МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

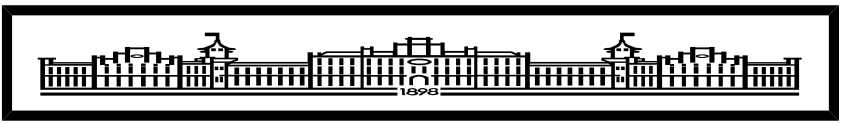
Національний технічний університет України

“Київський політехнічний інститут”

АЛГОРИТМИ І СТРУКТУРИ ДАНИХ

Методичні вказівки до виконання
лабораторних робіт
для студентів напрямку 6.050103
«Програмна інженерія»

Рекомендовано вченою радою факультету інформатики та обчислювальної техніки НТУУ «КПІ»



Київ 2012

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

Національний технічний університет України

“Київський політехнічний інститут”

АЛГОРИТМИ І СТРУКТУРИ ДАНИХ

Методичні вказівки до виконання
лабораторних робіт
для студентів напрямку 6.050103
«Програмна інженерія»

Затверджено

на засіданні кафедри обчислювальної техніки

ФІОТ НТУУ «КПІ»

Протокол № 11 від 12.05.12

Київ НТУУ «КПІ» 2012

УДК 681.3

ББК 0513-048р

П 797

Алгоритми і структури даних: Методичні вказівки до виконання лабораторних робіт для студ. напрямку підготовки 6.050103 «Програмна інженерія» / В.І.Пустоваров – К.: НТУУ «КПІ», 2012. – 40 с.

Гриф надано вченою радою ФІОТ
(протокол № 4 від 28 листопада 2011 р.)

Методичні вказівки вміщують завдання до виконання лабораторних робіт з курсу «Алгоритми і структури даних».

Наведені варіанти завдань. До кожної роботи надається необхідний теоретичний матеріал і перелік рекомендованої літератури.. Призначені для студентів напряму підготовки 6.050103 «Програмна інженерія».

Навчальне електронне видання

Укладачі: *Пустоваров Володимир Ілліч*, канд.техн.наук, доц.

Відповідальний
редактор: .*Стіренко С.Г. к.т.н., доцент*
Рецензент: *Марченко О.І.,* канд. техн. наук, доц.

За редакцією авторів

ЗМІСТ

ВСТУП

Курс «Алгоритми і структури даних» є модулем нормативної дисципліни підготовки фахівців рівня бакалавр з напрямку 6.050103 «Програмна інженерія». Призначений для вивчення методів та засобів програмування для систем них програм. Модуль вивчається у другому семестрі першого курсу, тому вважається, що студенти вже засвоїли дисципліну «Програмування», «Інженерія програмного забезпечення» і в достатній мірі володіють навичками створення програм за допомогою мов програмування Pascal та C/C++.

Цикл лабораторних робіт складається з восьмі робіт і призначений для покриття практичної частини кредитного модулю дисципліни «Алгоритми і структури даних».

Роботи дозволяють отримати практичні навики написання елементів програмного забезпечення з використанням сучасних мов Pascal та C/C++ та бібліотек програмування.

Методичні вказівки включають для кожної роботи:

- теоретичний матеріал, який необхідний їх виконання;
- питання для самостійної перевірки;
- посилання на список рекомендованих джерел.
- варіанти завдань для виконання кожної лабораторної роботи з циклу.

1. МЕТОДИЧНІ ВКАЗІВКИ ЩО ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

1.1 Мета циклу лабораторних робіт

Метою проведення лабораторних занять з кредитного модулю " Алгоритми і структури даних" є закріплення теоретичних знань, умінь и навиків, пов'язаних з розробкою програмного забезпечення, що базується на використанні механізму процесів. Виконання робіт пов'язано з розробкою паралельних програм, отриманні практичних навиків при роботі з мовами програмування типу Pascal та C/C++, а також використанні спеціальних бібліотек.

1.2 Зміст протоколу та оформлення лабораторних робіт

Протокол виконання кожної лабораторної роботи має містити:

- титульний лист;
- завдання на лабораторну роботу згідно варіанта;
- лістинг програми з коментаріями.
- результати експериментального виконання роботи;
- висновки по роботі (аналіз одержаних результатів).

Під час оформлення роботи слід звернути увагу на лістинг програми, в якому за допомогою відступів і коментарів відокремлювати основні частини програми, забезпечити легке читання лістинга. Програма повинна розпочинатися з заголовка, в якому треба вказати назву та варіант роботи, прозвище виконавця, дату, групу.

2. ЛАБОРАТОРНІ РОБОТИ

Цикл лабораторних робіт включає 8 робіт. Варіанти завдань видаються викладачем на підставі таблиць в описах робіт.

2.1 Лабораторна робота 1

Тема роботи: Створення та використання абстракцій для вхідних та вихідних даних задач

Мета роботи: Вивчення типів кодованих даних в програмах і конструкцій базової мови програмування для їх визначення. Вивчення перенумерованого типу та методики його кодування. Визначення кодування шляхом визначення окремих імен. Ознайомлення з комбінуванням різних полів в структурах даних. Програмування формування та відображення кодованих даних.

Принципи побудови професійних програм

Щоб раціонально створювати програми і програмні комплекси, важливо використовувати принципи **модульного програмування**, тобто оформлення основи програми у вигляді комплексів процедур, функцій або інших підпрограм, які утворюють **модулі широкого використання** або **бібліотечні модулі мов програмування**. Базовий текст прикладних або системних програм також будується у вигляді окремого **управляючого модуля програми** або **модуля тестування**, які є головними модулями програми і включають набір контрольних прикладів або закінчену програму автономного тестування.

До того ж модульне програмування складає технологічну основу об'єктно-орієнтованого програмування та проектування об'єктів, коли модулі формуються з окремих об'єктів або груп взаємозв'язаних однотипних, успадкованих або близьких об'єктів. В таких об'єктно-орієнтованих модулях визначається як структура класу об'єкта, так і набори методів (процедур і функцій) для обробки примірників даних цього класу. В деяких випадках об'єктно-орієнтовані модулі реалізують через структури в рамках мов C/C++ та Асемблер або записи в рамках мови Pascal, наприклад, технологічні програми організації графічних інтерфейсів в середовищі UNIX, але такі засоби не мають вбудованих механізмів контролю за інкапсуляцією на відміну від примірників сучасних об'єктів та їх прототипів.

Розробка програм починається з визначення математичних абстракцій та їх реалізації в термінах обраних мов програмування. Спочатку для математичних абстракцій і позначень доцільно визначити відповідні типи елементарних даних, найчастіше числові, текстові або кодовані. Для цілих числових даних та покажчиків або суттєвим елементом є розрядність, яка обмежує кількість елементів в агрегатах та їх доступність в адресному просторі. Для числових даних з плаваючою точкою істотними є розрядності мантиси та порядку, які повинні забезпечити обмеженні на абсолютні та відносні похибки в представленні даних та їх розрахунках.

При необхідності використання кодованих або структурованих даних сучасні мови програмування дозволяють ви-

користовувати типи даних користувача: перенумеровані типи, а також структуровані типи та об'єкти, яким в цій роботі приділено основну увагу.

Контрольні функції прикладних та системних програм звичайно будуються для налагодження окремих модулів програмного забезпечення.

Короткі теоретичні відомості

Перенумеровані типи звичайно утворюються для створення кодованих даних з іменованими значеннями. Наприклад, найпростіший перенумерований тип, що кодує значення кольору, звичайно об'являється у файлі заголовку наступним чином:

```
enum color{red, green, blue}; // опис кольору
```

Примірники, покажчики та масиви перенумерованого типа описуються у відповідному файлі реалізації, що реалізує функції для обробки даних цього типу:

```
enum color c1, c2, *pc, pa[7]; // опис даних кольорів
```

Прикладом визначення функції відображення даних перенумерованого типа може бути наступне:

```
// виведення одного значення перенумерованого типа
void prClr(enum color cl)
{char* sClr[3]={"blue", "green", "red"};
   printf("%s",sClr[cl]);
}
```

В загальному випадку дані перенумерованого типу формуються так, що для них автоматично утворюються всі операції ари-

фметичних відношень, тобто для них завжди визначається відношення порядку. Якщо об'ява даних перенумерованого типа містить примусову ініціалізацію окремих значень, то для проміжних значень доцільно відображати або порожній рядок, або текст «не визначено». Для кодування станів об'єкта більш різноманітними числовими значеннями доцільно використовувати рівнів препроцесорні твердження `#define` та комбінування окремих кодованих полів у структурах.

У форматі структур доцільно визначати ключові рядки та поля індексів, які використовуються як аргументи пошуку та впорядкування, та таблиці разом з відповідними методами, які реалізують типові функції для роботи зі структурними об'єктами. Для відображення 24-бітових кольорів на сучасних дисплеях можна використати структуру наступного вигляду

```
struct RGB // структура 24-бітових кольорів
{char _red;  // яскравість червоного
 char _green;// яскравість зеленого
 char _blue; // яскравість блакитного
 char _dummy};//
```

Оскільки поля структури вкладаються в 32 біти, для 32-бітових компіляторів її можна представити в бітовому форматі:

```
struct RGB // структура 24-бітових кольорів
{char _red:8;  // яскравість червоного: 8 біт
 char _green:8;// яскравість зеленого: 8 біт
 char _blue:8; // яскравість блакитного: 8 біт
 char _dummy:8};//
```

Рекомендації зі створення функцій відображення

Для настройки виведення таблиць згідно з варіантом треба підготувати потрібний виклик функції виведення рядка. Помилка може призвести до аварійного завершення програми. Головним для коректного відображення є додержання правил звертання до функції `printf`, яка виконує в мові С форматне виведення на пристрій відображення або в стандартний файл виведення stdout в формі

`printf(`*рядок формату*, *список аргументів виведення*`);`

Рядок формату задається або у вигляді константи-рядка, який включає символи коду ASCII з відповідною національною таблицею кодування або покажчик на масив знаків або символів. Кількість аргументів визначається кількістю специфікаторів перетворення даних виведення, наведених в таблиці 1.1, а тип даних, що виводиться, повинен відповідати типу специфікатора. При цьому відповідальність за кількість об'єктів виведення лягає на програміста і діагностика не видається, через що потрібно бути дуже уважним при написанні подібних викликів функції.

В рядку формату спеціальні символи можуть задаватися після знака '\', а власне специфікатори перетворення починаються зі знака "%". Для включення в текст рядка одного знака "%" його необхідно повторити. Специфікатор перетворення в загальному випадку має синтаксис:

%*[вирівнювання][ширина][префікс формату]*
специфікатор формату

***Таблиця* 1.1**

Формати для виведення даних

Специфікатор (остання літера)		Типи даних та відображення	Вирівнювання	Префікси формату
	Системачисленняі Знаки	**Цілі**		
c		Символ char	*[-]*	
s		Рядок *char	*[-] [+]* [ ]	*[l] [N] [F]*
d		**Цілі зі знаком** int, long	[-] [#]	[l]
x		**Цілі без знака** Шістнадцяткові,		
X		Те саме, великими		*[N] [F]*
o		Вісімкові		
u		Десяткові		
f e E g G	Формат подання	**З плаваючою точкою** Фіксована точка Експоненційний Те ж з великим E Найкоротший f | e Найкоротший f | E	*[-] [+] [#]* *[ ]*	*[.точність]* *[L]*
p		**Вказівники** У форматі адреси		

Умовні позначення:

Нахилені квадратні дужки *[]* – ознака факультативності або необов'язковості конструкції;

[-] – зміщення результату перетворення до лівої межі поля виведення;

[+] – виведення знака перед числом;

[ ] – пропуск замість плюса або виведення мінуса;

[#] – кодове подання з початковими або кінцевими нулями;

[l] – префікс довгого формату, який визначає подання даних, що перетворюються в більш довгих форматах;
ширина – визначає мінімальне число знаків або символів виведення.
точність – визначає число цифр після десяткової точки для форматів f, e і E або число значущих цифр для форматів g і G. Округлення виконується шляхом відкидання молодших розрядів. За умовчання приймається точність в шість десяткових цифр.

Ці префікси можуть використовуватися при роботі з рядками в форматі `%s` і `%p`, тому що звертання до рядків завжди реалізуються в формі вказівників.

При виконанні функції `printf` рядок формату передається в вихідний файл без змін до одержання першого символу `%`. Після чого виконується перетворення чергового аргументу у відповідності з поточним форматом, результат цього перетворення також розміщується в вихідному файлі.

Структура програми для виконання роботи

Для спрощення розв'язання задачі слід створити проект apLb1.dsp, який зберігається в робочому просторі apLb1.dsw в папці apLb1, яка зберігає шаблони прототипів прикладів для всіх лабораторних робіт циклу «Алгоритми і структури даних»:

- apLb1.cpp – модуль опису даних та викликів функцій для тестування функцій відображення типів.

Розділ визначень і декларацій основної частини програми тестування в модулі apLb1.cpp повинен включати визначення типів та окремих значень, ініціалізацію елементів даних, спеціальні примірники даних, що зберігають їх різні значення. Виконавча частина модуля apLb1.cpp проекту повинна включати циклічно по-

вторювані виклики функцій для перевірки та відображення різних варіантів вхідних даних так, щоб перевірити роботу функцій при типових та кінцевих значеннях з множини припустимих значень аргументів.

Завдання на роботу

Завдання на підготовку до роботи на комп'ютері

1. Визначити варіант завдання для основних і додаткової задачі за Табл.1.2.

2. Розробити структуру даних для елементу таблиці згідно з заданими варіантами способу представлення кольорів за Табл.1.2. Якщо за варіантом треба використати тип даних, який визначається програмістом (**`enum, struct, union`**), то цей тип необхідно визначити перед використанням. Перед початком заняття показати викладачу ***текст опису структури і необхідних типів даних функціональної частини поля***.

3. Створити одномодульний програмний проект apLb1 на мові С. Настроїти відповідні дані в програмному проекті.

4. Настроїти **програми відображення** з модуля apLb1 так, щоб вони давали можливість відтворювати потрібні типи даних. Перед початком заняття показати викладачу ***текст оператора виведення функціональних полів***.

5. Внести контрольні виклики функцій, які відображують дані кодованих типів, а також дозволяють перевірити коректність виконання програм.

Завдання на роботу на комп'ютері

6. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

7. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми.

8. Одержати результати виконання, проаналізувати їх і зробити висновки.

Порядок вибору варіанту:

За останньою цифрою номера залікової книжки або за порядковим номером студента в списку підгрупи визначте варіант завдання для задач за табл. 1.2. В таблиці визначається комбінація з трьох типів даних для визначення числових або перенумерованих характеристик декого об'єкта. Значення даних A і B відображують однакові характеристики різними типами даних Значення даних C визначає додатковий тип для управління відтворенням значень A і B.

Таблиця 1.2

Варіанти завдань для виконання програми ініціалізації та відображення даних

№ вар.	Дані для представлення типом внутрішнього подання A	Дані для кодування окремих значень B	Визначення коду за числовим номером C	Комбінація кодованих полів у **бітовій структурі**
1	Перенумерований тип кольорів спектра веселки	Кольори для відтворення 24-бітовими значеннями	Номер палітри кольорів	A+B+C
2	Перенумерований тип кольорів спектра веселки	Кольори для відтворення 24-бітовими значеннями	Номер палітри кольорів	B+A+C
3	Перенумерований тип кольорів спектра веселки	Кольори для відтворення 24-бітовими значеннями	Номер палітри кольорів	B+C+A
4	Перенумерований тип кольорів спектра веселки	Кольори для відтворення 24-бітовими значеннями	Номер палітри кольорів	A+C+B
5	Перенумерований тип кольорів спектра веселки	Кольори для відтворення 24-бітовими значеннями	Номер палітри кольорів	C+B+A

6	Перенумерований тип кольорів спектра веселки	Кольори для відтворення 24-бітовими значеннями	Номер палітри кольорів	C+A+B
7	Дані цілих типів	Значення до 16 бітів	Номер типу користувача	A+B+C
8	Дані цілих типів	Значення до 16 бітів	Номер типу користувача	B+A+C
9	Дані цілих типів	Значення до 16 бітів	Номер типу користувача	B+C+A
10	Дані цілих типів	Значення до 16 бітів	Номер типу користувача	A+C+B
11	Дані цілих типів	Значення до 16 бітів	Номер типу користувача	C+B+A
12	Дані цілих типів	Значення до 16 бітів	Номер типу користувача	C+A+B
13	Дані типів з плаваючою точкою	Значення до 16 бітів	Номер типу користувача	A+B+C
14	Дані типів з плаваючою точкою	Значення до 16 бітів	Номер типу користувача	B+A+C
15	Дані типів з плаваючою точкою	Значення до 16 бітів	Номер типу користувача	B+C+A
16	Дані типів з плаваючою точкою	Значення до 16 бітів	Номер типу користувача	A+C+B
17	Дані типів з плаваючою точкою	Значення до 16 бітів	Номер типу користувача	C+B+A
18	Дані типів з плаваючою точкою	Значення до 16 бітів	Номер типу користувача	C+A+B
19	Дані типів визначених користувачем	Значення до 32 бітів	Номер типу користувача	A+B+C
20	Дані типів визначених користувачем	Значення до 32 бітів	Номер типу користувача	B+A+C
21	Дані типів визначених користувачем	Значення до 32 бітів	Номер типу користувача	B+C+A
22	Дані типів визначених користувачем	Значення до 32 бітів	Номер типу користувача	A+C+B
23	Дані типів визначених користувачем	Значення до 32 бітів	Номер типу користувача	C+B+A
24	Дані типів визначених користувачем	Значення до 32 бітів	Номер типу користувача	C+A+B

Питання для самоперевірки

1. Дайте визначення основних типів даних, які визначаються користувачем.
2. Якими структурами даних або об’єктами визначаються кодовані дані?
3. Які типи даних використовуються для визначення кодів даних користувача?
4. Якими масивами можна програмно визначити формати даних користувача?
5. Якими операторами мов програмування організуються обробка даних перенумерованого типу?
6. В яких випадках доцільно використовувати бітові структури?
7. В яких випадках доцільно використовувати наперед задані числові значення кодів даних?
8. Які механізми використовують для визначення числових значень кодів даних?
9. В яких випадках доцільно використовувати комбінації перенумерованих та структурованих даних?

. 2.2 Лабораторна робота 2

Тема роботи: Визначення алгоритмів обробки задач за структурами і агрегатами типів вхідних та вихідних даних

Мета роботи: Вивчення способів побудови алгоритмів за структурами та форматами вхідних, вихідних і внутрішніх даних та їх реалізація конструкціями базової мови програмування відповідно спроектованому комплексу типів даних. Вивчення циклів обробки даних всіх типів розгалужень за числовими даси і даними перенумерованого типу. Визначення повторних дій для обробки рядків структурованих даних. Визначення обмежень при комбінуванні полів в рядках структур і підсумках кодованих даних.

