарів;
- вивід на екран інформації про товари, що продаються в магазині, назва якого введена з клавіатури; якщо такого магазину немає, вивести на екран відповідне повідомлення.

Варіант 20.

Описати структуру з іменем Bill, що містить наступні поля:

- розрахунковий рахунок платника;
- розрахунковий рахунок одержувача;
- перерахована сума в гривнях.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Bill;
- впорядкування масиву структур за алфавітним порядок розрахункових рахунків платників;
- вивід на екран інформації про суму, зняту з розрахункового рахунку платника, введеного з клавіатури; якщо такого розрахункового рахунку немає, вивести на екран відповідне повідомлення.

Варіант 21.

Звітна відомість результатів екзаменаційної сесії студентської групи для кожного студента містить прізвище, ініціали і оцінки з п’яти предметів.

Скласти програму, за допомогою якої можна корегувати список (добавляти, вилучати, редагувати інформацію) і отримувати:

- список всіх студентів;
- список студентів, що склали іспити тільки на «5»;
- список студентів, що мають трійки;
- список студентів, що мають двійки. При цьому студент, що має більш ніж одну двійку, виключається із списку.

Варіант 22.

Підприємство має місцеву телефонну станцію. Телефонний довідник цього підприємства для кожного номера телефону містить номер приміщення і список службовців, що сидять в цьому приміщенні.

Скласти програму, яка:

- корегує базу (добавляє, вилучає, редагує інформацію);
- за номером телефону видає номер приміщення і список людей, що сидять в ньому;
- за номером приміщення видає номер телефону;
- за прізвищем службовця видає номер телефону і номер приміщення.

Номер телефону – двозначний. У одному приміщенні може знаходитися від одного до чотирьох службовців.

Варіант 23.

У готелі є 15 номерів, з них 5 одномісних і 10 двомісних. Скласти програму, яка заповнює та корегує дані про мешканців (добавляє, вилучає, редагує інформацію) і за прізвищем визначає номер, де проживає мешканець.

Програма запрошує прізвище мешканця.

- Якщо мешканця з таким прізвищем немає, про це видається повідомлення.
- Якщо мешканець з таким прізвищем в готелі єдиний, програма видає прізвище мешканця і номер помешкання.
- Якщо в готелі проживає два або більше мешканців з таким прізвищем, програма додатково запрошує ініціали.

Варіант 24.

Є список службовців. Для кожного службовця вказано прізвище і ініціали, назву посади, рік прийому на роботу і оклад (величину заробітної плати).

Написати програму, що виконує наступні дії:

- корегування списку з клавіатури (добавлення, вилучення, редагування інформації);
- сортування за прізвищем, окладом або роком прийому на роботу;
- вивід на екран інформації про службовця, прізвище якого введене з клавіатури.

Варіант 25.

Розклад електричок зберігається у вигляді сукупності записів. Кожен запис містить назву пункту призначення, позначки типу «звичайний», «підвищеного комфорту», «швидкісний експрес» та час відправлення.

Написати програму, що виконує наступні дії:

- корегування розкладу з клавіатури (добавлення, вилучення, редагування

інформації);

- сортування за станцією призначення або за часом відправлення;
- вивід на екран інформації про поїзди, що відходять після введеного часу.

Варіант 26.

Є список товарів. Для кожного товару вказані його назва, вартість одиниці товару в гривнях, кількість і одиниця вимірювання (наприклад, кількість: 100 шт., одиниця вимірювання: упаковка по 20 кг.).

Написати програму, що виконує наступні дії:

- корегування списку з клавіатури (добавлення, вилучення, редагування інформації);
- сортування по назві товару або за загальною вартістю;
- вивід на екран інформації про товар, назва якого введена з клавіатури;
- вивід на екран інформації про товари із заданого з клавіатури діапазону вартості.

Варіант 27.

Є список товарів. Для кожного товару вказані його назва, назва магазину, в якому продається товар, вартість одиниці товару в гривнях і його кількість з вказівкою одиниці вимірювання (наприклад, кількість: 100 шт., одиниця вимірювання: упаковка по 20 кг.).