Принцип відповідності структури алгоритмів їх цільовим даним та типам і структурам поточних агрегатів даних

В сучасному модульному програмному забезпеченні більшість програм будується через вкладені виклики підпрограм різного рівня. Звичайно ці підпрограми виконують логічно закінчений набір дій, що розв’язує часткову, але функціонально закінчену задачу. Цільові змінні таких підпрограм, як процедури і функції визначаються аналітичними виразами присвоювань відповідних змінних та операторами виведення:

- в головних програмах цільові змінні визначаються їх включенням до операторів виведення;

- для функцій внутрішньої обробки програм цільовими даними є результат, що формує функція та дані, що можуть змінюватися через аргументи-покажчики (однак такі дії найчастіше розглядаються як небажаний бічний ефект);
- для процедур і функцій відображення цільовими діями є дії зовнішніх пристроїв над аргументами викликів функцій, що відображуються на зовнішньому пристрої;
- для процедур внутрішньої обробки програм цільовими даними є вихідні або змінні аргументи та дані, що змінюються за аргументами-покажчиками.

Контрольні функції програмних систем звичайно будуються для налагодження окремого модуля програми та не утворюють даних, істотних для розв'язання задач.

Короткі теоретичні відомості

Хоча досі процес програмування розглядається як творчий, існують численні рекомендації з формального контролю і доведення правильності перетворень на етапі проектування програм, що складає основу формальної верифікації. Контроль результатів виконання програм на контрольних прикладах в процесі їх перевірки та експлуатації також частіше за все виконується шляхом формального зіставлення результатів рішення з очікуваними результатами та обмеженнями специфікацій, що використовує при тестуванні програм.

В процесі формалізованої побудови програм програміст одержує можливість аналізу власних перетворень в декількох

напрямках: створення абстракцій даних, визначення обмежень даних, визначення методів та побудова алгоритмів обчислень. Основи формалізації процесу програмування закладені в таких багатьох сучасних технологіях і стилях програмування, як структурне, модульне та об'єктно–орієнтоване програмування. Однак в сфері мистецтва досі залишаються методи побудови і підбору оптимальних та узагальнюючих методів, алгоритмів програм, а також багато дій з визначення критичних ділянок програм. Причиною цього залишаються слабкі темпи автоматизації таких високоінтелектуальних областей програмування, як формальна верифікація програм, цілеспрямовані еквівалентні перетворення, автоматизований синтез програм, тощо.

Вибір методу декомпозиції

Розробка програм за методом перетворення формульних аналітичних зв'язків математичної моделі предметної області для досягнення визначеної цілі задач інформаційних перетворень та декомпозиції задач на послідовність дій з послідовного наближення до кінцевої мети виконується у відповідності з наступною стандартною послідовністю дій:

1) визначити математичну модель предметної області та специфікацію задачі: аналітичні зв'язки між вхідними та вихідними (цільовими) даними в математичній формі, мету перетворень в програмі та вимоги до діапазону оброблюваних даних;

2) визначити типи та структури вхідних і вихідних даних та інформаційні ресурси (в мові Асемблера регістри або пам'ять, що

іменується користувачем) для їх збереження (що було розглянуто в попередній лабораторні роботі);

3) визначити способи кодування (які також розглянуто в попередній роботі) і внутрішнього представлення даних, а також необхідні обмеження на структури і розміри даних, що визначаються вимогами специфікацій до функціонування програми; якщо вбудовані ресурси обчислювальної системи не дозволяють задовольнити вимог, то треба проробити кропітку роботу з визначення таких структур і методів або знизити вимоги до характеристик ефективності розв'язання задачі;

4) визначити комплекс обробляючих або операційних ресурсів обчислювальної системи, доступних для розв'язання задач, починаючи з пакетів програм, включаючи стандартні процедури, функції і оператори мов програмування і операційної системи і закінчуючи машинними командами цільового процесора;

5) обрати як базовий найбільш потужний операційний ресурс, за допомогою якого можна хоча б частково розв'язати поставлену задачу або максимально наблизитися до поставленої мети;

6) визначити шляхом декомпозиції структуру алгоритму розв'язання головної задачі і виділити відмінності від застосованого для її розв'язання базового комп'ютерного ресурсу, які в подальшому будуть розглядатися як залишкові або окремі цілі розв'язання фрагментів задач;

7) рекурсивно довизначити нереалізовані блоки для обраного варіанта декомпозиції, використовуючи пункти цієї самої методики для менших блоків задачі.

Найбільш важливою частиною розглянутого процесу є визначення базових ресурсів, що реалізують елементи декомпозиції, та визначення параметрів цих елементів як фрагментів програм. Ресурси розв'язання задач та їх фрагментів визначаються на базі традиційних принципів структурного програмування і побудови алгоритмів програм шляхом низхідного аналізу задачі. Такі прості види декомпозиції, як декомпозиція на послідовні та паралельні блоки програм, потребують схожості окремих частин (підзадач) P або Q з наявними ресурсами розв'язання задач і мають в базовому варіанті дві структурні складові: блоки P і Q, показані на рисунку 2.1.

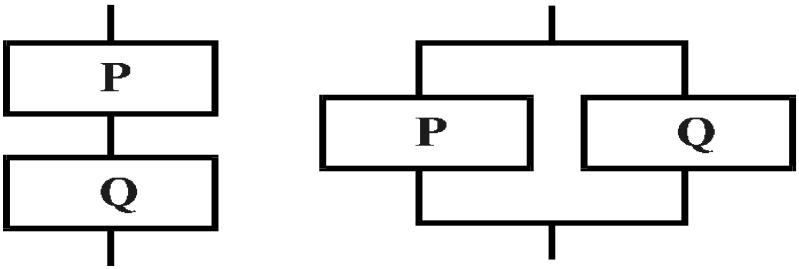


Рис. 2.1. Блок-схеми послідовних і паралельних блоків

Для визначення схожості треба мати доступ до бази даних з семантичними характеристиками ресурсів. Опис кожного ресурсу розв'язання задачі (програма, підпрограма, машинна або апаратно-

реалізована операція) в базі даних повинен включати списки або об'яви вхідних та вихідних даних разом з їх типами, а також описані формально-математично або неформально у текстовій формі зв'язки між вхідними та вихідними даними. Якщо поставлена задача відрізняється від задачі, розв'язуваної наявними ресурсами тільки за типом, то створення модифікованих ресурсів для розв'язання задачі потребує модифікації типів даних, яка на найпростіших прикладах розглядалась у попередній лабораторні роботі. Якщо ж виявляються відмінності у виразах для розв'язання задачі, то доводиться завершувати етап структурного синтезу блоків ресурсу розв'язання задач за наведеною вище методикою. ….

При програмуванні на мовах програмування високого рівня спектр ресурсів, що використовуються, доходить до рівня архітектури апаратних елементів системи і може охоплювати елементи процесорів та інтерфейсних пристроїв, доступних для виконання операцій обробки та обміну даними. Якщо вихідні дані кожного з блоків не є вхідними даними іншого блока, то декомпозиція розглядається як паралельна і може використовуватися для прискорення розв'язання задач при наявності апаратних ресурсів системи, що працюють паралельно.

Більш складні варіанти структурних блоків при декомпозиції обираються з множини традиційних операторів обраної базової мови програмування. Декомпозиція за допомогою умов, приклади якої показано на рисунку 2.2, на виникає як наслідок обмежень інтервалів значень та інтервалів коректності математичної моделі в

специфікації зв'язків предметної області. Блок умови контролю *C* визначається логічним зв'язком, перевірка якого здійснюється на мовах високого рівня умовними операторами, а на рівні машинних команд – командами умовної передачі управління. Умова *C* з верхнього прикладу рисунку 2.2 призводить до розгалуження програми на блок *P* – обробка за позитивним результатом перевірки умови, а блок *N* – обробка за негативним результатом перевірки. Використання групи команд перевірки умов блока *C* в сполученні з групою команд умовного переходу або використання команд непрямої передачі управління з індексацією призводить до множинного розгалуження умовного блока.

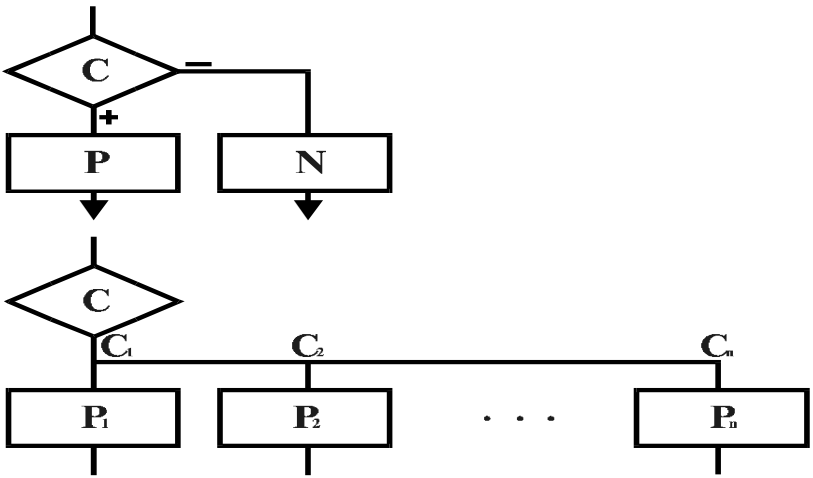


Рис. 2.2. Блок-схеми розгалужених блоків

Для багатьох варіантів декомпозиції з блоками умов базовими управляючими структурами є оператори **`if`** і **`case`** мов високо-

го рівня, які з одного боку можуть бути наслідком визначення діапазонів для типів даних користувача, а з іншого – наслідком застосування умовних структур даних типа **`record`** з використанням **`case`** в мові Pascal і **`union`** – в мові C. На рівні машинних команд if-декомпозиція включає перевірку умов шляхом обчислення вразів і аналізу ознак результатів, що легко реалізується командами умовних переходів, для яких тип використаної операції визначається типом оброблюваних даних. В поточній роботі такі види декомпозиції використовуються для перевірки обмежень вхідних та вихідних даних.

Декомпозиція за допомогою циклічних блоків, приклади яких наведено на рисунку 2.3, зв'язується або з повторюваними даними та ритмічними структурами даних задачі, або з рекурсивними аналітичними зв'язками. Основною класифікаційною ознакою циклів є організація перевірки умови закінчення циклу C_f. Розрізнюють типи циклів з передумовами і післяумовами довільної логічної складності, цикли, що управляються значеннями параметрів циклу, і цикли з контролем допустимого відхилення результатів, або цільових змінних та з використанням спеціальних індикаторів. Як параметри циклу найчастіше використовують позиції даних у файлах, індекси елементів даних в масивах або відносні адреси. Контроль закінчення циклів здійснюється за значеннями параметрів, або аналізом вмісту лічильників при наперед відомому числі циклів. При цьому важливо враховувати, що при перевірці після умови, тіло циклу виконується принаймні один раз.

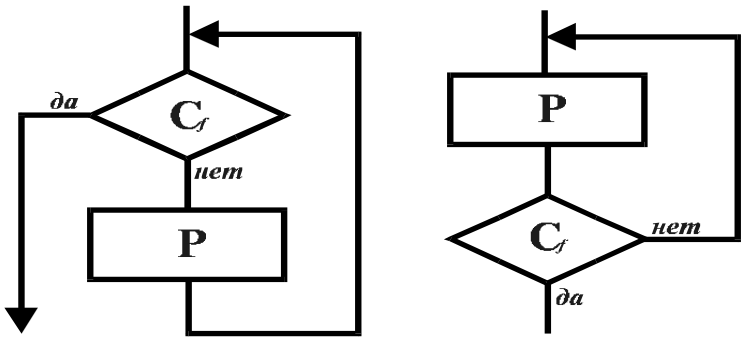


Рис. 2.3. Блок-схеми циклічно повторюваних блоків

Розвиток і розширення ресурсів програмування може виконуватися, як в низхідному процесі проектування програмного забезпечення внаслідок цілеспрямованої декомпозиції задач, так і внаслідок аналізу раніше розроблених програм при висхідному проектуванні шляхом накопичення все більш потужних програмних ресурсів, що дозволяють досягнути більш серйозних цілей. Третій напрямок розвитку програмних ресурсів можна визначити як вдосконалення або узагальнення програм для їх застосування до більш потужних і загальних структур і форматів даних. Розширені програмні ресурси організуються як фрагменти програм в формі макровизначень або як процедури, функції або задачі, що працюють з даними узагальнених форматів. Макровизначення, що визначають прообрази або прототипи фрагментів програм, при генерації макророзширень за макровикликом (макрокомандою) дають результати, близькі до функцій типа inline в мові С. При такому підході до побудови розширених ресурсів нема принципової

різниці у використанні процедурних мов високого рівня і машинних кодів.

.

Завдання на роботу

Завдання на підготовку до роботи на комп'ютері

1. Визначити варіант завдання для основних і додаткової задачі за Табл.2.1.

2. Розробити структуру даних для елементу таблиці згідно з варіантом за Табл.2.1. Якщо за варіантом треба використати тип даних, який визначається програмістом (**`enum`**, **`struct`**, **`union`**), то цей тип необхідно визначити перед використанням. Перед початком заняття показати викладачу ***текст опису структури і необхідних типів даних функціональної частини поля***.

3. Створити одномодульний програмний проект apLb1 на мові С. Настроїти відповідні дані в програмному проекті.

4. Настроїти програми відображення з модуля apLb1 так, щоб вони давали можливість відтворювати потрібні типи даних. Перед початком заняття показати викладачу ***текст оператора виведення функціональних полів***.

5. Внести контрольні виклики функцій, які відображують дані кодованих типів, а також дозволяють перевірити коректність виконання програм.

Завдання на роботу на комп'ютері

9. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

10. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми.

11. Одержати результати виконання, проаналізувати їх і зробити висновки.

Порядок вибору варіанту:

За останньою цифрою номера залікової книжки або за порядковим номером студента в списку підгрупи визначте варіант завдання для задач за табл. 2.1.

Таблиця 2.1

Варіанти завдань для виконання типових алгоритмів обробки

№ вар.	Обмеження для даних полів, занесених до таблиці	Визначення підсумків за стовпцями даних в таблиці	Оцінка обмежень безпеки	Контроль результатів обробки за квадратом значень
1	[3..5]	sum< max	Комбінації характеристик підсумків	За ступенем значень 1-го поля
2	[0..4]	min	Комбінації характеристик підсумків	За ступенем значення підсумку
3	[3..5]	max	Комбінації характеристик підсумків	За ступенем значень 1-го поля
4	[0..4]	avrg	Комбінації характеристик підсумків	За ступенем значення підсумку
5	[3..5]	nmb	Комбінації характеристик підсумків	За ступенем значень 1-го поля
6	[0..4]	disp	Комбінації характеристик підсумків	За ступенем значення підсумку
7	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	sum< max	Комбінації характеристик підсумків	За ступенем значень 1-го поля
8	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	min	Комбінації характеристик підсумків	За ступенем значення підсумку
9	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	max	Комбінації характеристик підсумків	За ступенем значень 1-го поля

10	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	avrg	Комбінації характеристик підсумків	За ступенем значення підсумку
11	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	nmb	Комбінації характеристик підсумків	За ступенем значень 1-го поля
12	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	disp	Комбінації характеристик підсумків	За ступенем значення підсумку
13	[3..5]	sum< max	За останньою характеристикою підсумків	За ступенем значень 1-го поля
14	[0..4]	min	Підсумки за першою характеристикою	За ступенем значення підсумку
15	[3..5]	max	За останньою характеристикою підсумків	За ступенем значень 1-го поля
16	[0..4]	avrg	За першою характеристикою підсумків	За ступенем значення підсумку
17	[3..5]	nmb	За останньою характеристикою підсумків	За ступенем значень 1-го поля
18	[0..4]	disp	За першою характеристикою підсумків	За ступенем значення підсумку
19	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	sum< max	За останньою характеристикою підсумків	За ступенем значень 1-го поля
20	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	min	За першою характеристикою підсумків	За ступенем значення підсумку
21	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	max	За останньою характеристикою підсумків	За ступенем значень 1-го поля
22	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	avrg	За першою характеристикою підсумків	За ступенем значення підсумку
23	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	nmb	За останньою характеристикою підсумків	За ступенем значень 1-го поля

24	$r<r_{max}$, $g<g_{max}$, $b<b_{max}$	disp	За першою характеристикою підсумків	За ступенем значення підсумку

Питання для самоперевірки

1. Дайте визначення основних типів ресурсів розв’язання задач в комп’ютерних системах.
2. Дайте визначення основних видів декомпозиції, які використовуються при побудові алгоритмів розв’язання задач.
3. Які типи ресурсів розв’язання задач може використовувати програміст при побудові алгоритму?
4. Як структури даних або об’єктами впливають на процес побудови алгоритму розв’язання задач?
5. В яких випадках доцільно використовувати декомпозицію на послідовні або паралельні блоки?
6. В яких випадках доцільно використовувати декомпозицію на умовні блоки?
7. В яких випадках доцільно використовувати декомпозицію на циклічні блоки?

2.3 Лабораторна робота 3

Тема роботи: **Модульне програмування та його використання для побудови програм обробки таблиць**

Мета роботи: Вивчення типів таблиць в системних програмах і конструкцій базової мови програмування для їх визначення. Пошук за прямою адресою. Основні типи залежностей та відношень, які реалізуються через пошук в таблицях системних програм. Лінійний та двійковий пошук. Вимоги до унікальності ключів.

Принципи побудови професійних системних програм

Щоб раціонально створювати програми і програмні комплекси, важливо використовувати принципи **модульного програмування**, тобто оформлення основи програми у вигляді комплексів процедур, функцій або інших підпрограм, які утворюють **модулі широкого використання** або **бібліотечні модулі мов програмування**. Базовий текст прикладних або системних програм також будується у вигляді окремого **управляючого модуля програми** або **модуля тестування**, які є головними модулями програми і включають набір контрольних прикладів або закінчену програму автономного тестування.

До того ж модульне програмування складає технологічну основу об'єктно-орієнтованого програмування та проектування об'єктів, коли модулі формуються з окремих об'єктів або груп взаємозв'язаних однотипних, успадкованих або близьких об'єктів. В та-

ких об’єктно-орієнтованих модулях визначається як структура класу об’єкта, так і набори методів (процедур і функцій) для обробки примірників даних цього класу. В деяких випадках об’єктно-орієнтовані модулі реалізують через структури в рамках мов C/C++ та Асемблер або записи в рамках мови Pascal, наприклад, технологічні програми організації графічних інтерфейсів в середовищі UNIX, але такі засоби не мають вбудованих механізмів контролю за інкапсуляцією на відміну від примірників сучасних об’єктів та їх прототипів.

До організації раціонального комплексу програмних модулів, яка дозволяє багаторазове використання в системних програмах, висуваються наступні вимоги:

- функціональні елементи або методи повинні групуватись так, щоб кожна група включала і використовувала тільки або по 1, або по 2, або масив з заданою кількістю базових об’єктів;
- кожен функціональний елемент або метод повинен виконувати логічно закінчену дію і може включати будь-яку кількість викликів інших функціональних елементів;
- для загальності підпрограм більшість даних повинні передаватись до них через аргументи у вигляді власне даних, вказівників або посилань на складні агрегати даних;
- доцільно, щоб структури даних ієрархічно об’єднували дані, що стосуються єдиної фізичної сутності, що дозволяє зменшити кількість аргументів в підпрограмах (крім того, це складає основу

формування об’єктно-орієнтованих даних в мові С без використання класів С++);

- кількість глобальних та статичних даних програм повинна бути мінімізовані тільки використанням лічильників об’єктів в групах та дійсно унікальних об’єктів програми;
- допоміжні підпрограми відображення для контролю результатів функціональних підпрограм повинні розміщуватися в окремому модулі відображення;
- контрольні приклади повинні розміщуватися в окремому тестовому модулі, де поряд з головною програмою або початковою функцією мови повинні знаходитися підпрограми генерації тестових послідовностей та зв’язок з діалоговими засобами формування даних контрольних прикладів.

Контрольні тестові проекти програм застосувань звичайно будуються для автономного налагодження окремих модулів, а випускні та інсталяційні редакції програм – для комплексного тестування програм і експлуатації для розв’язання задач користувача.

Короткі теоретичні відомості

Таблиці як двовимірні агрегати даних, які визначають функціональну залежність між окремими характеристиками, використовують в системних програмах та системах управління базами даних для відтворення даних при виконанні програм і маніпуляцій з базами даних.

Таблиці як базові структури і сховища накопичуваної інформації в програмах застосувань

Основу комп'ютерних методів обробки складають абстрактні моделі об'єктів, що змінюються в процесі розвитку або життєдіяльності систем досліджень, проектування та управління. В залежності від типу об'єктів використовуються аналітичні (алгебраїчні), алгоритмічні (директивні), реляційні (табличні) та мережні моделі, а також їх різноманітні комбінації. В цій роботі основну увагу приділено вивченню техніки структурного проектування таблиць і побудові ефективних програм процедурними, об'єктно-орієнтованими та машинно-орієнтованими засобами програмування. При такому підході програми інформаційного пошуку розглядаються на загально системному рівні реалізації з абстрагуванням від прагматики конкретного застосування. Тому для прагматичних ілюстрацій візьмемо задачі пошуку при трансляції кодів з вхідних мов програмування та запитів.

В більшості системних програм головною метою пошуку є визначення характеристик, пов'язаних з символічними позначеннями елементів вхідної мови (ключових слів, імен, ідентифікаторів, роздільників, констант, тощо). Ці елементи розглядаються як аргументи пошуку або асоціативні ознаки інформації позначень. Функціями табличних перетворень є збереження або використання характеристик для обчислень елементів внутрішнього подання та вихідних кодів. Пошук за асоціаціями виконується як у вхідній опера-

тивній інформації системної програми, так і в інформаційній базі (ІБ), де зберігаються систематизовані аргументи пошуку та їх впорядковані або систематизовані характеристики елементів кодів, що поєднуються в таблицю. Найбільш важливі задачі – це формування таблиць та швидкий оперативний пошук інформації в таблицях.

Особливості побудови сучасної ІБ демонструються типовими прикладами, які відрізняються роздільним зберіганням взірців характеристик об'єктів та їх структурних зв'язків. Такий підхід використовується для зберігання мультимеднійних картин та сцен у вигляді кодів, узагальнених на рівні мови, призначеної для відтворення структур з керованими типами взірців.

Основу визначення функціональних залежностей в системних програмах та інформаційних системах складають однозначні та багатозначні табличні зв'язки між показниками або атрибутами об'єктів системних програм та інформаційних систем. Таблиці є основою так званої реляційної моделі в базах даних [1].

Реалізація таблиць як об'єктів в рамках мови C/C++ вимагає визначення структури для рядка таблиці, в якій повинні зберігатися ключові і функціональні поля. Саму ж таблицю варто визначати як об'єкт класу, який включає рядки таблиці як динамічну складову у формі масиву структур. При додержанні вимог модульного програмування в проекті на мові C/C++ створюються файли заголовків, в яких зберігаються прототипи рядка таблиці, конструкторів, деструкторів та базових функцій або операцій, а також відповідний файл реалізації, приклади яких наведені в наступних параграфах. У

відповідності з традиціями програмування роботи з таблицями реляційних баз даних [3] базовими операціями є: `select` – вибірка даних з таблиці; `insert` – додавання рядків даних до таблиці; `delete` – вилучення рядків даних з таблиці; `update` – заміна даних в рядку таблиці.

Найпростіший спосіб побудови ІБ полягає у визначенні структури окремих елементів, що вбудовуються в структуру таблиці. Рядки таблиці в загальному випадку складаються з полів двох типів елементів – аргументів та функціональних характеристик. Така класифікація умовна та визначається цільовими характеристиками предметної галузі та окремих задач. Як вже відзначалося, аргументом пошуку в загальному випадку можна використовувати декілька полів, але спочатку для спрощення задачі визначимо такі правила:

- аргумент пошуку подається одним полем структурного типу та пов'язаною з ним функцією порівняння, що дозволяє перекласти труднощі, викликані множиною значень та функціональними комбінаціями аргументів на методи порівняння;
- функціональна частина запису також зібрана в одному полі структурного типу;
- елемент, що аналізується або прототип пошуку, має таку ж або близьку структуру, як і елемент таблиці.

Ці припущення практично не знижують загальності аналізу та побудованих в результаті алгоритмів, але істотно спрощують програмування методів маніпуляції з таблицями. Кожний елемент

звичайно зберігає декілька (m) характеристик і займає в пам'яті послідовність адресованих байтів. Якщо елемент займає k байтів і треба зберігати N елементів, то необхідно мати хоча б kN байтів пам'яті. Розмістити інформацію можна декількома способами [3]:

1. Всі елементи розмістити в kN послідовних байтах і побудувати таблицю з N елементів у вигляді масиву. Приклад такої таблиці наведено нижче, де елементи задаються структурою `struct` в мові C, записами `record` в мові Pascal, довжиною k байтів, що визначається сумою розмірів ключової та функціональної частини елемента таблиці.
2. Побудувати m таблиць у вигляді масивів, скажімо, T_1, T_2,... T_m, для кожної з m характеристик. При цьому i-й елемент таблиці буде розподілено по елементах масивів T_{1i}, T_{2i},..., T_{mi}. Цікаво відзначити, що при роботі коротких аргументів пошуку довжиною 1, 2 та 4 (для 32-розрядного режиму) байти процес обробки можна істотно прискорити, використовуючи машинні команди роботи з ланцюжками або рядками даних.
3. Розділити таблицю на l блоків, сегментів або підтаблиць. В граничному випадку можна одержати сукупність блоків, в яких зберігаються по одному елементу. В таку таблицю необхідно додати покажчики на зв'язки у вигляді адрес пов'язаних елементів і, таким чином, організувати зв'язані списки або впорядковані деревоподібні структури.

При використанні мов високого рівня доцільно спочатку визначити структуру окремого запису або рядка таблиці за способами

1 і 3 як сукупності полів, для способу 2 – як контейнери окремих полів.

У форматі структур і класів можливо зберігати будь-які комплекси даних. У подальших прикладах цього параграфу визначаються тільки прототипи, для яких можна визначити різні реалізації методів залежно від обраних підходів та алгоритмів доступу даних та маніпуляцій даними. Формат структур доцільно використати для окремого групування функціональних і ключових полів, до яких здійснюється доступ за іменами. Структури ключових і функціональних полів таблиць в рамках мови C/C++ можуть бути спочатку оголошені, а потім детально визначені як структури або класи.

```
struct keyStr;
            // структура ключової частини таблиці
struct fStr;
          //структура функціональних полів таблиці
```

У форматі структур доцільно визначати ключові рядки та поля індексів, які використовуються як аргументи пошуку та впорядкування, та таблиці разом з відповідними методами, які реалізують типові функції для роботи зі структурними об’єктами. При визначенні розширених структур більш високих рівнів до них включаються простіші структури, як в **контейнери**.

Серед широкої множини варіантів побудови таблиць оберемо для прикладів логічно найпростішу організацію записів реляційних таблиць у вигляді структур та додаткових змінних, які визначають поточні характеристики таблиці. Файл заголовків `"tables.h"`

для створення таблиць в мові C/C++ може бути подібним до наступного прикладу, причому імена власних даних та їх типів програміст, як завжди, може зручно задавати за власними міркуваннями:

```
struct keyStr // ключова частина запису
{char* str;  // ключові поля
 int  nMod;};// (уточнюється за варіантом)
struct fStr;// функціональна частина запису
{long double _f;};
//f-поле (уточнюється за завданням)
struct recrd // структура рядка таблиці
{struct keyStr key;// примірник структури ключа
 struct fStr func;
// примірник функціональної частини
 char _del;};// ознака вилучення
// обробка таблиць за прямою адресою
// вибірка за прямою адресою
struct recrd* selNmb(struct recrd*, int nElm);
// включення за прямою адресою
struct recrd* insNmb(struct recrd*pElm,
       struct recrd*tb, int nElm, int*pQnElm);
// вилучення за прямою адресою
struct recrd* delNmb(struct recrd*, int nElm);
// корекція за прямою адресою
struct recrd* updNmb(struct recrd *pElm,
       struct recrd*tb, int nElm, int*pQnElm);
// порівняння рядків за відношенням порядку
int cmpStr(unsigned char* s1, unsigned char* s2);
```

```
// порівняння ключів за відношенням нерівності
int neqKey(struct recrd*, struct keyStr);
// порівняння ключів за відношенням порядку
int cmpKey(struct recrd*, struct keyStr);
// порівняння за відношенням схожості
int simKey(struct recrd*, struct keyStr);
// вибірка за лінійним пошуком
struct recrd* selLin(struct keyStr kArg,
        struct recrd*tb, int ln);
// вибірка за двійковим пошуком
struct recrd*selBin(struct keyStr kArg,
                    struct recrd*tb, int ln);
```

Прикладом функції пошуку за прямою адресою у файлі реалізації до файлів шаблону включено наступну функцію:

```
// вибірка за прямою адресою
struct recrd* selNmb(struct recrd* tb, int nElm)
{return &tb[nElm];
}
```

а прикладами функцій порівняння

```
// порівняння рядків за відношенням порядку
int neqKey(struct recrd* el, struct keyStr kArg)
// порівняння за відношенням нерівності
{return (strcmp(el->key.str, kArg.str)||
        el->key.nMod != kArg.nMod);
}
// порівняння структур за відношенням порядку
int cmpStr(unsigned char* s1, unsigned char* s2)
{unsigned n;
```

```
 while(s1[n]==s2[n]&&s1[n]!=0)n++;
 return s1[n]-s2[n];}
int cmpKey(struct recrd* el, struct keyStr kArg)
{ int i=cmpStr((unsigned char*)el->key.str,
            (unsigned char*)kArg.str);
 if(i)return i;
 return el->key.nMod - kArg.nMod;
}
// вибірка за лінійним пошуком
struct recrd*selLin(struct keyStr kArg,
                    struct recrd*tb,int ln)
{while(--ln>=0&&cmpKey(&tb[ln], kArg));
 if(ln<0)return 0;
 return &tb[ln];
}
// вибірка за двійковим пошуком
struct recrd*selBin
(struct keyStr kArg,// ключ аргументу пошуку
 struct recrd*tb,   // адреса початку таблиці
 int ln)     // кількість елементів таблиці
{int i, nD=-1, nU=ln, n=(nD+nU)>>1;
 while(i=cmpKey(&tb[n],kArg))
    {if(i>0)nU=n; else nD=n;
     n=(nD+nU)>>1;
     if(n==nD)return NULL;
    }
 return &tb[n];
}
// виведення рядка базової таблиці
```

```
void prRow(struct recrd* rw)
{if(rw==0)printf("is absent\n");
 else printf("%10s %3u %5.3f\n",
// Останній шаблон рядка формату і відповідне
// поле списку виведення даних необхідно
// узгодити з варіантом завдання
      rw->key.str,rw->key.nMod,rw->func._f);
}
```

При використанні об'єктно-орієнтованого програмування на базі класів C++ об'єкти можуть бути побудовані функціями передачі відповідних повідомлень ізоморфно до об'єднань структур. В цьому випадку відповідні функції включаються до класів як методи, зв'язані з базовим об'єктом, який стає досяжним без передачі параметрів, що спрощує описи методів-функцій. Базовий клас можна розширювати через успадковані класи, включаючи до нього додаткові ключові поля будь-яким чином. За необхідності використання класів доцільно задавати декларації класів з визначенням повних характеристик об'єкта таблиці з додаванням записів у вигляді примірників класу або запису.

Основні типи відношень в процедурах пошуку

Відношення рівності та нерівності

При пошуку за прямою адресою та лінійному пошуку в невпорядкованих таблицях перевіряються відношення рівності або нерівності для вибірки потрібних записів та елементів. Операції

відношення мови C/C++ "=" та "!=" створюються за умовчанням для всіх визначених типів даних порівняння всіх відповідних пар полів основної структури. Однак для типів даних з динамічними складовими нерівність полів вказівників ще не свідчить про нерівність даних, що знаходяться за покажчиками. Тому операції "=" та "!=" для типів даних з динамічними складовими необхідно перевизначити. Стандартна функція ANSI C `strcmp` перевіряє саме відношення нерівності рядків і може успішно використовуватись в різних варіантах лінійного пошуку.

Відношення порядку

Впорядкування таблиць за ключовими полями стає можливим лише у випадку впорядкування кодів кожного з потрібних ключових полів. Відношення порядку встановлюються для даних з одновимірними множинами визначення значень (доменами). Всі числові і перенумеровані типи даних, в тому числі і полів, мають визначені відношення порядку. Тому набори операцій мови C/C++ "=" та "!=", які створюються за умовчанням, необхідно розширити додатковими операціями відношень мови C "<", "<=", ">=" та ">", в тому числі з урахуванням полів з вказівниками.

Для визначення відношення порядку багатокомпонентних типів необхідно, щоб кожний компонент мав відношення порядку, і щоб узагальнююче відношення будувалося за допомогою монотонних функцій. Відношення порядку завжди визначаються при побудові індексів в реляційних системах управління базами даних (СУБД), наприклад, в більшості SQL-серверів як зважене об’єднан-

ня полів, а в деяких системах, наприклад у FoxPro, припустимий індексний вираз, що визначає функцію полів.

Методи двійкового пошуку можна використовувати лише у впорядкованих таблицях або у відповідних індексах. Тобто таблиці для виконання такого пошуку повинні бути попередньо відсортовані і для роботи з ними треба створити функцію сортування.

Відношення близькості

При пошуку помилково підготовлених ключів в текстових редакторах та процесорах часто виникає потреба в визначенні схожості ключів пошуку. Такі дії часто виконуються в текстовому процесорі MS Word. Вони можуть будуватися на підрахунку кількості однакових n_e, схожих літер n_{si} за i-м типом схожості, а також літер, які не мають відповідника в іншому ключі і можуть спиратися на абсолютні і відносні формульні критерії схожості. Схожість літер може визначатися залежно від випадку аналізу за схожістю написання літер в різних алфавітах n_{s1}, за близькістю комп'ютерних кодів n_{s2} та за близькістю розташування на клавіатурі n_{s3}, а також з урахуванням кількості літер n_{s4}, які не мають відповідників в обох ключах.

При створенні програм порівняння за мірою близькості треба побудувати загальний критерій близькості як монотонну функцію $f(n_{s1}, n_{s2}, n_{s3}, n_{s4})$ в одному напрямку від n_{s1}, n_{s2} і n_{s3} та в іншому напрямку від n_{s4}. Крім того, попередньо необхідно організувати підрахунок n_{s1}, n_{s2}, n_{s3} і n_{s4}, при перегляді порівнюваних ключів. Результат пошуку за таким критерієм може бути неоднозначним,

навіть за умови вимоги однозначності ключів. На алгоритм лінійного пошуку це практичного не впливає, а у випадку базового двійкового пошуку доцільно виконати додатковий пошук навколо найближчого ключа, знайденого за відношенням порядку.

Рекомендації з вибору алгоритму оцінки міри близькості за відношенням близькості полягають в тому, що найбільш повну і точну оцінку міри близькості можна одержати просуваючись за алгоритмами, в яких організуються вкладені цикли. В таких циклах симетрично визначається міра близькості між двома порівнюваними ключами, шляхом підрахунку однакових та/або різних символів і наступного підрахунку за формулами абсолютної або відносної міри близькості (відносно загальної довжини імен).

Вимога унікальності ключів і аргументів пошуку

В системних програмах звичайно додержуються вимог унікальності ключів, і порушення такої вимоги розглядається як помилка повторного опису. Таких помилок і порушень вимоги унікальності можна уникнути, додаючи до запису таблиці додаткове ключове поле позиції розміщення визначення ключа. Але в реляційних СУБД повторювані значення ключів часто використовуються при зв'язуванні таблиць, і поля з однаковими значеннями ключів можна впорядковувати довільним чином. В цьому випадку важливо мати набір функцій, які дозволятимуть формувати: 1) перший ліпший результат пошуку; 2) черговий результат пошуку при черговому запиті; 3) всі можливі результати пошуку при одному виклику.

Структура програмного шаблона виконання роботи

Для спрощення розв'язання задачі слід використовувати прототип проекту spLb1.dsp, який зберігається в робочому просторі spLb1.dsw в папці spLb1, яка зберігає шаблони прототипів для всіх лабораторних робіт циклу «Алгоритми і структури даних». До складу проекту консольної прикладної програми, орієнтованої на можливість використання в числі інших бібліотек стандартних функцій та комплексу об'єктів MFC (Microsoft Foundation Classes) входять модулі з наступними вхідними файлами заголовків і реалізацій:

- StdAfx.h і StdAfx.cpp – модуль для організації використання об'єктів класів MFC, створених поза стандартами ANSI;
- tables.h і tables.cpp – модуль для опису і організації використання структур або класів таблиць;
- vistab.h і vistab.cpp – модуль для відображення рядків або повних таблиць;
- spLb1.cpp – модуль контрольних прикладів для тестування функцій обробки таблиць.

При побудові проектів без функцій MFC та в інших системах, що дозволяють побудову проектів, файли StdAfx.h і StdAfx.cpp можуть бути опущені, а посилання на них в операторах `#include "StdAfx.h"` повинні бути виключені.

Розділ визначень і декларацій основної частини програми тестування в модулі spLb1.cpp повинен включати визначення таб-

лиць, ініціалізацію їх елементів, спеціальні змінні, що зберігають довжину (тобто кількість елементів) таблиці. Виконавча частина модуля spLb1.cpp програми повинна включати циклічно повторювані виклики функцій для перевірки та відображення різних варіантів вхідних даних так, щоб перевірити роботу функцій при типових та кінцевих значення з множини припустимих значень аргументів.

Рекомендації з модифікації і розширення функцій модулів

Для настройки виведення таблиць згідно з варіантом треба внести зміни до функції виведення рядка з потрібним форматом.

При побудові функцій вставки, вилучення і корекції таблиць для двійкового пошуку необхідне попереднє розпізнавання успішності пошуку. Для цього зручно використати функцію двійкового пошуку `struct recrd*selBin(struct keyStr kArg, struct recrd *tb, int ln)`, яка повертає або адресу запису зі знайденим ключем, або стандартне значення невизначеного вказівника `NULL`. Для вилучення і корекції відповідні зміни можна виконати за адресою, яку повертає функція, а для вставки нового елемента треба змістити останні елементи таблиці до місця вставки в операторі циклу, а потім занести новий елемент до таблиці.