Написати програму, що виконує наступні дії:

- корегування списку з клавіатури (добавлення, вилучення, редагування інформації);
- сортування за назвою товару або за назвою магазину;
- вивід на екран інформації про товар, назва якого введена з клавіатури;
- вивід на екран інформації про товари, що продаються в магазині, назва якого введена з клавіатури.

Варіант 28.

Є список студентської групи. Кожен елемент списку містить прізвище студента і три екзаменаційні оцінки, причому список ніяк не впорядкований.

Скласти програму, яка корегує список (добавляє, вилучає, редагує інформацію) і сортує його залежно від вибору користувача: або за середнім балом, або за прізвищами – в алфавітному порядку, або за оцінками із заданого предмету.

Варіант 29.

Є список товарів. Для кожного товару вказані його назва, назва магазину, в якому продається товар, вартість одиниці товару в гривнях і його кількість з вказівкою одиниці вимірювання (наприклад, кількість: 100 шт., одиниця вимірювання: упаковка по 20 кг.).
Написати програму, що виконує наступні дії:

- корегування списку з клавіатури (добавлення, вилучення, редагування інформації);
- сортування за назвою магазину або за загальною вартістю;
- вивід на екран інформації про товари, що продаються в магазині, назва якого введена з клавіатури;
- вивід на екран інформації про товари із заданого з клавіатури діапазону вартості.

Варіант 30.

Описати структуру з іменем Route, що містить наступні поля:

- назву початкового пункту маршруту;
- назву кінцевого пункту маршруту;
- номер маршруту.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Route;
- впорядкування масиву структур за номерами маршрутів;
- вивід на екран інформації про маршрут, номер якого введений з клавіатури; якщо такого маршруту немає, вивести на екран відповідне повідомлення.

Варіант 31.

Описати структуру з іменем Route, що містить наступні поля:

- назву початкового пункту маршруту;
- назву кінцевого пункту маршруту;
- номер маршруту.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Route;
- впорядкування масиву структур за номерами маршрутів;
- вивід на екран інформації про маршрути, які починаються або закінчуються в пункті, назва якого введена з клавіатури; якщо таких маршрутів немає, вивести на екран відповідне повідомлення.

Варіант 32.

Описати структуру з іменем Note, що містить наступні поля:

- прізвище, ім’я;
- номер телефону;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Note;
- впорядкування масиву структур за датами днів народження;
- вивід на екран інформації про людину, номер телефону якої введений з клавіатури; якщо такої людини немає, вивести на екран відповідне повідомлення.

Варіант 33.

Описати структуру з іменем Note, що містить наступні поля:

- прізвище, ім’я;
- номер телефону;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Note;
- впорядкування масиву структур за прізвищами (в алфавітному порядку);
- вивід на екран інформації про людей, що народилися в тому місяці, значення якого введене з клавіатури; якщо таких немає, вивести на екран відповідне повідомлення.

Варіант 34.

Описати структуру з іменем Note, що містить наступні поля:

- прізвище, ім'я;
- номер телефону;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Note;
- впорядкування масиву структур за телефонними номерами;
- вивід на екран інформації про людину, прізвище якої введене з клавіатури; якщо такої немає, вивести на екран відповідне повідомлення.

Варіант 35.

Описати структуру з іменем Zodiac, що містить наступні поля:

- прізвище, ім’я;
- знак Зодіаку;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Zodiac;
- впорядкування масиву структур за датами днів народження.
- вивід на екран інформації про людину, чиє прізвище введене з клавіатури; якщо такої людини немає, вивести на екран відповідне повідомлення.

Варіант 36.

Описати структуру з іменем Zodiac, що містить наступні поля:

- прізвище, ім’я;
- знак Зодіаку;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Zodiac;
- впорядкування масиву структур за прізвищами (в алфавітному порядку).
- вивід на екран інформації про людей, що народилися під знаком, найменування якого введене з клавіатури; якщо таких немає, вивести на екран відповідне повідомлення.

Варіант 37.

Описати структуру з іменем Zodiac, що містить наступні поля:

- прізвище, ім’я;
- знак Зодіаку;
- день народження (масив з трьох чисел).