Завдання на роботу

Завдання на підготовку до роботи на комп'ютері

1. Визначити варіант завдання для основних і додаткової задачі за Табл.3.2.

2. Розробити структуру даних для елементу таблиці згідно з варіантом за Табл.3.2. Якщо за варіантом треба використати тип даних, який визначається програмістом (**`enum`**, **`struct`**, **`union`**), то цей тип необхідно визначити перед використанням. Перед початком заняття показати викладачу ***текст опису структури і необхідних типів даних функціональної частини поля***.

3. Ознайомитись з шаблоном програмного проекту spLb1. Настроїти відповідні дані в програмному проекті на мові С.

4. Настроїти функції пошуку за прямою адресою, а також лінійного та двійкового пошуку, відмітки про вилучення, упакування таблиці, впорядкування таблиці і вставку до таблиці за значенням ключових полів з використанням різних методів.

- для таблиці з доступом за прямою адресою налагодження виконати в файлах tables.h, table.cpp і vistab.cpp;
- для таблиці з доступом за ключовим полем або групою полів за методом лінійного пошуку – налагодження виконати в файлах tables.h, table.cpp і vistab.cpp;

5. Настроїти програми відображення з модуля vistab.cpp так, щоб вони давали можливість відтворювати потрібні типи даних. Перед початком заняття показати викладачу ***текст оператора виведення функціональних полів***.

6. Скласти алгоритми функцій вставки, вилучення і корекції в таблицях на базі методу двійкового пошуку і підготувати програмні модулі, які забезпечують вставку, вилучення і корекцію елемента таблиці за значенням ключового елемента: для таблиці з до-

ступом за ключовим полем або групою полів за методом двійкового пошуку) – заготовки для функцій `insBin`, `delBin` і `updBin` в кінці модуля tables.cpp.

8. Підготувати програмний модуль контрольної задачі, який виконує задані варіанти програм пошуку і вставки в таблицю за значенням ключового елемента, а також додаткових задач і дозволяє перевірити коректність виконання програм.

Завдання на роботу на комп'ютері

9. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

10. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми.

11. Одержати результати виконання, проаналізувати їх і зробити висновки.

Порядок вибору варіанту:

За останньою цифрою номера залікової книжки або за порядковим номером студента в списку підгрупи визначте варіант завдання для задач за табл. 3.1.

Таблиця **3.1**

Варіанти завдань для виконання пошуку

№ вар.	Тип ключа для прямої адреси	Тип ключа для інших видів пошуку	Тип функціонального поля	Тип вибірки
1	**unsigned char**	**char*_ unsigned short**	**float**	Перший
2	**unsigned char**	**char*_ unsigned int**	**double**	Всі
3	**unsigned char**	**char*_ unsigned long**	**struct**	Черговий
4	**unsigned char**	**char*_ unsigned char**	**union**	Перший
5	**unsigned char**	**char*_ unsigned short**	**enum**	Всі
6	**unsigned short**	**char*_ unsigned int**	**float**	Черговий
7	**unsigned short**	**char*_ unsigned long**	**double**	Перший
8	**unsigned short**	**char*_ unsigned char**	**struct**	Всі
9	**unsigned short**	**char*_ unsigned short**	**union**	Черговий
10	**unsigned short**	**char*_ unsigned int**	**enum**	Перший
11	**unsigned int**	**char*_ unsigned long**	**float**	Всі
12	**unsigned int**	**char*_ unsigned char**	**double**	Черговий
13	**unsigned int**	**char*_ unsigned short**	**struct**	Перший
14	**unsigned int**	**char*_ unsigned int**	**union**	Всі
15	**unsigned int**	**char*_ unsigned long**	**enum**	Черговий
16	**unsigned long**	**char*_ unsigned char**	**float**	Перший
17	**unsigned long**	**char*_ unsigned short**	**double**	Всі
18	**unsigned long**	**char*_ unsigned int**	**struct**	Перший
19	**unsigned long**	**char*_ unsigned long**	**union**	Всі
20	**unsigned long**	**char*_ unsigned char**	**enum**	Черговий

Варіанти типу вибірки означають:

- “Перший” – складання програми пошуку першого елементу поля з унікальними значеннями або з повтореннями значень;
- “Всі” – складання програми багатоваріантного пошуку всіх можливих результатів для ключа з повтореннями значень;
- “Черговий” – складання програми пошуку чергового варіанта результату для ключового поля з повтореннями значень.

Питання для самостійної перевірки

1. Як в мовах Pascal та C можна реалізувати елементи таблиці та поєднати їх в повноцінні таблиці?
2. Яку роль грають ключові поля записів таблиць?
3. Яку роль грають функціональні поля записів таблиць?
4. Порівняйте основні методи пошуку за швидкістю?
5. В яких випадках доцільно використання пошуку за прямою адресою.
6. В яких випадках доцільно використання лінійного пошуку.
7. В яких випадках доцільно використання двійкового пошуку.

2.4 Лабораторна робота 4

Тема роботи: Динамічне створення таблиць та їх обробка з використанням індексів

Мета роботи: Вивчення методів динамічного створення і розширення таблиць. Вивчення типів індексів для прискореного доступу до даних таблиць в системних програмах і конструкцій базової мови програмування для їх визначення. Вивчення деревоподібних структур таблиць та індексів і їх використання для прискорення пошуку в таблицях. Вивчення методів hash-пошуку.

Короткі теоретичні відомості

Для того щоб надати можливість використання системних програм в умовах різних обсягів пам'яті комп'ютера, часто під таблиці виділяють сховище окремими сегментами за аналогією з виділенням дискового простору для файлів. Потім початкові сегменти розширюють додатковими сегментами, коли базові сегменти виявляються вичерпаними. Для наближення до більш коректного використання об'єктно-орієнтованого доцільно інкапсулювати в одній структурі або в одному класі об'єкта як інформаційні елементи, так і допоміжні елементи таблиць, які зберігають довжину і вказівники на початок інформаційних сегментів.

Основні способи побудови динамічно розширюваних таблиць

Найпростіший спосіб побудови таблиці з динамічними елементами полягає у використанні функцій динамічного виділення пам'яті. Тоді кожний сегмент таблиці повинен визначатися струк-

турою, яка зберігає виділену і використану кількість елементів і вказівник на наступний сегмент або інший механізм об'єднання сегментів типу FAT в файлових системах. А власне таблиця буде визначитися структурою, в якій поля будуть визначати ідентифікацію таблиці, кількість сегментів таблиці, граничну кількість елементів в кожному сегменті та механізм поєднання сегментів.

```
struct sgTbStr // структура сегмента
{int   nFlEl;   //  кількість використаних елементів
 struct sgTbStr*pNxtSg;//адреса наступного сегмента
 struct recrd* pRcPtr; //вказівник на блок записів
};
struct hdTbStr // структура динамічної таблиці
{int   nSgLm;// гранична кількість елементів сегмента
 int   nSgBg; // довжина початкового виділення
 int   nSgSc; // довжина повторного виділення
 struct sgTbStr* pNxtSg;//адреса початкового сегмента
};
```

Тоді для створення та розширення окремих таблиць, необхідно визначити функції, що створюють crDnTb та розширюють таблиці `extDnTb1`.

```
struct hdTbStr*crDnTb( //адреса початкового сегмента
       int nLm, int nBg, int nSc)
{struct hdTbStr*p = (struct hdTbStr*)// заголовок
       malloc(sizeof(struct hdTbStr));
 p->nSgLm = nLm;
    // в шаблоні виділяється лише початковий сегмент
 p->nSgBg = 1;
```

```
 p->nSgSc = nSc;
 p->frstSg = (struct sgTbStr*)// сегмент
       malloc(sizeof(struct sgTbStr));
 p->frstSg->nRsEl = nLm*nBg;
 p->frstSg->nFlEl = 0;
 p->frstSg->pNxtSg = NULL;
 p->frstSg->pRcPtr = (struct recrd*)//  записи
   malloc(p->frstSg->nRsEl*sizeof(struct recrd));
 return p;
}
void  extDnTb1(struct hdTbStr*p)//початковий сегмент
{struct sgTbStr*ps = p->pNxtSg;
 while(ps)ps=ps->pNxtSg;
 ps->pNxtSg = (struct sgTbStr*)// сегмент
          malloc(sizeof(struct sgTbStr));
 ps->nRsEl = p->nSgLm*p->nSgSc;
 ps->nFlEl = 0;
 ps->pNxtSg = NULL;
 ps->pRcPtr = (struct recrd*)// записи
       malloc(ps->nRsEl*sizeof(struct recrd));
}
```

Функція (метод) `crDnTb` створює динамічну таблицю з виділенням сегментів первинного резервування, а метод `extDnTb1` розширює динамічну таблицю на один сегмент. До цих методів доцільно додати методи імпорту даних до таблиць з заповненням з початку та додаванням елементів (методи `impDnTb` і `insDnTb` в шаблоні програми).

Індекси як базові структури впорядкування даних, накопичених в інформаційних таблицях

Різні підходи до пошуку спираються на ті елементарні операції, які закладені у фізичну структуру відповідних файлів. Такими операціями можуть бути.

1. Знайти запис за його адресою
2. Знайти запис, наступний за даним.
3. Знайти запис перед даним.
4. Знайти перший запис файлу
5. Знайти останній запис файлу

Ці операції залежать від інформації про абсолютне або відносне положення записів. Ця інформація зберігається у рамках фізичної організації файлу і не має ніякого відношення до вмісту записів. Однак практичний інтерес представляє перш за все пошук на основі вмісту полів запису, тобто *асоціативний пошук.* Наприклад, пошук у файлі співробітників напевне повинен бути за прізвищем, іменем та по-батькові співробітника. Саме за цією інформацією може бути потрібним знайти іншу інформацію, таку як вік, місце роботи тощо. При використанні лише базового набору операцій для організації такого пошуку може виникнути потреба у послідовному перегляді в середньому половини записів, а у найгіршому випадку усього масиву або файлу.

Для ефективної реалізації асоціативного пошуку використовуються спеціальні структури даних, які називаються **індексами.** Загальну ідею індексу можна продемонструвати на

прикладі предметного покажчика вкінці книги. Предметний покажчик представляю собою список термінів у алфавітному порядку з зазначенням сторінок, на яких певний термін зустрічається.

Пошук такого терміна власне у книзі, по суті, складається з систематичного перегляду усієї книги (якщо потрібно знайти всі входження даного терміна). У покажчику термін можна швидко знайти, використовуючи алфавітний порядок і моментально визначити сторінки, де цей термін зустрічається. Щоб організувати індекс потрібно вирішити, за значенням яких полів буде відбуватися пошук, тобто **ключів пошуку.** Зазвичай індекс реалізується у вигляді окремого файлу, кожний запис якого вміщає значення ключа пошуку і вказівник (адреса) на відповідний запис в основному файлі. Індекс організовується так, що, знаючи значення полів можна швидко знайти відповідний запис індексу, а потім, використовуючи вказівник на основний файл – сам запис. Якщо в різних випадках пошук проходить по різним наборам полів, то для одного основного файлу можна мати декілька різних індексів.

Існує два основних і у деякому випадку протилежних підходу до організації індексу: впорядкування і хешування. Відповідно можна говорити про *впорядковані індекси та хешовані індекси.*

У впорядкованому індексі записи підтримується у порядку зростання або зменшення значень ключа. При додаванні, зміні або видаленні записів з основного файлу СУБД автоматично реорганізує індекс так, щоб він відповідав поточному вмісту основного файлу

Впорядкованість записів дозволяє використовувати швидкий двійковий пошук для знаходження запису за заданим ключем. Крім того впорядкований індекс дозволяє ефективно знаходити записи, ключі пошуку яких знаходяться у заданому діапазоні, тобто відповідати на питання типу: при заданих значеннях k_1 і k_2 знайти всі записи з ключем k таким, що $k_1 < k < k_2$.

Впорядкований індекс має на увазі можливість порівняння значень ключа. Якщо значення ключа є цілими або дійсними числами, то для них визначені загальні відношення порядку. Для символьних рядків визначений словниковий, а бо лексикографічний порядок. Нехай $a_1a_2... a_m$ і $b_1b_2... b_m$ два символьних рядки, де a_i і b_j - окремі символи. Тоді буде $a_1a_2... a_m < b_1b_2... b_m$ тоді і тільки тоді, коли $m <$ n і $a_i = b_j$ для $i \in [1, m]$ або існує таке $i \in [1, \min(m, n)]$, що $a_i = b_j$ для всіх $j \in [1, i - 1)]$ і $a_i < b_j$.

При цьому мається на увазі звичайний алфавітний порядок символів. Значення ключа, який складається більш ніж з одного поля, можна сортувати у порядку слідування полів. В результаті сортування по першому полю утворюються групи записів з одним значення у цьому полі. Після сортування кожної групи по значенню другого поля, отримуємо групи з одними й тими же значеннями у двох полях.

Отримані групі сортуються по третьому полю тощо. Потрібно відмітити, що таке впорядкування аналогічне лексико-графічному порядку, де в якості символів виступають значення полів.

В хешованому індексі записи розподіляються по спеціальній таблиці на основі перетвореного ключа. Перетворення ключа виконується деякою хеш-функцією. Вирахувавши значення хеш-функції для якого-небудь ключа, можна швидко визначити положення одиночних записів, але не дають ніяких переваг при пошуку всередині діапазону, оскільки записи не впорядковані

Індекси як двовимірні агрегати даних, які впорядковують таблиці і дозволяють насамперед підвищувати швидкість вибірки з реляційних таблиць даних, широко використовують в системах управління базами даних з різних варіантами доступу і побудови індексів за ключовими полями та їх значеннями. В системних програмах індекси можуть використовуватись не лише для впорядкування таблиць, а і для впорядкування зв'язаних або незв'язаних довільно розпорошених в пам'яті структур для прискорення співставлення та пошуку серед них за впорядкованими ключами.

По суті індекси є спрощеними або частковими таблицями, в яких базові поля зберігають вказівники на ключові частини полів таблиць, а додаткові поля включають організуючі вказівники на елементи, зв'язані індексом. В СУБД, ІС та системних програмах не прийнято виконувати впорядкування безпосередньо в таблицях, тому що воно пов'язане з виконанням функцій, нерегулярних за часом виконання, які іноді потребують надто багато часу.

В більшості програм головною метою індексації є прискорення розв'язання задач пошуку за рахунок додаткової інформації, що

зберігається в структурах індексів. Фактично індекси є посередниками доступу до даних, зв'язані з ключовими полями таблиць.

Основні типи індексів для підвищення швидкості пошуку

Основу побудови індексів складають адреси, вказівники або посилання на сусідні або зв'язані елементи таблиць. Найпростішу організацію ланцюгових структур з посиланнями називають списками, серед яких однозв'язні списки мають посилання на сусідні елементи в одному напрямку, а двозв'язні – мають зв'язки в обох лінійних напрямках. Спискові структури таблиць, як і інші структури з посиланнями, використовують спеціальні значення для кінцевих елементів або їх ключових частин. При реалізації списків в рамках мови C/C++ як спеціальні значення можна використовувати стандартне значення `NULL` порожнього покажчика, визначене в заголовках `stdio.h` і `stdlib.h`. Відзначимо, що механізми списків та структур з посиланнями можна використати безпосередньо для побудови самих таблиць.

Наведемо уніфіковану структуру індексного вузла, яка дозволятиме будувати однозв'язні та двозв'язні списки, а також структури двійкових та розгалужених дерев з невеликою надлишковістю.

```
struct indStr// структура елемента індексу
{struct keyStr* pKyStr;//вказівник на ключову
частину
```

```
 struct indStr* pLtPtr;//вказівник вліво
 struct indStr* pRtPtr;//вказівник вправо
};
```

Використання таких вузлів для кожного запису реляційної таблиці дозволяє побудувати індекс в окремих файлух або масивах індексу, або в індексних сегментах єдиного файлу бази даних. Крім вказівника ключової частини необхідно визначити кількість та порядок полів, що використовуються в індексі. До того ж в більшості реляційних баз даних використовується або послідовності, або вирази, які визначають індекс. В найпростішому випадку використовується впорядкована послідовність полів, і, таким чином, в загальному випадку потрібна функція, яка буде визначати відношення порядку в індексі.

Найбільш поширені три варіанти індексів:

1. індекс у вигляді впорядкованого масиву;
2. індекс у вигляді двійкового дерева, де вказівники утворюють структуру з двома відгалуженнями;
3. індекс у вигляді розгалуженого B-дерева (Brunched-tree), де покажчики утворюють структуру з *n* відгалуженнями, де *n* називають основою дерева (звичайно *n* невелике – до 256).

При виконанні базової операції `select` повинно виконуватись просування за індексом з наступною вибіркою даних з таблиці. При обробці базової операції `insert` нові рядки даних повинні додаватись до таблиці в найближче вільне місце, а індекс повинен реконфігуруватись для збереження впорядкування. Операції

delete і update спираються на результати операції select з наступними діями відмітки вилучення або заміни окремих полів запису з відповідною корекцією індексу.

Тоді для створення та розширення окремих індексів визначаються функції, що створюють та розширюють індексні структури або файли. Функція crLsIx створює індекс динамічної таблиці з виділенням сегментів первинного резервування, а для розширення пам'яті зарезервованої для індексу динамічної таблиці на одну область сегментів використовують функцію extDnIx1, яка розробляється для найскладнішого пункту завдання.

Функція crLsIx створення індексу у вигляді впорядкованого списку може бути визначена наступним чином

```
struct hdIxStr*crLsIx(struct hdTbStr*p)
{int n0, ln=p->frstSg->nFlEl;
 struct indStr *pCur, *pCurl, **pCurp, **pCurp0;
 struct hdIxStr*pI = (struct hdIxStr*)// заголо-
вок
         malloc(sizeof(struct hdIxStr));
 pI->pHdTb=p;
 pI->pBgNdx=(struct indStr*)// тип сегмента
      malloc(p->nFlEl*ln*// кількість елементів
            sizeof(struct indStr));
 pI->pRtNdx = NULL;
 pCur = pI->pBgNdx;
```

```
 for(n0=0; n0<ln; n0++) // побудова індексу як
списку
     {pCur[n0].pKyStr=&((p->frstSg->pRcPtr+n0)-
>key);
   pCur[n0].pRtPtr = NULL;
   pCur[n0].pLtPtr = NULL;
   if(n0){pCurl = pI->pRtNdx;
 pCurp0 = pCurp = &(pI->pRtNdx);
    while(pCurl)
     {if(cmpKys(pCur[n0].pKyStr,pCurl-
>pKyStr)<0)
      {pCur[n0].pRtPtr = pCurl;
       if(pCurp==pCurp0)*pCurp0=pCur+n0;
       else(*pCurp0)->pRtPtr=pCur+n0;
       break;}
      else{pCurp0=pCurp;
        pCurp=&(pCurl->pRtPtr);
        pCurl=pCurl->pRtPtr;
        if(pCurl==0)*pCurp=pCur+n0;}}}
    else pI->pRtNdx = pCur;
   }
 return pI;
}
void  extDnIx1(struct hdTbStr*p)// початковий
сегмент
{struct sgTbStr*ps = p->pNxtSg;
```

```
 while(ps)ps=ps->pNxtSg;
 ps->pNxtSg = (struct sgTbStr*)// сегмент
         malloc(sizeof(struct sgTbStr));
 ps->nRsEl = p->nSgLm*p->nSgSc;
 ps->nFlEl = 0;
 ps->pNxtSg = NULL;
 ps->pRcPtr = (struct recrd*)// записи
      malloc(ps->nRsEl*sizeof(struct recrd));
}
```

До цих методів доцільно додати методи перебудови індексів до таблиць з заповненням з початку та додаванням елементів.