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Zodiac;
- впорядкування масиву структур за знаками Зодіаку;
- вивід на екран інформації про людей, що народилися в тому місяці, значення якого введене з клавіатури; якщо таких немає, вивести на екран відповідне повідомлення.

Варіант 38.

Описати структуру з іменем Price, що містить наступні поля:

- назва товару;
- назва магазину, в якому продається товар;
- вартість товару в гривнях.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Price;
- впорядкування масиву структур за алфавітним порядком назв магазинів;
- вивід на екран інформації про товар, назва якого введена з клавіатури; якщо таких товарів немає, вивести на екран відповідне повідомлення.

Варіант 39.

Описати структуру з іменем Price, що містить наступні поля:

- назва товару;
- назва магазину, в якому продається товар;
- вартість товару в гривнях.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Price;
- впорядкування масиву структур за алфавітним порядком назв товарів;
- вивід на екран інформації про товари, що продаються в магазині, назва якого введена з клавіатури; якщо такого магазину немає, вивести на екран відповідне повідомлення.

Варіант 40.

Описати структуру з іменем Bill, що містить наступні поля:

- розрахунковий рахунок платника;
- розрахунковий рахунок одержувача;
- перерахована сума в гривнях.

Написати програму, що виконує наступні дії:

- введення даних з клавіатури в масив, що складається з елементів типу Bill;
- впорядкування масиву структур за алфавітним порядок розрахункових рахунків платників;
- вивід на екран інформації про суму, зняту з розрахункового рахунку платника, введеного з клавіатури; якщо такого розрахункового рахунку немає, вивести на

екран відповідне повідомлення.

Питання та завдання для контролю знань

1	**Які операції можна виконувати над даними типу «структура»?** (позначте відповідні елементи списку).		порівняння
			присвоєння
			ввід значення з клавіатури
			вивід значення на екран

2	**Як визначається обсяг пам'яті, необхідний для зберігання структури?** (позначте відповідні елементи списку).		дорівнює розміру найдовшого поля
			дорівнює розміру найкоротшого поля
			дорівнює сумі розмірів всіх полів
			дорівнює добутку розмірів всіх полів
			дорівнює середньому арифметичному розмірів всіх полів

3	**Як визначається обсяг пам'яті, необхідний для зберігання об'єднання?** (позначте відповідні елементи списку).		дорівнює середньому арифметичному розмірів всіх полів
			дорівнює розміру найдовшого поля
			дорівнює розміру найкоротшого поля
			дорівнює сумі розмірів всіх полів
			дорівнює добутку розмірів всіх полів

4	За наведеним нижче фрагментом програми визначити істинні твердження (позначте відповідні елементи списку).		

	`int a;`
	
	`struct S  // велика літера`
	`{`
	`  int a;`
	`  double b;`
	`} s;  // мала літера`
			Присвоєння значення змінній **a** приведе до автоматичного присвоєння полю **a** структури **s** того самого значення

			Компілятор розглядає змінну **a** та поле **a** структури **s** як різні змінні
			Компілятор видає помилку

5	Поставте знак «+», якщо твердження вірне, та знак «–» в протилежному випадку.	
		Структура має складатися із полів різних типів
		Тип поля структури може бути будь-яким, крім файлового
		Структуру можна вивести на екран, вказавши в списку виводу її ім'я
		Структури можна порівнювати на рівність та нерівність
		Поле структури, в свою чергу, може бути структурою

6	Задано наступні оголошення. Які із наступних команд будуть хибними і чому? (позначте відповідні елементи списку та вкажіть в останньому стовпчику причину хибності)			

	`typedef unsigned int Score;`
	
	`struct Exam`
	`{`
	`  Score  Min, Max;`
	`  double Avg, StDev;`
	`};`
	
	`Exam exam1, exam2;`
			`exam1 = exam2;`	

			`cin >> exam1.Avg;`	
			`exam1.Max = exam2;`	
			`exam2.Min = exam1.Max;`	
			`exam2.Min = exam1.Avg;`	
			`cout << exam1 << endl;`	