Метод `crDnTb` створює динамічну таблицю з виділенням сегментів первинного резервування, а метод `extDnTb1` розширює динамічну таблицю на один сегмент. До цих методів доцільно додати методи імпорту даних до таблиць з заповненням з початку та додаванням елементів (методи імпорту `impDnTb` і розширення таблиці `insDnTb` в шаблоні програми).

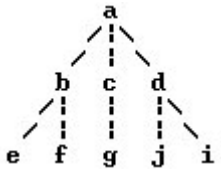
Організація пошуку в деревоподібних структурах

Основою перегляду списків та деревоподібних структур є адреси, посилання або покажчики на зв'язані записи. Кореневе дерево (орієнтоване дерево, дерево) - орієнтований граф, в якому:

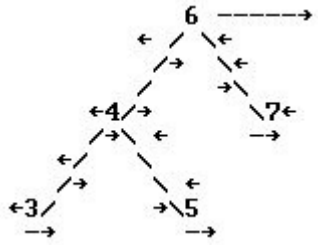
1) всі вершини, окрім однієї (кореня), знаходяться в голові тільки однієї дуги;

2) корінь дерева не знаходиться в голові будь-якої дуги;

3) корінь пов'язаний з кожною вершиною дерева..



Двійкове дерево - це дерево, в якому у кожної вершини не більше 2-х нащадків. Приклад:



Обхід дерева визначається впорядкованою послідовністю вершин дерева, в якому кожна вершина зустрічається лише один раз. У кожну вершину потрапляємо принаймні один раз, а взагалі кажучи 3 рази. При обході дерева можуть виконуватися різні дії, наприклад, друк позначення. Якщо друкувати мітку при першій зустрічі з нею, то виходить послідовність 6,4,3,5,7; якщо при другому, то - 3,4,5,6,7; якщо при третьому, то - 3,5,4, 7,6. Ці три способи обходу називають відповідно обходом зверху, обходом зліва направо, обходом знизу.

Лінійний та двійковий пошук в структурах з посиланнями

Для вибірки відповідного запису з таблиці через індекс у формі списку треба виконати послідовність порівнянь у відповідності з визначеними відношеннями рівності та порядку.

```
// вибірка за лінійним пошуком через індекс в
формі // списку
struct recrd*selLin(struct keyStr kArg,
  struct indStr *ndx)
{while(ndx!=NULL&&cmpKey(ndx->pKyStr,kArg))
ndx=ndx->pRtPtr;
 if(ndx==NULL)return NULL;
 return ndx->pKyStr;
}
```

Якщо лінійний пошук треба виконати через деревоподібний індекс, то для цього використовуються методи повного рекурсивного обходу дерева. Методи двійкового пошуку можна використовувати лише у впорядкованих деревах або у відповідних індексах. Тобто індекси для виконання такого пошуку повинні бути попередньо впорядковані і для роботи з ними треба мати функції включення відповідно порядку та балансування дерев.

```
//вибірка за двійковим пошуком через індекс в
формі дерева
struct recrd*selBin
(struct keyStr kArg,// ключ аргументу пошуку
 struct indStr *ndx)// адреса кореня індексу
```

```
{ while(ndx!=NULL&&(i=cmpKey(ndx->pKyStr,kArg)))
    {if(i>0)ndx=ndx->pLtPtr;else ndx=ndx->pRtPtr;
    }
 if(ndx==NULL)return NULL;
 return ndx->pKyStr;
}
// виведення індексованого рядка базової таблиці
void prRow(struct recrd* rw)
void prRow(struct recrd* rw)
{if(rw==0)printf("is absent\n");
 else printf("%10s %3u %5.3f\n",
rw->key.str,rw->key.nMod,rw->func._f);
}
```

Системні програми в процесі свого виконання використовують власну вбудовану інформаційну базу (ІБ), а також будує інформаційну базу про текст на заданій мові,

Вимога унікальності ключів і аргументів пошуку

В системних програмах звичайно додержуються вимог унікальності ключів і порушення такої вимоги розглядається як помилка повторного опису. Такої помилки і порушення вимоги унікальності можна уникнути додаючи до запису таблиці додаткове ключове поле позиції визначення. Але в реляційних СУБД повторювані значення ключів часто використовуються при зв'язуванні таблиць, і поля з однаковими значеннями ключів можна впорядко-

вувати довільним чином. В цьому випадку важливо мати набір функцій, які дозволятимуть одержувати перший ліпший результат пошуку, черговий результат пошуку при черговому запиті і всі можливі результати пошуку при одному виклику.

Індекси у вигляді B-дерев

Очевидно, що фізичне впорядкування записів індексу має ті ж недоліки, що і фізичне впорядкування самих записів основного файлу: необхідність реорганізації файлу при операціях вставки, видалення, модифікації. Отже, для організації індексу повинен використовуватися деякий варіант кіпи. Однак при цьому недостатньо мати просту структуру типу двонаправленого списку, оскільки доведеться при кожній модифікації виконувати сортування. При цьому також потрібно мати на увазі, що кількість записів в у індексі можу бути дуже великим (розміри файлу можуть бути у декілька разів більшими за наявну оперативну пам'ять), такими чином, використання звичайних методів сортування, які орієнтовані на розміщення усіх елементів сортування в оперативній пам'яті, може бути неможливим.

В оперативній пам'яті впорядковані структури, які допускають ефективну вставку, видалення і модифікацію даних зазвичай організуються у вигляді збалансованих дерев (AVL-дерев). Збалансовані дерева – це двійкові дерева, для яких постійно підтримується рівномірних розподіл вершин між піддеревами. При цьому досягається близька до теоретичної (для двійкового пошуку) ефективність пошуку (близько $\log_2 N$ кроків, де N - загальна кількість

елементів, серед яких проходить пошук) при нормальному використанні ресурсів для балансування.

Дерева можна розглядати як упорядкований список, у якому чергуються ключі та посилання (покажчики) на нащадків.

Кожна вершина має не більше ніж 2*n* ключів.

Кожна вершина за виключенням, може бути, кореневої складається не менш ніж з *n* ключів (коренева вершина, якщо вона не є листочком, має не менш ніж два сина)

Кожна вершина або являється листом і не має дітей, або має точно *m*+1 нащадків, де *m* - кількість ключів у вершині

Всі листові вершини знаходяться на одному рівні.

Число *n* називається порядком B-дерева.

Широке практичне застосування отримала модифікація механізму B-дерев, яку прийнято називати B+-деревами. Ці дерева схожі на звичайні B-дерева. Вони теж розгалужені, і довжина шляху від кореня до будь-якої листової сторінки одна й та ж. Але структура внутрішніх і листових сторінок різна. Внутрішні сторінки влаштовані так само, як у B-дерева, але в них зберігаються тільки ключі (без записів) і посилання на сторінки-нащадки. У листових сторінках зберігаються всі ключі, що містяться в дереві, разом із записами, причому цей список впорядкований за зростанням значення ключа.

Завдання на роботу

Завдання на підготовку до роботи на комп'ютері

1. Визначити варіант завдання для основних задач за таблицею 5.1. Типи ключових і функціональних полів зберегти з попередньої роботи.

2. Відповісти на контрольні запитання.

3. Ознайомитись з шаблоном програмного проекту spLb2. Настроїти відповідні дані в програмному проекті на мові С.

4. Настроїти виклики функцій лінійного, двійкового або хеш-пошуку для роботи з динамічними таблицями, а також формування відміток про вилучення, упакування таблиці з вилученими елементами і вставку до таблиці за значенням ключового елемента з використанням методу, заданого у варіанті.

5. Використати структуру індексу **`struct`** `indStr` з файлу index.h шаблону програмного проекту spLb2 для побудови елементу індексу таблиці і визначити наступні підпрограми з методом впорядкування заданим в таблиці 5.1.

- для вставки до таблиці з корекцією індексу;
- для вибірки з таблиці за індексом;
- для корекції таблиці з індексом.

6. Підготувати програмний модуль контрольної задачі, який виконує задані варіанти програм пошуку і вставки в таблицю за значенням ключового елемента і дозволяє перевірити коректність виконання програм.

Завдання на роботу на комп'ютері

7. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

8. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми.

9. Одержати результати виконання, проаналізувати їх і зробити висновки.

Питання для самостійної перевірки

1. Дайте визначення індексів таблиць як агрегатів даних, які визначають реляційну залежність між окремими характеристиками.
2. Який вигляд можуть мати індексні частини таблиць?
3. Якими структурами даних або об'єктами визначається елемент індексу?
4. Як визначається кількість елементів або кінець динамічної таблиці?
5. Яка інформація використовується для ключів пошуку з використанням індексів?
6. Як створюються динамічні таблиці у вигляді масивів структур мови С або записів мови Pascal?
7. Які вимоги висуваються перед функціями пошуку системних програм?
8. Які вимоги висуваються до підпрограм, функцій та процедур, що обробляють індекси?
9. Чим деревоподібні індекси відрізняються від хеш-індексів?

Порядок вибору варіанту:

За останньою цифрою номера залікової книжки або за порядковим номером студента в списку підгрупи з доданим номером групи або визначте варіант завдання для задач за табл. 5.1.

Таблиця 5.1

Варіанти завдань для виконання пошуку за індексом

№ вар.	Тип індексу	Метод для вставки	Тип
1	**Впорядкований масив**	**Двійковий**	Перший
2	**Двійкове дерево**	**Лінійний**	Всі
3	**B-дерево**	**Двійковий**	Черговий
4	**Хеш-функція**	**Колізія: Лінійний**	Перший
5	**Двійкове дерево**	**Двійковий**	Всі
6	**B-дерево**	**Лінійний**	Черговий
7	**Хеш-функція**	**Колізія: Повторний хеш**	Перший
8	**Двійкове дерево**	**Лінійний**	Всі
9	**B-дерево**	**Двійковий**	Черговий
10	**Впорядкований масив**	**Лінійний**	Перший
11	**Двійкове дерево**	**Двійковий**	Всі
112	**B-дерево**	**Лінійний**	Черговий
113	**Хеш-функція**	**Колізія: Лінійний**	Перший
114	**Двійкове дерево**	**Лінійний**	Всі
115	**B-дерево**	**Двійковий**	Черговий
116	**Впорядкований масив**	**Лінійний**	Перший
117	**Двійкове дерево**	**Двійковий**	Всі
118	**B-дерево**	**Лінійний**	Перший
119	**Впорядкований масив**	**Двійковий**	Всі
220	**Двійкове дерево**	**Лінійний**	Черговий

2.5 Лабораторна робота 5

Тема роботи: Програмування з використанням функціональних та процедурних типів

Мета роботи: Вивчення методів створення та використання функціональних та процедурних типів мов програмування для розрахунків визначених інтегралів та одержання практичних навичок параметризації підінтегральних функцій. Вивчення можливостей функціональних типів для управління переключеннями функцій.

Короткі теоретичні відомості

Більшість числових методів розрахунку визначених інтегралів спирається на розбиття інтервалів інтегрування на деяку кількість фрагментів, яка забезпечує потрібну точність розрахунків. В цьому випадку важливо побудувати загальну функцію нітрування заданим методом, в якій параметром буде підінтегральна функція.

Визначення функціональних типів

Аргументи функцій та покажчики абстрактного функціонального типу можуть визначатися без іменування, тобто неявно, так само і дані інших типів, що визначаються користувачем. Дані функціонального та процедурного типу фактично не є інформаційними даними, а управляючими даними для звернення для функцій і процедур, тому в цьому типі слід визначати принаймні послідовність та типи аргументів, як і в звичайному заголовку або прототипі функції або процедури.

Дані функціонального або процедурного типів не можуть бути звичайними примірниками даних або елементами масивів, але можуть бути покажчиками та масивами покажчиків, які в свою чергу можуть бути елементами структур та об'єднань даних. Прикладом може бути заголовок (об'ява) та визначення функції інтегрування:

```
double integral(double f(double),
                double xB, double xE, unsigned n);
double integral(double f(double),
                 double xB, double xE, unsigned n)
    {unsigned i; double s=0.0, dx=(xE-xB)/n;
     for(i=0; i<n; i++)
       s+=dx*f(xB+i*dx);
     return s;}
```

В цьому прикладі функції інтегрування за формулою лівих прямокутників перший аргумент `f` є формальним аргументом функціонального типа, який повертає значення типа **`double`** і потребує одного аргументу того самого типа. Як і більшість інших типів користувача функціональні типи в мові C/C++ можуть явно визначатися та іменуватися в операторах **`typedef`**, але можуть визначатися і неявно відповідною послідовністю позначень [5]. При цьому головне визначити спосіб доступу до функції, як правило, через покажчик або посилання, тип результату, а також кількість та типи потрібних аргументів. Наприклад:

```
typedef double funct(double);
```

В цьому функціонального типа **`funct`** є іменем функціонального типа, який повертає значення типа **`double`** і потребує одного аргументу того самого типа. Це ім'я може використовуватись для посилання на функціональний тип при визначенні будь-яких припустимих даних цього типа.

Реалізація розрахунку значень та інтервалі інтегрування

Основу найпростішої програмної реалізації скінченного автомата складають коди стану автомата та так звана матриця переходів автомата. Коди стану, частіше за все, визначаються перенумерованим типом з іменованими значеннями станів. Приклад опису типу для подання автомата з 5-ма станами показано нижче на рис. 5.1. У вузлах, позначених кружальцями записано ім'я стану, а біля дуг, позначених стрілками записані імена сигналів, що переключають автомат до стану, показного в напрямку стрілки. Відсутність дуги, що виходить з вузла з відповідним номером сигналу, дає інформацію про те, що за цим сигналом відповідний стан не змінюється.

Завдання на роботу

Завдання на підготовку до роботи на комп'ютері

1. Визначити варіант завдання для основних задач за таблицею 5.1.

2. Підготовити програмний проект з визначенням функціонального типу для підінтегральних функцій з однією змінною.

3. Створити параметризовану функцію обчислення значень визначених інтегралів та перевірити роботу програми принаймні на двох прикладах, заданих в таблиці 5.1.

Таблиця 5.1

Варіанти завдань для інтегрування підінтегральних функцій

№ вар.	Інтеграл 1	Інтеграл 2	Метод інтегрування	Інтервал інтегрування
1	**Y=∫sin(x)dx**	**Y=∫sin(x) cos(x)dx**	лівого прямокутника	**a<=x<=b**
2	**Y=∫cos(x)dx**	**Y=∫tan(x) dx**	трапеції	**0<=x<=1;**
3	**Y=∫ch(x)dx**	**Y=∫cos(x) dx**	прямокутника	**-π/2<=x<=-π/4**
4	**Y=∫sh(x)dx**	**Y=∫cos(x)dx**	трапеції	**0<=x<= e**
5	**Y=∫tan(x) dx**	**Y=∫ch(x)dx**	прямокутника	**0<=x<=1;**
6	**Y=∫cos(x) dx**	**Y=∫sh(x)dx**	трапеції	**0<=x<=1;**
7	**Y=∫cosec(x) dx**	**Y=∫tan(x) dx**	прямокутника	**0<=x<= π/4**
8	**Y=∫sec(x) dx**	**Y=∫cos(x) dx**	трапеції	**-π/2<=x<=-π/4; -π/4<=x<= 0**
9	**Y=∫exp(x) dx**	**Y=∫tan(x) dx**	прямокутника	**0<=x<=1;**
10	**Y=∫sin(x)dx**	**Y=∫ln(x)dx**	трапеції	**0<=x<=π/2**
11	**Y=∫ln(x)dx**	**Y=∫tan(x) dx**	прямокутника	**0<=x<=1;**
12	**Y=∫exp(x) dx**	**Y=∫cos(x) dx**	трапеції	**0<=x<=1;**
13	**Y=∫tan(x) dx**	**Y=∫ln(x)dx**	прямокутника	**-π/2<=x<=-π/4**
14	**Y=∫cos(x) dx**	**Y=∫cosec(x) dx**	трапеції	**0<=x<= 1/e**
15	**Y=∫ln(x)dx**	**Y=∫exp(x) dx**	правого прямокутника	**0<=x<=1; 0<=x<= e**
16	**Y=∫cos(x)sin(x)dx**	**Y=∫cos(x) dx**	трапеції	**0<=x<= π/2; 0<=x<= π/2**
17	**Y=∫exp(x) dx**	**Y=∫tan(x) dx**	прямокутника	**0<=x<= π/2**
18	**Y=∫exp(x) dx**	**Y=∫cosec(x) dx**	трапеції	**0<=x<= π/2**
19	**Y=∫cos(x)dx**	**Y=∫sec(x) dx**	прямокутника	**0<=x<= π/2**
20	**Y=∫ch(x)dx**	**Y=∫cosec(x) dx**	трапеції	**0<=x<=n**
21	**Y=∫sh(x)dx**	**Y=∫sec(x) dx**	прямокутника	**0<=x<=n**

4. Підготувати програмний модуль контрольної задачі, який виконує задані варіанти програм і дозволяє перевірити коректність і досягнення потрібної точності виконання програми і її окремих модулів.

Завдання на роботу на комп'ютері

6. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

7. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми для перевірки результатів контрольних прикладів.

8. Одержати результати виконання, проаналізувати їх і зробити висновки.

Порядок вибору варіанту:

За останньою цифрою номера залікової книжки або за порядковим номером студента в списку підгрупи з доданим номером групи визначте за табл. 5.1 варіант оброблюваних даних та настройки програм за прикладом.

Питання для самоперевірки

1. Дайте визначення основних математичних методів числового обчислення визначених інтегралів.
2. Як визначаються функціональні та процедурні типи?
3. Як використовуються функціональні та процедурні типи?
4. Для чого можна визначати та використовувати покажчики функціональних та процедурних типів?

   Як можна зробити ім'я функції фактичним аргументом?

2.6 Лабораторна робота 6

Тема роботи: Програмування та налагодження таблиць для програмних кодів зміни стану автомата

Мета роботи: Вивчення методів створення та використання вузлів графів автоматів та одержання практичних навичок аналізу та налагодження програмної реалізації автомата з використанням контрольних точок призупинення програм.

Короткі теоретичні відомості

Більшість видів організації складного управління в програмному забезпеченні можна реалізувати за допомогою скінченних автоматів. Розпізнавання та перетворення управляючих сигналів, вхідних текстів програм на управляючі дії спираються на графи зміни станів автоматів, в яких важливими є коди станів та сигналів переключення автомату.