7. Чи вірні наступні твердження:
   a) опис структури починається з ключового слова `struct` і містить перелік оголошень елементів структури, записаний у фігурних дужках;
   b) за словом `struct` має бути записаний ідентифікатор, який називається тегом (іменем типу) структури;
   c) тег структури використовується в якості імені типу при опису змінних;

d) імена елементів структури можуть збігатися з іменами змінних в тій самій області видимості;

e) ім’я тегу структури може збігатися з іменами змінних в тій самій області видимості;

f) ім’я тегу структури може збігатися з іменами елементів цієї ж структури;

g) імена різних елементів різних структур можуть збігатися;

h) за описом структури (після правої закриваючої фігурної дужки) обов’язково має бути записаний перелік змінних;

i) змінні x, y, z – різних типів:

1)

```
struct s { int a; float f; } x, y;
struct s z;
```

2)

```
typedef struct { int a; float f; } s;
    s x, y;
    struct { int a; float f; } z;
```

3)

```
struct s { int a; float f; };
typedef struct s new_s;
struct s x; new_s y, z;
```

4)

```
struct s { int a; float f; };
    typedef struct s s1;
    typedef struct s s2;
        s1 x, y; s2 z;
```

j) змінні x, y, z – одного типу:

1)

```
struct { int a; float f; } x, y;
struct { int a; float f; } z;
```

2)

```
struct { int a; float f; } x, y;
struct { float f; int a; } z;
```

k) для доступу до елементів структури використовується операція . (крапка);

l) структури не можуть бути вкладеними;

m) структурну змінну при її оголошенні можна ініціалізувати переліком константних виразів, записаних всередині фігурних дужок.

8. Яким чином в мові С визначається еквівалентність типів? Яка еквівалентність типів розглядається: структурна чи іменна? Чим вони відрізняються?

9. Описати у вигляді структури наступні поняття:
   a) дата (число, місяць, рік);
   b) адреса (країна, місто, вулиця, будинок, квартира);
   c) трикутник (дві сторони та кут між ними);
   d) коло (радіус та центр);
   e) розклад занять студента групи КН-16 фіз.-мат. факультету (день тижня, предмети (з вказівкою – лекції чи семінари), години занять, аудиторія, прізвище викладача);
   f) результати перевірки контрольної роботи (номер групи, номер контрольної роботи, тема, таблиця із 25 рядків с полями: прізвище студента, варіант, інформація про кожну із п’яти задач (її номер, оцінка за її розв’язок, характеристика помилок), підсумкова оцінка студента за цю контрольну роботу).

10. Використовуючи визначений в задачі 9 тип, описати змінну цього типу та присвоїти їй значення:
    a) дата – 17 березня 2014 року;
    b) адреса – Україна, Дрогобич, вул. Стрийська, буд. 3, кв. 5;
    c) трикутник – 5, 6.7, 35°;
    d) коло – радіус 4.567, центр (1.4, 5.6);
    e) розклад занять студента КН-16 фіз.-мат. факультету – вівторок, математичний аналіз (лекція) – 1 пара, 58 ауд., Шаран В.Л., Web-програмування (семінар) – 2 пара, 5 ауд., Карпин Д.С., Інформаційні технології (семінар) – 3 пара, 68 ауд., Шілінг А.Ю.

11. Що виведе програма?

```
#include <iostream>
using namespace std;

int main()
{
    struct data1
    {
        char c[4];
        char *s;
    } d1 = { "abc", "def" };
```

```
    struct data2
    {
        char *cp;
        struct data1 inf;
    } d2 = { "ghi", { "jkl", "mno"} };

    cout << "d1.c[0]=" << d1.c[0]
         << "  *d1.s=" << *d1.s << endl;
    cout << "d1.c=" << d1.c
         << "  d1.s=" << d1.s << endl;
    cout << "d2.cp=" << d2.cp
         << "  d2.inf.s=" << d2.inf.s << endl;
    cout << "++d2.cp=" << ++d2.cp
         << "  ++d2.inf.s=" << ++d2.inf.s << endl;

    return 0;
}
```