Представлення графів автомата в програмному забезпеченні

З двох базових методів представлення автоматів в теорії (графового та у вигляді матриці переходів автоматів) для програмування швидкої роботи придатні матриці переходів, а для зручного відображення графові форми [1]. Для подальшої їх визначення в програмах необхідно визначити кодування станів та сигналів автомата.

Реалізація швидкісної роботи автоматів з пам'яттю

Основу найпростішої програмної реалізації скінченного автомата складають коди стану автомата та так звана матриця переходів автомата. Коди стану, частіше за все, визначаються перенумерованим типом з іменованими значеннями станів. Приклад опису типу для подання автомата з 5-ма станами показано нижче на рис. 6.1. У вузлах, позначених кружальцями записано ім'я стану, а біля дуг, позначених стрілками записані імена сигналів, що переключають автомат до стану, показного в напрямку стрілки. Відсутність дуги, що виходить з вузла з відповідним номером сигналу, дає інформацію про те, що за цим сигналом відповідний стан не змінюється.

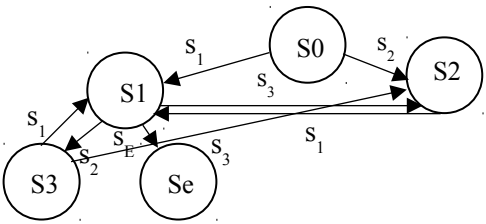


Рис. 6.1. Стани кінцевого автомата

```
enum autStat
{S0,   // S0 - Початковий стан
 S1,   // S1 - Перший стан
 S2,   // S2 - Другий стан
 S3,   // S3 - Третій стан
 Se    // Se - Кінцевий стан
```

```
};
```

Коди сигналів також зручно визначати іншим перенумерованим типом з іменованими значеннями кодів сигналів. Приклад опису типу для подання 5 типів сигналів, показаного на рис. 2.1, наведено нижче.

```
enum autSgn
{sg0,  // sg0 - Початковий сигнал
 sg1,  // sg1 - Перший сигнал
 sg2,  // sg2 - Другий сигнал
 sg3,  // sg3 - Третій сигнал
 sgE   // sgE - Останній сигнал
};
```

Матриця переходів визначається двомірним масивом типу **enum** autStat, перший індекс якого визначає ціле число, яке відповідає попередньому стану автомата, а другий індекс – число, яке відповідає сигналу або класу сигналу для переведення автомата в наступний стан.

```
enum autStat nxtSts[Se+1][ sgE+1] =
{{S0,S1,S2,S0,S0},  //для S0
 {S1,S1,S1,S2,Se},  // S1
 {S2,S1,S2,S2,S2},  // S2
 {S3,S1,S3,S2,S3},  // S3
 {Se,Se,Se,Se,Se}}; // Se
```

Функція переходів визначена в лабораторній роботі наступним чином:

```
enum autStat nxtStat(enum autSgn sgn)
{static enum autStat s=S0;// поточний стан лексеми
```

```
 return s=nxtSts[s][sgn];} // новий стан лексеми
```

До виконавчої частини треба додати масив з послідовністю вхідних сигналів та цикл звертання до обробки з роздруком результатів моделювання роботи автомата:

```
enum autSgn ASgn[10]={sg1,sg2,sg1,sg0,...,sgE}; ...
for(n=0;n<10;n++)
      printf("%5d->%2d  ",ASgn[n],nxtStat(ASgn[n]));
```

Структура програмного шаблона виконання роботи

Для спрощення розв'язання задачі слід використовувати прототип проекту spLb3.dsp, побудований на базі проекту spLb1.dsp і зберігається в робочому просторі spLb1.dsw в підпапці spLb3 папки spLb1, яка зберігає шаблони прототипів для всіх лабораторних робіт циклу «Алгоритми і структури даних». До складу проекту консольної прикладної програми, орієнтованої на можливість використання в числі інших бібліотек стандартних функцій та об'єктів MFC (Microsoft Foundation Classes) входять модулі з наступними вхідними файлами заголовків і реалізацій:

- StdAfx.h і StdAfx.cpp – модуль для організації використання об'єктів класів MFC;
- token.h і token.cpp – модуль для опису кодування типу лексеми, структури вузла лексеми і організації використання структур або класів лексем в процесі лексичного і синтаксичного аналізу;
- visgrp.h і visgrp.cpp – модуль для відображення рядків або повних таблиць;

- automat.h і automat.cpp – модуль для визначення та управління автоматом;
- spLb3.cpp – модуль контрольних прикладів для тестування функцій обробки графів.

Розділ визначень і декларацій основної частини програми тестування в модулі spLb3.cpp повинен включати визначення послідовностей вузлів, що утворюють графи лексем. Розділ визначень модуля visgrp.cpp забезпечує визначення мов реконструкції за варіантом завдання. Виконавча частина модуля spLb3.cpp програми повинна включати циклічно повторювані виклики функцій реконструкції для різних варіантів операторів і мов.

Завдання на роботу

Завдання на підготовку до роботи на комп'ютері

1. Визначити варіант завдання для основних задач за таблицею 3.1. Визначити приклади лексем через константи в модулі тестування spLb3.cpp.

2. Ознайомитись з шаблоном програмного проекту spLb3.dsp. Настроїти відповідні дані символьних позначень та структур вузлів графів в програмному проекті на мові С.

3. Підготовити настройку управляючої матриці згідно з варіантами, заданими в табл. 6.1. Використати масив матриці переходів **`char`** `nxtSts [Se+1][sgE+1]` з файлу automat.cpp шаблону програмного проекту spLb3 для формування станів автомату, які будуть пройдені за заданим варіантом.

5. Підготувати настройку програми реконструкції для еквівалентних конструкцій на альтернативній мові програмування.

6. Підготувати програмний модуль контрольної задачі, який виконує задані варіанти програм реконструкції і управління станом автомата і дозволяє перевірити коректність виконання програми і її окремих модулів.

Таблиця 6.1

Варіанти завдань для виконання роботи з автоматами

№ вар.	Настроювання графа автомата з послідовними станами	Мова відтворення
1	Стани 0..9; 3->7(*dlm*), 5->8(*cfr*)	C
2	Стани 1..7; 3->7(*dlm*), 5->2(*cfr*)	Pascal
3	Стани 2..9; 3->8(*dlm*), 5->3(*ltr*)	C
4	Стани 0..9; 3->7(*dlm*), 5->8(*cfr*), 3->2(*ltr*)	Pascal
5	Стани 1..8; 3->8(*dlm*), 5->5(*ltr*)	C
6	Стани 0..9; 3->7(*dlm*), 5->8(*cfr*), 3->3(*ltr*)	Pascal
7	Стани 0..8; 3->3(*dlm*), 5->1(*ltr*)	C
8	Стани 0..8; 3->7(*dlm*), 5->5(*cfr*), 3->2(*ltr*), 6->8(*cfr*)	Pascal
9	Стани 1..9; 3->3(*dlm*), 5->9(*ltr*)	C
10	Стани 1..9; 7->7(*dlm*), 5->3(*cfr*), 3->1(*ltr*)	Pascal
11	Стани 1..8; 3->7(*dlm*), 5->5(*cfr*), 4->8(*ltr*)	C
12	Стани 2..9; 8->2(*dlm*), 5->7(*cfr*), 3->7(*ltr*)	Pascal
13	Стани 0..6; 3->3(*dlm*), 5->2(*ltr*)	C
14	Стани 2..7; 3->7(*dlm*), 3->3(*cfr*), 1->4(*ltr*)	Pascal
15	Стани 1..7; 7->7(*dlm*), 3->3(*cfr*), 1->1(*ltr*)	C
16	Стани 2..9; 5->7(*dlm*), 3->3(*cfr*), 9->4(*ltr*)	Pascal
17	Стани 0..7; 3->7(*dlm*), 5->3(*cfr*), 4->4(*ltr*)	C
18	Стани 2..9; 2->2(*dlm*), 5->9(*ltr*)	Pascal
19	Стани 2..9; 2->2(*dlm*), 5->9(*ltr*)	C
20	Стани 1..7; 3->6(*dlm*), 5->5(*cfr*)	C

Завдання на роботу на комп’ютері

6. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

7. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми для перевірки результатів контрольних прикладів.

8. Одержати результати виконання, проаналізувати їх і зробити висновки.

Порядок вибору варіанту:

За останньою цифрою номера залікової книжки або за порядковим номером студента в списку підгрупи з доданим номером групи визначте за табл. 6.1 варіант оброблюваних даних та настройки програм за прикладом.

Питання для самоперевірки

8. Дайте визначення основних типів автоматів, які визначають процес управління обробкою даних.
9. Якими структурами даних або об'єктами визначається графи автомата?
10. Які типи даних використовуються для визначення станів і сигналів управління автоматів?
11. Якими масивами можна програмно визначити скінченний автомат?
12. Якими операторами мов програмування організуються переходи автомату з одного стану до іншого?

2.7 Лабораторна робота 7

Тема роботи: Побудова і використання об'єктів вузлів деревоподібних та ієрархічних графів

Мета роботи: Вивчення методів створення та використання структурованих вузлів деревоподібних та ієрархічних графів, організації доступу до інформації, відображення графів та реконструкції збереженого в графі тексту. Програмування відтворення графів підлеглості операцій через покажчики у форматі вхідного тексту мови програмування.

Короткі теоретичні відомості

Більшість видів граматичного аналізу та семантичної або змістовної обробки текстів програм або скриптів у форматах внутрішнього подання спираються в комп'ютері на різні подання графів та їх деревоподібних підвидів. Більшість текстових редакторів та процесорів, в тому числі редактори фірми Microsoft зберігають інформацію у внутрішній формі, і часто навіть між сеансами обробки у вигляді структур, що відображуються графами. Такі графи, крім зберігання різноманітних текстів, дозволяють зберігати хронологію виконання корекцій, авторів тексту і навіть структуру та елементи змісту речення у суворо визначеному внутрішньому представленні.

Графи, що використовуються для зберігання текстів у внутрішніх формах

Для змістовного аналізу та інших видів семантичної обробки математичних виразів вони перетворюються на графи підлеглості операцій та ключових слів або скорочені спрямовані ациклічні графи (directed acyclic graph – DAG) [7,8], які відображують роздільники і ключові слова операцій як нетермінали, а імена та константи в програмах як термінали.

Для того щоб надати можливість використання одноманітних структур даних на різних етапах обробки комп'ютерних мов і, таким чином, мінімізувати витрати на повторювані пересилання даних, важливо визначити базову структуру елемента внутрішнього подання або лексеми стандартним для всіх етапів виконання програми. Для обробки графів важливо визначити покажчики, за якими виконуються всі види обробки. Тоді кожна елемент виразу повинен визначатися структурою, яка зберігає дані про його зв'язки та характеристики.

```
struct lxNode//вузол дерева або САГ
{int ndOp;     //код операції або типу лексеми
 unsigned stkLength;// номер модуля для терміналів
 struct lxNode* prvNd, *pstNd;// до підлеглих вузлів
 struct lxNode*prnNd;};//до батьківського вузла
```

Такі вузли з одного боку визначають масив елементів тексту у послідовності їх надходження. При мінімізації вузли графа розглядаються як елементи таблиці, в якій ключами повинні бути однозначно подані вирази, для яких визначається відношення поря-

дку в послідовності ієрархічного або рекурсивного відтворення рядків з ключових полів вузлів графа.

Визначення відношення порядку спирається на принципи відображення в деревах або графах префіксних форматів відтворення виразів. Як було показано в лабораторній роботі 3, відношення порядку ключових полів складає основу впорядкування таблиць та побудови впорядковуючих індексів для всіх видів підграфів. При використанні відношень порядку індекси для послідовностей записів (структур) вузлів лексем дозволяють підвищувати швидкість вибірки при розв'язанні різних задач семантичної обробки в системних програмах, насамперед, машинно-незалежної оптимізації. Приклади графів простих виразів і умовних операторів показані на рис 7.1, рис. 7.2, рис. 7.3 і рис. 7.4.

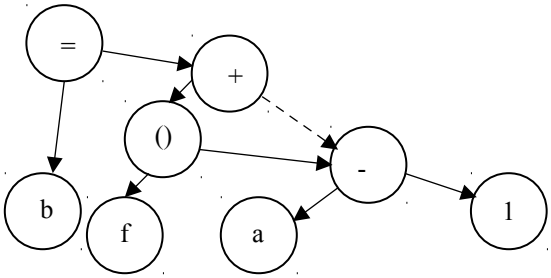


Рис. 7.1. Спрямований ациклічний граф оператора присвоювання `b=f(a-1)+a-1`

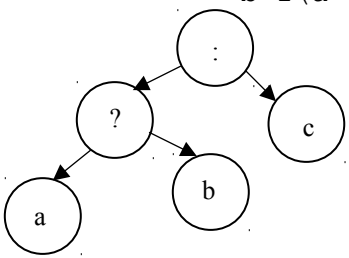


Рис. 7.2. Дерево підлеглості операцій для виразу `a?b:c`

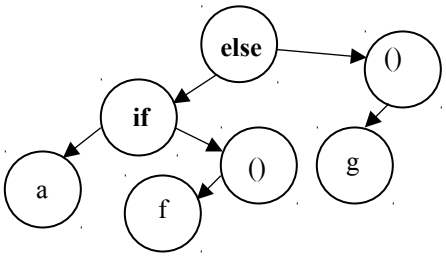


Рис.7.3. Дерево підлеглості для оператора **`if`**`(a)f();`**`else`** `g();`

Приклади подання графів простих виразів, показані на рис. 7.1, 7.2 і 7.3, будуються на основі масиву управління відтворенням символьних образів **`char`** `*imgs[]`. Нижче показано приклад масиву вузлів для рис. 7.1 з кореневим вузлом `pic1[1]`.

```
char *imgs[]={"b","a","1","f","c","g","k","ky",
                                    "n","nD","nU","el"};
struct lxNode pic1[]=      // b=f(a-1)+a-1
{{_nam,(struct lxNode*)imgs[0], NULL},
 {_ass,&pic1[0], &pic1[2]},
 {_nam, (struct lxNode*)imgs[3], NULL},
 {_brkt,&pic1[2], &pic1[5] },
 {_nam,(struct lxNode*)imgs[1], NULL},
 {_sub, &pic1[4], &pic1[6]},
 {_srcn,(struct lxNode*)imgs[2], NULL},
 {_add, &pic1[3], &pic1[5]}
};
```

Наступний опис показує приклад масиву вузлів для рис. 7.2 з кореневим вузлом `pic2[1]`.

```
struct lxNode pic2[]=      // b=a?b:c
```

```
{{_nam, (struct lxNode*)imgs[0], NULL},
 {_ass, &pic2[0], &pic2[5]},
 {_nam, (struct lxNode*)imgs[1], NULL},
 {_qmrk,&pic2[1], &pic2[4]},
 {_nam, (struct lxNode*)imgs[0], },
 {_cln, &pic2[3], &pic2[6]},
 {_nam, (struct lxNode*)imgs[4], NULL}
};
```

Наступний опис показує приклад масиву вузлів для рис. 7.3 з кореневим вузлом `pic3[4]`.

```
  struct lxNode pic3[]=       // if(a)f();else g();
  {{_nam,(struct lxNode*)imgs[1], NULL},
   {_if, &pic3[0], &pic3[3]},
   {_nam,(struct lxNode*)imgs[3], NULL},
   {_brkt,&pic3[2], NULL, 0, 0, 0, 0, 0, NULL, 0},
   {_else,&pic3[1], &pic3[6] },
   {_nam,(struct lxNode*)imgs[5], NULL},
   {_brkt,&pic3[5], NULL},
  };
```

Алгоритм реконструкції як база для вивчення методики обходу графів

Для побудови систем програмування важливо побудувати програми аналізу та реконфігурації на основі єдиних таблиць, зв'язаних з реалізованою мовою так, щоб для кожної нової мови будувався мінімум схожих таблиць на основі стандарту внутрішнього подання, прийнятого в багатомовній системі програмування.

Гнучкість перебудови мов звичайно забезпечується наборами управляючих таблиць. Існує багато шляхів для визначення таких таблиць. Один з них розглянемо в цій роботі детально.

Спочатку визначимо іменування та кодування типів лексем за допомогою перенумерованого типу **enum** tokType, в якому доцільно визначитись з впорядкуванням кодів, яке буде корисним на більшості етапів розробки та настройки мови, включаючи всі види аналізу реконфігурацій і реконструкцій. Важливо забезпечити подання всіх змістовних синонімів та омонімів в окремих кодах лексем. В наведеному описі нижче типу використане узагальнене семантичне іменування лексем з прив'язкою до їх позначень в мовах C/C++, Pascal та інших комп'ютерних мовах.

```
#define begOprtr 0x50
       // зміщення початку виконавчих операторів
enum tokType
{_nil, _nam,// зовнішнє подання
 _srcn,_cnst,
            // вхідне і внутрішнє кодування константи
 _if, _then, _else, _elseif,  // if then else elseif
 _case, _switch, _default, _endcase,
                          //case switch defualt endcase
 _break, _return, _whileP, _whileN,
                            // break return while while
 _continue, _repeat, _untilN, _endloop,
                            // continue repeat until do
 _for, _to, _downto, _step,// for to downto step
 _untilP, _loop, _with, _endif,
```

```
 _goto,_extern,_var,_const,
 _enum,_struct/*_record*/,_union,_register,//
 _unsigned,_signed,_char,_short,
 _int,_long,_sint64,_uint64,//
 _float,_double,_void,_auto,
 _static,_volatile,_typedef,_sizeof,//
 _real,_array,_set,_file,_object, _string, _label,
// відкриті і закриті дужки
 _fork,_join,// паралельних операторів
 _opbr,_ocbr,// послідовних операторів
 _ctbr,_fcbr,_ixbr,_scbr,// конкатенацій і індексу
 _brkt,_bckt, _tdkt,_tckt,// порядку, функцій і даних
 _eosP, eosS,// паралельні та послідовні
 _EOS=begOprtr, _comma, _cln, _qmrk,// ; , : ?
 _asOr, _asAnd, _asXor, _asAdd,  //|= =& =^ =+
 _asSub, _asMul, _asDiv, _asMod, // -= *= /= %=
 _asShr, _asShl, _ass, _dcr, _inr, // <<= >>= = -- ++
 _lt, _le, _eq, _ne, _ge, _gt,     // < <= == != >= >
 _add, _sub, _mul, _div,_fldDt,_fldPt,// + - * / . ->
 _pwr, _shLfa, _shRga,_eqB,_neB,// ** <<< >>> === !==
 _addU,_subU,_mulU, _andU,    // + - * & унарні
 _norB,_nandB,_nxorB,_xornB, _addr, //~| ~& ~^
 _mod, _orB, _andB, _xorB,// % (div mod) | & ^
 _shLft, _shRgt, _or, _and,   //<< >> || &&
                       // (or and xor shl shr or and)
 _xmrk, _invB,_divI, _in //_not,  _notB
};
```

Наведений тип включає імена кодів, згруповані за використанням в операторах різних мов, починаючи з виконавчих операторів, які є найбільш загальними в усіх комп'ютерних мовах, і закінчуючи операторами описів і декларацій, які індивідуальними для різних комп'ютерних мов. Крім того, надається можливість включення кодів всіх ключових слів до цієї таблиці, навіть тих які виключаються при побудові внутрішньої форми кодів. Тобто такий тип можна використовувати, як при реконструкції вхідних текстів, так і при лексичному, синтаксичному та семантичному аналізі.