12. Чи вірні наступні твердження:
    a) опис об'єднання починається з ключового слова `union` і містить перелік оголошень елементів об'єднання, записаний в фігурних дужках;
    b) кожний елемент об'єднання розміщується в пам'яті починаючи з одної і тої самої адреси; обсяг пам'яті для кожного елемента виділяється відповідно до його розміру;
    c) для кожного із елементів об'єднання виділяється одна і та ж область пам'яті;
    d) всі проблеми, пов'язані з вирівнюванням, вирішує компілятор;
    e) в кожний момент часу об'єднання може містити значення лише одного із його елементів;
    f) всі операції, які можна застосовувати до структур, можна застосовувати і до об'єднань;
    g) «неузгодженість» при роботі з активним варіантом об'єднання контролюється компілятором.

13. Описати тип, за допомогою якого можна організувати зберігання даних про різні види транспорту: вантажівки, автобуси, легкові автомобілі та мотоцикли. Для кожного виду транспорту є як загальні характеристики (власник, рік виробництва та модель), так і індивідуальні (для вантажівок – кількість осей, вантажопідйомність; для автобусів – кількість місць для пасажирів, для легкових автомобілів – кількість дверей (2 чи 4), для мотоциклів – тип двигуна (двох- чи чотирьохтактний)).

14. Чи вірні наступні твердження:
    a) до структур однакового типу можна застосовувати операцію присвоєння;

b) до структур однакового типу, які не містять вкладених структур, можна застосовувати операції порівняння (виконується по елементне порівняння);

c) параметром функції може бути вказівник на структуру, але не сама структура;

d) параметри функції – структури передаються за значенням;

e) результатом роботи функції може бути структура;

f) результатом роботи функції може бути вказівник на структуру;

g) функція `sizeof(struct any)` видає результат, що дорівнює сумі розмірів всіх полів цієї структури;

h) до структур можна застосовувати операцію отримання адреси;

15. Перелічити всі операції, які можна застосовувати до структур.

16. Нехай точка на площині описана наступним чином:

```
struct point{ int x;  int y; };
```

Чи вірно розв'язана задача: «описати функцію, яка присвоює значення структурі типу `struct point` »

a)

```
void assign_to_point( struct point p, int a, int b )
{ p.x = a;  p.y = b; }
```

b)

```
void assign_to_point( struct point *p, int a, int b )
{ (*p).x = a;  (*p).y = b; }
```

c)

```
void assign_to_point( struct point *p, int a, int b )
{ *p.x = a;  *p.y = b; }
```

d)

```
void assign_to_point( struct point *p, int a, int b )
{ p->x = a;  p->y = b; }
```

17. Нехай точка на площині описана наступним чином:

```
struct point{ int x;  int y; };
```

Чи вірно розв'язана задача: «описати функцію, яка створює точку із двох цілих чисел»

a)

```
struct point create_point( int a, int b )
{
    struct point p;
    p.x = a;
    p.y = b;
    return p;
}
```

b)

```
struct point *create_point( int a, int b )
{
    struct point p;
    p.x = a;
    p.y = b;
    return &p;
}
```

c)

```
struct point *create_point( int a, int b )
{
    struct point *pp;
    pp->x = a;
    pp->y = b;
    return pp;
}
```

d)

```
struct point *create_point( int a, int b )
{
    struct point *pp;
    pp = (struct point *) malloc(sizeof(struct point));
    pp->x = a;
    pp->y = b;
    return pp;
}
```

e)

```
struct point *create_point( int a, int b )
{
    struct point *pp;
    pp = new struct point;
    pp->x = a;
    pp->y = b;
    return pp;
}
```

18. Нехай точка на площині описана наступним чином:

    ```
    struct point{ int x;  int y; };
    ```

    Описати функцію, яка за трьома точками, котрі є вершинами деякого прямокутника, визначає його четверту вершину.

19. Описати у вигляді структури
    a) точку на площині;
    b) кольорову точку на площині;
    c) комплексне число;
    d) раціональне число.

Розробити сукупність операцій для даних цих типів; реалізувати кожну з них у вигляді функції.