При реконструкції математичних і програмних текстів головною метою є коректне відтворення виразів математичних формул і програм з додержанням правил застосування мінімально потрібних дужок та інших допоміжних роздільників. Для цього треба, по-перше визначити стандартні і керовані шаблони відтворення, по-друге – автоматичне редагування табуляцій, обов'язкових знаків та переведень рядків текстів, по-третє – впорядкування конструкцій за шаблонами синтаксичного передування, а по-четверте блокування формування неявних ключових слів у вхідному тексті.

Базовим варіантом подання шаблону відтворення будемо вважати управляючі рядки видів `"xfx"`, `"xfy"` і `"yfx"` для інфіксних операцій, `"fx"` і `"fy"` для унарних префіксних операцій, `"xf"` і `"yf"` для унарних постфіксних операцій. Цей механізм використовується для відображення виразів та операторів в мові Prolog, яка має вбудовані засоби відтворення структур внутрішнього подання текстів на цій самій мові. В цих записах `y` означає

автоматично асоціативний аргумент з операцією з пріоритетом `f`, а `x` означає неасоціативний аргумент з операцією `f`, тобто у випадку операнда `y` не треба брати підлеглу операцію того ж пріоритету в дужки, а у випадку операнда `x` – треба [6]. Таким чином арифметичні операції і операції переліку можна описувати в процедурних мовах рядком `"yfx"`. Однак в більшості сучасних комп'ютерних мов, насамперед, мов запитів і процедурних мов існують більш різноманітні відношення між операціями і операндами, через що необхідно розширити набір шаблонів. Більше того, через використання літери `f` в багатьох ключових словах, що генеруються за шаблонами та бажаність використання символу табуляції для форматування тексту, краще замінити відтворювані літери `f`, `x` і `y` початковими літерами кодових таблиць коду ASCII або Unicode.

Програмна реалізація перевірки правил мінімального відтворення дужок та порожніх операторів

Основу реалізації правил складають шаблони, які визначають потреби дужок залежно від пріоритетів або передувань операцій та ключових слів операторів. Перевірка правил формування дужок виконується в процедурі **`void`** `prLxTxt(`**`struct`** `lxNode*rt)` і має вигляд продукційних правил, вбудованих в процедуру.

Типи дужок (порядку обробки виразів, операторних дужок та дужок визначення типів та значень даних) визначаються діапазоном значень, в якому знаходяться коди лексем. Таким чином, щоб скоротити програму формування дужок важливо впорядкувати зна-

чення типу **enum** `tokType`, для зменшення кількості діапазонів, які аналізуються при реконструкції.

Для того щоб відтворювати більш складні конструкції з двома підлеглими вузлами, треба визначити управляючі символи, віднісши решту символів до символів заповнення реконструйованого рядка. Управляючі символи повинні дозволяти відтворення будь-яких конструкцій будь-якої комп'ютерної мови, а також спрощувати еквівалентні перетворення істотно різного синтаксису семантично еквівалентних конструкцій для будь-якої пари комп'ютерних мов. Для реалізації цього необхідно обрати символи управління конструкцій повторення синтаксичних елементів та символи управління умовно еквівалентними конструкціями різних мов:

- `\1` – для створення невідтворюваного еквівалента літери `f`,
- `\2(377)` – для встановлення прапорця або лічильника,
- `\3(376)` – для знищення прапорця або лічильника,
- `\4` – для відтворення аргументу без дужок,
- `\5` – для відтворення аргументу з дужками,
- `\6` – для повернення до номера попереднього аргументу, наприклад при відтворенні накопичувальних присвоювань та декларацій мови С в мові Pascal),
- `\7` – для створення нульового прапорця при першому вході,
- `\11=\t` – для керованої табуляції,
- `z` – ознаку пропуску першого аргументу.

Запропоновані вище коди управління можна замінити іншими кодами однозначної системи кодування. Для інших варіантів відтворення будується масив рядків `cpr[]`, в якому розміщуються шаблони реконструкції з літерами управління та іншими літерами, які відтворюються при реконструкції без змін. Для мови С змістовна частина такого масиву має вигляд.

```
char *cprC[]={"", "", "", "", "\1\5y", "", "", "",
 "\7switch\5\n{\1y\377z\1 y", "", "", "\4;\n\376}",
 "","\1x","\1\5y","x\1\5","","\1x","x\1(!\5)","\1\4",
 "\1\5y", "", "", "", "\1(!\4)y", "", "", "",
 "", "", "", "", "\1\4\4", "", "", "",...
 "\4\1y","\4\1y","\4\1y","\4\1y","\4\1y","\4\1y",
 "\4\1y","\4\1y",   "\4\1y","\4\1y","\4\1y",
 "", "", "", "", "", "", "", "", "", "", "",...
};
```

Для відтворення базових функцій вузла будується масив рядків для заміни операції `f`, який для мови С має вигляд

```
  char *oprtrC[]={"", "", "", "",
  "if", "then", "else", "elseif",
  "case", "switch", "defualt", ""/*endcase*/,
  "break", "return", "while", "while", "continue",
  "do", "while", "do", "for", ";", ";", ";",
  "while", "do", "with", "endif",
  "goto", "extern", "var", "const",
  "enum", "struct", "union", "register",//
  "unsigned", "signed", "char", "short",
  "int","long","int64","int64",//
```

```
"float", "double", "void", "auto",
"static", "volatile", "typedef", "sizeof",//
"real", "array", "set", "file",
"object", "string", "label",
"int main()","function", "procedure",
"","","","","","","","",       //V+8
"","","","","","","","","","","", //V+19
"","","","","",             //V+24
"var","","","","","","","",       //V+31
"","","","","","","","","","",    //V+40
"","","","","","",             //V+46
"inline", "forward", "interrupt", "export",
"extern", "_asm", "","","",  //Verilog|SQL+3
"object", "constructor", "destructor",
"property", "resP", "abstract",      //P+9
"class", "public", "private", "protected",
"virtual", "friend",           //C++15
"new","delete","try","catch","throw",//C++20
"\nfork", "join",
"\n{", "}", "{", "}", "[", "]", "(", ")",
",;\n",".;\n",";\n", ",", ":", "?",
"|=", "&=", "^=", "+=", "-=", "*=", "/=", "%=",
"<<=", ">>=", "=", "--", "++",
"<", "<=", "==", "!=", ">=", ">",
"+", "-", "*", "/",
".", "->", "**", "<<<", ">>>", "===", "!==",
"+", "-", "*", "&", "~|", "~&", "~^", "^~", "&",
"%", "|", "&", "^", "<<", ">>", "||", "&&",
```

```
"!","~","/"};
```

Крім того, передбачається використання різних варіантів реконструкції при наступних повтореннях вкладених конструкцій з використанням управляючих кодів спеціальних символів. Так при визначенні повторень фраз операторів мови на прикладі оператора `switch` C/C++ при першій реконструкції внутрішній код `_case` відтворюється як `switch` + `case`, а при наступних реконструкціях просто як `_case`.

Для визначення мінімальної розстановки дужок, а також для аналізу передувань при синтаксичному аналізі використовуються ще два масиви функцій передувань `fpr[]` і `gpr[]`, які за позиціями (номерами елементів) відповідають масивам `oprtrC[]` і `cprC[]`, а за числовими пріоритетами – значенням з таблиці 3.1. З міркувань ідентичності відносних пріоритетів однакових операцій в різних мовах ці таблиці можуть бути єдиними для всіх мов, але, на жаль, цих міркувань додержуються не всі творці мов та систем трансляції.

Коди передувань набувають остаточних значень при проектуванні висхідного синтаксичного аналізатора в наступних лабораторних роботах. В базовій програмі замість кодів передувань, використані числа пріоритетів, які змінюються в зворотному порядку відносно кодів передувань з огляду на те, що більш пріоритетні операції передують, тобто виконуються раніше та мають менший порядковий номер передування ніж низькопріоритетні операції. Більше того для більш загальної обробки за таким поданням ви-

користовуються дві функції пріоритетів f_{prC} і g_{prC}. Частину елементи їх табличного визначення наведено нижче:

```
char fpr[]=
{0,0x4f,0x4f,0x4f,0x46,0x11,6,0x12,
 0x13,0x4e,0x12,0x1, 0x4e,0x4e,0x4e,0x4e,
 0x4e,0x4e,0x13,0x4e, 0x4e, 0x9, 0x9, 0x9, 0x4e,...
 0x4e,0x4e,0x4e,0x4e, 0x42, 0x2,
 0x42,2,0x43,3, 0x44,4,0x45,5,
 0x1, 0x1, 0x1, 0x2, 0x2, 0x43,
 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
 0x10, 0x10, 0x10, 0x3C,0x3C,
 0x1A, 0x1A, 0x1C, 0x1C, 0x1A, 0x1A,
 0x20, 0x20, 0x30, 0x30,
 0x3E,0x3E, 0x34,0x1E,0x1E,0x1C, 0x1C,
 0x3C,0x3C,0x3C,0x3C, 0x16,0x18,0x17,0x17, 0x4e,
 0x30,0x16,0x18, 0x17, 0x1E, 0x1E, 0x13, 0x15,
 0x4e,0x4e,0x30},
gpr[]=
{0, 0x4f, 0x4f, 0x4f, 6,0x12,6,0x12,
 0x13,0x4e,0x12,0x1, 0x12,0x1,0x4e,0x4e,
 0x11,0x1,0x1,0x11, 0x11, 0x9, 0x9, 0x9, 0x11,...
0x01,0x01,0x01,0x01, 0x2, 0x2,
 2,2, 3,3, 4,4, 5,5, 1,1, 1, 2,2, 0x43,
 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
 0x10, 0x10, 0x10, 0x3C,0x3C,
 0x1A, 0x1A, 0x1C, 0x1C, 0x1A, 0x1A,
 0x20, 0x20, 0x30, 0x30,
 0x3E,0x3E, 0x34,0x1E,0x1E,0x1C, 0x1C,
```

```
0x3C,0x3C,0x3C,0x3C, 0x16,0x18,0x17,0x17, 0x4e,
0x30,0x16,0x18,0x17, 0x1E, 0x1E, 0x13, 0x15,
0x4e, 0x4e, 0x30};
```

Таблиця 7.1

Таблиця передувань операцій в поширених мовах програмування і моделювання

Характеристики операцій			Позначення в поширених мовах / передування		
					Verilog
Назва	**Категорія**	**Типи аргументів**	**Pascal**	**C**	**HDL**
Впорядкування обчислень	Зміна порядку	Будь-які	(...) /0	(...) /0	(...) /0
Функції, процедури, задачі	Звертання до кодів і даних	Будь-які аргументи	(..., ...) /0	(..., ...) /0	(..., ...) /0
Масиви		Цілі індекси	[*ціле*,...]/0	[*ціле*] /0	[*ціле*] /0
Конкатенації		Будь-які аргументи	-	-	{...,...}/0
Доступ до елементів структур		За іменем структури	.	. /0	. /0
		За покажчиком	^.	–> /0	-
Покажчики	Унарні	Будь-які	@ /1	& /1	-
Доступ за покажчиком	Унарні	Вказівники	^ /1	*= /1	-
Інверсія	Унарні	Бітові	not /1	~ /1	~ /1
Інверсія	Унарні	Логічні	not /1	! /1	! /1
Інкремент	Унарні арифметичні	Цілі та вказівники	-	++ /1	-
Декремент		Цілі та вказівники	-	— /1	-
Зміна знаку		Числові	– 1	– /1	– /1
Повтор знаку		Числові	+ 1	+ /1	+ /1
Піднесення до ступеню	Експонентні	Числові	^ /2	-	** /2
Множення	Бінарні мультиплікативні	Числові	* /2	*= /3	* /3
Ділення		Дійсні	/ /2	/= /3	/ /3
Ціле ділення		Цілі	div /2	/= /3	/ /3
Модуль		Цілі	mod /2	%= /3	% /3
Додавання	Бінарні адитивні	Числа та вказівники	+ /3	+= /4	+ /4
Віднімання		Числа та вказівники	– /3	–= /4	– /4
Входження	Бінарні	Множинні	in /4	-	-
Зсув вліво	Бінарні бітові	Цілі або бітові	shl /2	<<= /5	<< /5
Зсув вправо		Цілі або бітові	shr /2	>>= /5	>> /5
Зсув вліво	Бінарні арифметичні	Цілі або бітові	-	-	<<< /5
Зсув вправо		Цілі або бітові	-	-	>>> /5
Менше		Числа або рядки	< /4	< /6	< /6

Більше	Відно-шення чисел і рядків	Числа або рядки	> /4	> /6	> /6
Не менше		Числа або рядки	<= /4	<= /6	<= /6
Не більше		Числа або рядки	>= /4	>= /6	>= /6
Рівність	Відно-шення	Будь-які однотипні	= /4	== /7	== /7
Нерівність		Будь-які однотипні	<> /4	!= /7	!= /7
Рівність	Відно-шення	Чотиризначні бітові	-	-	=== /7
Нерівність		Чотиризначні бітові	-	-	!== /7
Кон'юнкція	Унарні згортки чотири-значних бітових даних	Бітові або цілі	-	-	& /8
Інверсія кон'юнкції		Бітові або цілі	-	-	~& /8
Додавання за модулем 2		Бітові або цілі	-	-	^ /9
Рівнозначність		Бітові або цілі	-	-	~^ ^~ /9
Диз'юнкція		Бітові або цілі	-	-	| /10
Інверсія диз'юнкції		Бітові або цілі	-	-	~| /10
Кон'юнкція	Бінарні	Бітові і цілі	and /2	&= /8	& /8
Інверсія кон'юнкції*		Бітові і цілі	-	-	~& /8
Додавання за модулем 2		Бітові, цілі і логічні	xor /3	^= /9	^ /9
Рівнозначність*		Бітові, цілі і логічні	-	-	~^ ^~ /9
Диз'юнкція		Бітові і цілі	or 3	|= 10	| /10
Інверсія диз'юнкції*		Бітові і цілі	-	-	~| /10
Кон'юнкція		Логічні	and 2	&& 11	&& /11
Диз'юнкція		Логічні	or 3	|| 12	|| /12
Умовний вираз	Тернарна	будь-які? Числові	-	?: 13	?: /13
Присвоювання	Бінарні	Будь-які	:= 14	= /14	= /14
Перелік даних	Список	будь-які значення	-	, /15	, /15
Перелік дій	Список	будь-які вирази і дії	-	; /16	; /16

Примітка: знак рівності в індексі характеристик операцій мови С позначає припустимість накопичуючого присвоювання, яке в мові Pascal визначається в формі: *змінна* := *змінна операція вираз*; і для яких в мові С припустима форма: *змінна операція* = *вираз*; зі значенням передування для комбінованої операції 14. Зірочкою * помічені логічно можливі операції.

Програмна реалізація відтворення текстів в довільній комп'ютерній мові

В різних комп'ютерних мовах семантично еквівалентні оператори можуть відтворюватися по-різному. Тому для уніфікованої реконструкції операторів різних мов треба визначити, яким уніфікованим позначенням з перенумерованого типу **enum** tokType відповідають рядки форматів відтворення з масиву, створеного для відповідної мови **char** *oprtrC[] для мови C/C++ або для мови Pascal **char** *oprtrP[], і що робити, коли однозначно еквівалентні засоби в мові відтворення відсутні.

Ключовим моментом є використання рекурсивних звернень до базової функції обходу графа prLxTxt з виходом при досягненні термінальних вузлів. З використанням цього прийому функцію відтворення можна створити у наступному вигляді.

```
void  prLxTxt(struct lxNode*rt) //корінь піддерева
{static int  // лічильники входжень
 struct lxNode* rt0; // робочий вказівник
 char n=0, c, bC=0, opCnt=0;
 if(rt->ndOp<=_cnst)
 {if(rt->ndOp!=_nil)// обробка термінального операнда
    {if(mode==1&&rt->ndOp<begOprtr-8)printf(" ");
     printf("%s",rt->prvNd);
     mode=1;
 }}// вихід з рекурсії
 else
 while((c=cpr[rt->ndOp][n])!=0)// перегляд шаблону
```

```
 {n++;    // просування по шаблону
  switch(c)      // аналіз літери шаблона
  {case 7:if(mdCnt==0)mdCnt=-1; else
    while(c!=-1)c=cpr[rt->ndOp][n++];
    c=0; break;
   case 6: opCnt--;// повернення до першого аргументу
   case -1: break;
   case -2: mdCnt=0; break;
   case 1:
    if(mode!=0&&rt->ndOp>=_if&&rt->ndOp<begOprtr-8)
      printf(" ");
    printf("%s",oprtr[rt->ndOp]);
    if(rt->ndOp>=begOprtr-8)mode=0;
    else mode=1; break;
   case 'x': case 'y': case 5:case 4:
    if(opCnt)rt0=rt->pstNd;// вибір аргументу
    else rt0=rt->prvNd;//перевірка потреби обрамлення
    if(c=='y')bC=fpr[rt->ndOp]>fpr[rt0->ndOp]&&rt0;
    else if(c==5)bC=1; else if(c==4)bC=0;
    else bC=fpr[rt->ndOp]>=fpr[rt0->ndOp]&&rt0;
    if(bC)prOpBr(rt0); // обрамлення дужками
    prLxTxt(rt0);  // рекурсивний виклик відтворення
    if(bC)prClBr(rt0); // обрамлення дужками
   case 'z': prLxTxt(rt->pstNd); break;
   default: printf("%c",c);
  }if(c==-1&&mdCnt!=0)break; //
 }}
}
```

Такі функції реконструкції складають основу програм ***конверторів***, що перетворюють тексти з однієї до іншої мови приблизно одного рівня. Для їх простої реалізації необхідна достатня схожість відповідних типів даних і наявність у форматі внутрішнього подання всіх операторів і операцій вихідної мови.

Структура програмного шаблона виконання роботи

Для спрощення розв'язання задачі слід використовувати прототип проекту spLb3.dsp, побудований на базі проекту apLb1.dsp і зберігається в робочому просторі apLb1.dsw в підпапці spLb3 папки spLb1, яка зберігає шаблони прототипів для всіх лабораторних робіт циклу «Алгоритми і структури даних». До складу проекту консольної прикладної програми, орієнтованої на можливість використання в числі інших бібліотек стандартних функцій та об'єктів MFC (Microsoft Foundation Classes) входять модулі з наступними вхідними файлами заголовків і реалізацій:

- StdAfx.h і StdAfx.cpp – модуль для організації використання об'єктів класів MFC;
- token.h і token.cpp – модуль для опису кодування типу лексеми, структури вузла лексеми і організації використання структур або класів лексем в процесі лексичного і синтаксичного аналізу;
- visgrp.h і visgrp.cpp – модуль для відображення рядків або повних таблиць;

- parsP.cpp, parsC.cpp і parsV.cpp – модулі констант для відтворення лексем мов Pascal, C/C++ та Verilog HDL;
- automat.h і automat.cpp – модуль для визначення та управління автоматом;
- spLb3.cpp – модуль контрольних прикладів для тестування функцій обробки графів.