20. Нехай «цілочисельне» коло на площині описане наступним чином:

```
struct point { int x; int y; };
struct circle{ int radius; struct point center; };
```

Нехай є масив `struct circle plane[50];`, який містить інформацію про кола на площині. Описати функцію, яка визначає

a) чи є серед цих кіл хоча б два концентричних кола;

b) чи є серед цих кіл хоча б два вкладені (не обов'язково концентричні) кола;

c) чи є серед цих кіл три кола, які попарно перетинаються;

d) чи є серед цих кіл хоча б одне «окреме», тобто те, яке не має спільних точок з жодним іншим колом масиву `plane`.

21. Нехай результати аналізу деякого тексту, що складається з англійських слів, міститься в наступному частотному словнику `dictionary`:

```
#define MAXSIZE 1000
#define LENGHT 20 /* максимальна довжина слова */

struct info
{
    int count;   /* кількість повторень слова в заданому тексті */
    char *alias; /* синонім цього слова */
};

struct elem
{
    char *word;
    struct info *data;
};

struct
{
    struct elem *tabl[ MAXSIZE ];
    int number; /* кількість слів в словнику */
}
dictionary;
```

Кожне слово (`word`) зустрічається в словнику лише один раз; синоніми (`alias`) можуть бути однаковими в різних слів.

Описати функцію, яка визначає

a) чи міститься задане слово в цьому словнику: результат – вказівник на відповідний елемент або `NULL`:

1) слова невпорядковані за алфавітом;

2) слова впорядковані за алфавітом;

b) яке слово частіше інших зустрічається в заданому тексті; якщо таких слів кілька, то взяти перше за алфавітом; результат роботи функції – вказівник на відповідний елемент:
   1) слова невпорядковані за алфавітом;
   2) слова впорядковані за алфавітом;
c) чи на кожну букву латинського алфавіту в цьому словнику найдеться хоча б одне слово:
   1) слова невпорядковані за алфавітом;
   2) слова впорядковані за алфавітом;
d) на які букви (букву) латинського алфавіту в цьому словнику більше всього слів; результат роботи функції – вказівник на рядок, що складається з цих букв, перелічених в алфавітному порядку:
   1) слова невпорядковані за алфавітом;
   2) слова впорядковані за алфавітом;
e) чи є в цьому словнику різні слова, які мають однакові синоніми.

22. Нехай результати аналізу деякого тексту міститься в частотному словнику (див. попередню задачу).

    Описати функцію
    a) `struct elem *add_word( char *word, char *alias )`, яка вставляє нове слово та інформацію про нього в словник `dictionary` (вважається, що такого слова в словнику ще немає, воно перший раз зустрілося в тексті):
       1) слова невпорядковані за алфавітом;
       2) слова впорядковані за алфавітом;

       Результат роботи функції – вказівник на вставлений елемент;
    b) `struct elem *update_word( char *word, char *alias )`, яка змінює значення синоніму для заданого слова (вважається, що таке слово в словнику обов'язково є):
       1) слова невпорядковані за алфавітом;
       2) слова впорядковані за алфавітом;

       Результат роботи функції – вказівник на елемент словника, в якому змінено значення синоніму;
    c) `struct elem *delete_word( char *word )`, яка видаляє задане слово із словника; результат роботи функції – вказівник на структуру, яка містить інформацію про видалене слово або `NULL`, якщо такого слова немає в словнику:
       1) слова невпорядковані за алфавітом;

2) слова впорядковані за алфавітом;

23. Написати програму, яка визначає кількість входжень кожного службового слова в задану програму на мові С. Ретельно продумати спосіб представлення інформації про службових словах.

24. Чи допустимо в мові С? Якщо «так» – опишіть семантику цих дій та поясніть, що буде виведено на екран; якщо «ні» – поясніть чому.

```
#include <iostream>

using namespace std;

struct data { char *s; int i; struct data *dp; };

int main()
{
    static struct data a[] = { { "abcd", 1, a+1 },
                               { "efgh", 2, a+2 },
                               { "ijkl", 3, a }
                             };
    struct data *p = a;
    int i;

    cout << "a[0].s=" << a[0].s << "  p->s=" << p->s
         << "  a[2].dp->s=" << a[2].dp->s << endl;

    for ( i = 0; i < 2; i++ )
        cout << "a[i].i=" << a[i].i
             << "  a[i].s[3]=" << a[i].s[3] << endl;

     cout << "++(p->s)=" << ++(p->s) << endl;
     cout << "a[(++p)->i].s=" << a[(++p)->i].s << endl;
     cout << "a[--(p->dp->i)].s=" << a[--(p->dp->i)].s << endl;

     return 0;
}
```

25. Нехай

```
struct s { int k; float *f; char *p[2]; };
struct s *ps;
```

Чи вірно присвоєне значення змінній ps та всім об’єктам, пов’язаним з нею:

```
char str[5] = "abcd";
ps = (struct s*) malloc(sizeof(struct s));
(*ps).k = 5;
ps -> f = (float*) malloc(sizeof(float));
*(ps -> f) = 3.1415;
(*ps).p[0] = (char*) malloc(5*sizeof(char));
(*ps).p[1] = (char*) malloc(10*sizeof(char));
(*ps).p[0] = str;
(*ps).p[1] = "abcdefghi";
```

26. Присвоїти значення змінній q та всім об’єктам, пов’язаним з нею (див. завдання 25):

```
struct data { double **p; char *s; int *a[2]; };
struct data *q;
```

27. Присвоїти значення змінній a та всім об’єктам, пов’язаним з нею (див. завдання 25):

```
struct b { double *q; int *(*p)[2]; };
struct b **a[1];
```

28. Присвоїти значення змінній x та всім об’єктам, пов’язаним з нею (див. завдання 25):

```
struct r
{
    double *a[3];
    char **s;
    union
    {
        int i;
        float f;
    } u;
};
struct r x[2];
```

29. Присвоїти значення змінній x та всім об’єктам, пов’язаним з нею (див. завдання 25):

```
struct a
{
    char ***s;
    char (*p)[2];
};
typedef struct a *data;
data x[2];
```

30. Присвоїти значення змінній pt та всім об’єктам, пов’язаним з нею (див. завдання 25):

```
struct t
{
    int **pi;
    double (*k)[2];
    char *p[2];
};
struct t *pt;
```

31. Присвоїти значення змінній a та всім об’єктам, пов’язаним з нею (див. завдання 25):

```
struct data
{
    int *i;
    int (*f)[3];
    char **s;
};
struct data a[2];
```

Предметний покажчик

Література

Основна

1. Ковалюк Т.В. Основи програмування. – К. BHV, 2005. – 384 с.
2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985.
3. Павловская Т.А. C/C++. Программирование на языке высокого уровня. СПб.: Питер, 2007. – 461 с.
4. Павловская Т.А., Щупак Ю.А. C/C++. Структурное программирование: Практикум. СПб.: Питер, 2005. – 239 с.
5. Прата Стивен. Язык программирования C++. Лекции и упражнения. Учебник: Пер. С англ./Стивен Прата – СПб.: ООО «ДиаСофтЮП», 2003. – 1194 с.

Додаткова

6. Кнут, Дональд, Эрвин Искусство программирования. 3-е издание. Том1. М.: Вильямс, 2001.
7. Кнут, Дональд, Эрвин Искусство программирования. 3-е издание. Том2. М.: Вильямс, 2001.
8. Кнут, Дональд, Эрвин Искусство программирования. 3-е издание. Том3. М.: Вильямс, 2001.
9. Брудно А.Л., Каплан Л.И. Московские олимпиады по программированию. – М.: Наука, 1990.
10. Вирт Н. Систематическое программирование. Введение. – М., Мир, 1977.
11. Ахо А.В., Хопкрофт Дж., Ульман Дж. Д. Структуры данных и алгоритмы. Пер. с англ.: М.: «Вильямс», 2001.– 384 с.: ил.
12. Себест Р. Основные концепции языков программирования. 5-е издание. М.: Вильямс, 2001.