Розділ визначень і декларацій основної частини програми тестування в модулі spLb3.cpp повинен включати визначення послідовностей вузлів, що утворюють графи лексем. Розділ визначень модуля visgrp.cpp забезпечує визначення мов реконструкції за варіантом завдання. Виконавча частина модуля spLb3.cpp програми повинна включати циклічно повторювані виклики функцій реконструкції для різних варіантів операторів і мов.

Завдання на роботу

Завдання на підготовку до роботи на комп'ютері

1. Визначити варіант завдання для основних задач за таблицею 3.1. Визначити приклади лексем через константи в модулі тестування spLb3.cpp.

2. Ознайомитись з шаблоном програмного проекту spLb3.dsp. Настроїти відповідні дані символьних позначень та структур вузлів графів в програмному проекті на мові С.

3. Підготовити настройку управляючих таблиць мов програмування `oprtrC[]` або `oprtrP[]`, `cprC[]` або `cprP[]`, `fpr[]` і `gpr[]`, а також згідно з варіантами, заданими в табл. 7.1. Також

згідно з варіантами визначити масиви лексем **struct** lxNode token[] і образів **char** *imgs[] для оператора, заданого у варіанті.

Таблиця 7.2

Варіанти завдань для виконання реконструкцій і роботи з графами

№ вар.	Вираз, який відтворюється в графі внутрішнього подання	Мова відтворення
1	**if**(c)b=(2*a +c/d)*2*a;	C
2	**if** c<>0**then**b:=(2*a+c)*2*a;	Pascal
3	b+=a[n] * (n--);	C
4	n:=n-1; b:=b+a[n]	Pascal
5	b+=a[--n]/n;	C
6	b:=b+a[n]; n:=n-1;	Pascal
7	**if**(c)b=sin(2*a);**else** b=2*a;	C
8	**if** c **then** b:=sin(2*a) **else** b:=a;	Pascal
9	b=c?d:2*a[n];	C
10	**if** c!=0 **then** b:=d **else** b:=2*a[n];	Pascal
11	b=2*a[n]; b=d**;**	C
12	: b:=2*a[n]; b:=d	Pascal
13	**for**(b=0;n;n--)b+=a[n];	C
14	b:=0; b:= b+a[n];	Pascal
15	**if**(b==a[n]) --n;	C
16	b1:= n=0 **or** b!=a[n];	Pascal
17	**if**(b==a[n]) n++**;**	C
18	**if** a-c=0 **then** b:=(a-c)*2*a;	Pascal
19	b=n++&&b==a[n];	C
20	b=--n&&b==a[n];	C

4. Підготувати програмний модуль контрольної задачі, який виконує задані варіанти програм реконструкції і управління станом автомата і дозволяє перевірити коректність виконання програми і її окремих модулів.

Завдання на роботу на комп'ютері

5. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

6. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми для перевірки результатів контрольних прикладів.

7. Одержати результати виконання, проаналізувати їх і зробити висновки.

Порядок вибору варіанту:

За останньою цифрою номера залікової книжки або за порядковим номером студента в списку підгрупи з доданим номером групи визначте за табл. 7.2 варіант оброблюваних даних та настройки програм за прикладом.

Питання для самоперевірки

1. Дайте визначення основних типів внутрішнього подання у вигляді графів даних, які визначають процес обробки виразів та операторів.
2. Якими структурами даних або об'єктами визначається вузол внутрішнього подання?
3. Як реалізуються зв'язки з підлеглими вузлами виразів в поданні графа?
4. Як відтворюються вузли роздільників і зв'язки з підлеглими вузлами в поданні графа?
5. Як відтворюються вузли ключових слів і зв'язки з підлеглими вузлами в поданні графа?

6. Як відтворюються вузли термінальних позначень і розміщуються дані в графі внутрішнього подання?
7. Які загальні альтернативи полям вказівникам вузлів графу можна використовувати при побудові внутрішнього подання?
8. Які типи даних використовуються для визначення станів і сигналів управління автоматів?
9. Якими масивами можна програмно визначити скінченний автомат?
10. Якими операторами мов програмування організуються переходи автомату з одного стану до іншого?

8. Лабораторна робота 8

Тема роботи: Побудова та настроювання програм обходу дерев та графів

Мета роботи: Виконання побудови програм аналізу структури та характеристик вузлів дерев або спрямованих ациклічних графів для розрахунків вторинних характеристик вузлів графів або дерев, що відображують них програм і дерев підлеглості з запам'ятовуванням типів піддерев або підграфів як результатів аналізу кожного графа внутрішнього подання текстової інформації. Вивчення механізмі параметричного запуску підлеглих функцій змістовної (семантичної) обробки за деревами підлеглості операцій і операторів та спрямованими ациклічними графами (САГ).

Короткі теоретичні відомості

На основі попередньо визначених полів базової структури елемента внутрішнього подання лексеми треба організувати поступове заповнення полів типів і характеристик піддерев на етапі семантичної обробки текстів на будь-яких комп'ютерних та природних мовах. Це надасть можливість використання даних попередніх етапів обробки на наступних етапах семантичної обробки. Важливо розпізнавати лексеми-термінали і лексеми-нетермінали графа. Нетермінальні вузли графа підлеглості повинні мати можливість використання у формі кореневих вузлів як бази для обходу дерев та піддерев.

Заповнення таблиць і правил для семантичного аналізу

Семантичний аналіз, який є попереднім етапом для всіх видів семантичної обробки, призначений для визначення змістовної суперечності або коректності тексту на комп'ютерній мові. Вхідними даними семантичного аналізу є дерева підлеглості операцій з полями для зберігання типів даних результатів кожного піддерева внутрішнього подання програми. Управляючими даними для визначення типів результатів є таблиці відповідності операндів та операцій. Результатом семантичного аналізу є також визначення характеристик типів результатів операцій та операторів на синтаксичному дереві розбору або спрямованому графі внутрішнього подання.

Гнучкість семантичного аналізу забезпечується як табличною організацією обробки складних типів і типів, що визначаються користувачами, так і табличною організацією аналізу відповідності операндів і результатів використаним операціям і операторам мови. Для виконання семантичного аналізу важливо визначити раціональний підхід для кодування типів даних. З одного боку потрібно визначити нумерацію системних типів, складених з декількох ключових слів, що визначається перенумерованим типом **`enum`** `datType`, з другого треба визначити лічильник рівня вкладеності покажчиків, за одиницю якого обираємо константу `cdPtr`, з третього – треба визначити поточне значення модифікатора для заданого типу з набору визначень модифікаторів.

Вимоги до кодування внутрішнього подання типів:

- однозначний код для будь-яких типів з різними варіантами модифікаторів;
- легкість розбиття елементів коду на окремі поля;
- легкість визначення або підрахунку номера типу користувача та типу і рівня покажчика та забезпечення загальної кількості типів.

Приклад реалізації кодування внутрішнього подання типів для занять з комп'ютерного практикуму включає базовий перенумерований тип, який фактично задає розбиття на три поля кодування.

Номер рівня покажчика (12 бітів)	Ознака константи (1 біт)	Тип пам'яті (3 біти)	Ознака масиву (1 біт)	Код типу користувача (3 біти)	Номер типу користувача (12 бітів)

```
enum datType//кодування типів даних в семантичному
аналізі
{_v,                  // порожній тип даних
 _uc=4,_us,_ui,_ui64,// стандартні цілі без знака
 _sc=8,_ss,_si,_si64,  // стандартні цілі зі знаком
 _f,_d,_ld,_rel,    // дані з плаваючою точкою
 _lbl,           // мітки
// інші стандартні типи
 _geq = 0x0ffe,     // загальний тип для рівності
 _gen = 0x0fff,     // загальний (довільний) тип
 _enm = 0x1000,     // перенумеровані типи enum
 _str = 0x2000,     // структурні типи /*_record*/,
 _unn = 0x3000,     // типи об'єднань union
```

```
 _cls = 0x4000,     // типи класів
 _obj = 0x5000,     // типи об'єктів
 _fun = 0x6000,     // функціональні типи
 _ctp = 0x7000,     // умовні типи мови Pascal
 _fl, _tp, _vl, _vr,   //
};
// модифікатори кодів типів мови C/C++
#define cdPtr 0x100000// код покажчика 1-го рівня
#define cdCns 0x080000// код константного типу даних
#define cdArr 0x108000// код даних типу масиву
#define cdCna 0x188000// код константного масиву
#define cdReg 0x010000// код регістрового типу даних
#define cdExt 0x020000// код зовнішнього типу даних
#define cdStt 0x030000// код статичного типу даних
#define cdAut 0x040000// код автоматичного типу даних
#define cdVlt 0x070000// код примусового типу даних
```

Таке визначення базових типів, покажчиків, масивів та модифікаторів кодів дозволяє гнучко визначати практично будь-який тип в рамках однієї системи кодування і визначати при цьому до 2^{12} варіантів типів користувача.

Таблиці семантичного аналізу включають механізми визначення кодів типів даних за допомогою перенумерованого типу **enum** datType та констант модифікаторів типів. Коди типів даних визначаються в операторах обробки декларацій і використовують такі основні таблиці: визначення типів з модифікаціями; управління семантичною обробкою операцій.

Елементи основних таблиць визначаються наступними структурами:

```
// Елемент таблиці модифікованих типів
struct recrdTPD  // структура рядка таблиці
                         // модифікованих типів
{enum tokType kTp[3];//примірник структури ключа
 unsigned  dTp;//примірник функціональної частини
 unsigned  ln;   // довжина даних типу
};
struct recrdSMA
              // структура рядка таблиці припустимості
                         // типів для операцій
{enum tokType oprtn;// код операції
 int oprd1,ln1;//код типу та довжина 1-го арг.
 int oprd2,ln2;//код типу та довжина 2-го арг.
 int res, lnRes;//код типу та довжина результату
 _fop*pintf; // покажчик на функцію інтерпретації
 char  *assCd;
};
```

Базові типи, що визначаються ключовими словами, показані в таблиці характеристик типів відносно коду першого ключового слова, що визначає тип. Елементи цієї таблиці мають структуру.

```
struct recrdTMD// структура рядка таблиці
базових типів
{enum datType tpLx;// примірник структури ключа
 unsigned  md;   // модифікатор
```

```
 unsigned  ln;   // базова або гранична довжина даних
типу
};
```

А сама таблиця для мови C/C++ має вигляд.

```
struct recrdTMD tpLxMd[]=
          // масив кодів та ознак ключових слів типів
{{_v, 0, 0},     //0 _void
 {_v, 0, 0},     //1 _extern
 {_v, 0, 0},     //2 _var
 {_v,cdCns,0},   //3 _const
 {_enm, 0,32},   //4 _enum
 {_str, 0, 0},   //5 _struct/*_record*/
 {_unn, 0, 0},   //6 _union
 {_v,cdReg,0},   //7 _register
 {_ui,0,32},     //8 _unsigned
 {_si,0,32},     //9 _signed
 {_si,0,8},      //10 _char
 {_si,0,16},     //11 _short
 {_si,0,32},     //12 _int
 {_si,0,32},     //13 _long
 {_si,0,64},     //14 _sint64
 {_ui,0,64},     //15 _uint64
 {_f,0,32},      //16 _float
 {_d,0,64},      //17 _double
};
```

Таблиця типів, що визначаються декількома словами для мови C/C++, має наступний вигляд.

```
struct recrdTPD tpTbl[]= // таблиця модифікованих типів
{{{_void,_void,_void},_v,0},
 {{_enum,_void,_void},_enm,32},
 {{_struct,_void,_void},_str,0},
 {{_union,_void,_void},_unn,0},
 {{_unsigned,_void,_void},_ui,32},
 {{_signed,_void,_void},_si,32},
 {{_char,_unsigned,_void},_uc,8},
 {{_char,_signed,_void},_sc,8},//4
 {{_char,_void,_void},_sc,8},
 {{_short,_void,_void},_si,16},
 {{_short,_unsigned,_void},_ui,16},
 {{_short,_signed,_void},_si,16},
 {{_int,_void,_void},_si,32},//9
 {{_int,_unsigned,_void},_ui,32},
 {{_int,_signed,_void},_si,32},
 {{_int,_long,_void},_si,32},
 {{_long,_void,_void},_si,32},
 {{_float,_void,_void},_f,32},//14
 {{_double,_void,_void},_d,64},
 {{_double,_long,_void},_ld,80},
 {{_class,_void,_void},_cls,0},
};
```

Якщо таблицю типів розширити константними, регістровими та зовнішніми типами, то в кожному з їх елементів в останньому елементі ключа додається відображення першого ключового слова-модифікатора з потроєнням загального обсягу таблиці.

Частина таблиці припустимості для типів операндів для типових комбінацій вузлів графів для оператора `if`, закодованого значенням `_if`, та операції накопичувального присвоювання +=, закодованої значенням `_asAdd`, має вигляд.

```
struct recrdSMA  ftTbl[]=
          // таблиця припустимості типів для операцій
{{_if,_ui,32,_v,0,_v,0},
 {_if,_ui,32,_ui,32,_v,0},
 {_if,_ui,32,_si,32,_v,0},
 {_if,_ui,32,_f,32,_v,0},
 {_if,_ui,32,_d,64,_v,0},
 {_if,_si,32,_v,0,_v,0},
 {_if,_si,32,_ui,32,_v,0},
 {_if,_si,32,_si,32,_v,0},
 {_if,_si,32,_f,32,_v,0},
 {_if,_si,32,_d,64,_v,0},
 {_if,_f,32,_v,0,_v,0},
 {_if,_f,32,_ui,32,_v,0},
 {_if,_f,32,_si,32,_v,0},
 {_if,_f,32,_f,32,_v,0},
 {_if,_f,32,_d,64,_v,0},
 {_if,_d,64,_v,0,_v,0},
 {_if,_d,64,_ui,32,_v,0},
 {_if,_d,64,_si,32,_v,0},
 {_if,_d,64,_f,32,_v,0},
 {_if,_d,64,_d,64,_v,0},
 {_if,_ui|cdPtr,32,_v,0,_v,0},
```

```
{_if,_ui|cdPtr,32,_ui,32,_v,0},
{_if,_ui|cdPtr,32,_si,32,_v,0},
{_if,_ui|cdPtr,32,_f,32,_v,0},
{_if,_ui|cdPtr,32,_d,32,_v,0},
{_if,_si|cdPtr,32,_v,0,_v,0},
{_if,_si|cdPtr,32,_ui,32,_v,0},
{_if,_si|cdPtr,32,_si,32,_v,0},
{_if,_si|cdPtr,32,_f,32,_v,0},
{_if,_si|cdPtr,32,_d,32,_v,0},
{_if,_ui|cdPtr|cdArr,32,_v,0,_v,0},
{_if,_ui|cdPtr|cdArr,32,_ui,32,_v,0},
{_if,_ui|cdPtr|cdArr,32,_si,32,_v,0},
{_if,_ui|cdPtr|cdArr,32,_f,32,_v,0},
{_if,_ui|cdPtr|cdArr,32,_d,32,_v,0},
{_if,_si|cdPtr|cdArr,32,_v,0,_v,0},
{_if,_si|cdPtr|cdArr,32,_ui,32,_v,0},
{_if,_si|cdPtr|cdArr,32,_si,32,_v,0},
{_if,_si|cdPtr|cdArr,32,_f,32,_v,0},
{_if,_si|cdPtr|cdArr,32,_d,32,_v,0},
{_if,_f|cdPtr,32,_v,0,_v,0},
{_if,_f|cdPtr,32,_ui,32,_v,0},
{_if,_f|cdPtr,32,_si,32,_v,0},
{_if,_f|cdPtr,32,_f,32,_v,0},
{_if,_f|cdPtr,32,_d,64,_v,0},
{_if,_d|cdPtr,64,_v,0,_v,0},
{_if,_d|cdPtr,64,_ui,32,_v,0},
{_if,_d|cdPtr,64,_si,32,_v,0},
{_if,_d|cdPtr,64,_f,32,_v,0},
```

```
{_if,_d|cdPtr,64,_d,64,_v,0},
{_else,_v,0,_ui,32,_v,0},
    . . .
{_asXor,_si,32,_si,32,_si,32},
{_asAdd,_ui,32,_ui,32,_ui,32},
{_asAdd,_ui,32,_si,32,_ui,32},
{_asAdd,_ui,32,_f,32,_ui,32},
{_asAdd,_ui,32,_d,32,_ui,32},
{_asAdd,_si,32,_ui,32,_si,32},
{_asAdd,_si,32,_si,32,_si,32},
{_asAdd,_si,32,_f,32,_si,32},
{_asAdd,_si,32,_d,32,_si,32},
{_asAdd,_f,32,_ui,32,_f,32},
{_asAdd,_f,32,_si,32,_f,32},
{_asAdd,_f,32,_f,32,_f,32},
{_asAdd,_f,32,_d,64,_f,32},
{_asAdd,_d,64,_ui,32,_d,32},
{_asAdd,_d,64,_si,32,_d,64},
{_asAdd,_d,64,_f,32,_d,64},
{_asAdd,_d,64,_d,64,_d,64},
{_asAdd,_ui+cdPtr,32,_ui,32,_ui+cdPtr,32},
{_asAdd,_ui+cdPtr,32,_si,32,_ui+cdPtr,32},
{_asAdd,_si+cdPtr,32,_ui,32,_si+cdPtr,32},
{_asAdd,_si+cdPtr,32,_si,32,_si+cdPtr,32},
{_asAdd,_f+cdPtr,32,_ui,32,_f+cdPtr,32},
{_asAdd,_f+cdPtr,32,_si,32,_f+cdPtr,32},
{_asAdd,_d+cdPtr,32,_ui,32,_d+cdPtr,32},
{_asAdd,_d+cdPtr,32,_si,32,_d+cdPtr,32},
```

```
{_asSub,_ui,32,_ui,32,_ui,32},
    . . .
```

Елементи наведеної частини таблиці визначають відповідності аргументів і результатів для всіх числових типів даних та покажчиків на дані цих типів.

Структура програмного шаблона виконання роботи

Для спрощення розв'язання задачі слід використовувати прототип проекту цього практичного заняття spLb7.dsp, який зберігається в робочому просторі apLb1.dsw в папці apLb1, яка зберігає шаблони прототипів для всіх лабораторних занять циклу «Алгоритми і структури даних». До складу проекту консольної прикладної програми, орієнтованої на можливість використання в числі інших бібліотек стандартних функцій входять модулі з наступними вхідними файлами заголовків і реалізацій:

- StdAfx.h і StdAfx.cpp – модуль для організації використання об'єктів класів MFC;
- langio.cpp – модуль для введення-виведення даних транслятора.
- tables.h і tables.cpp – модуль для опису і організації використання структур або класів таблиць;
- index.h і index.cpp – модуль для побудови впорядковуючого індексу для образів імен і констант;
- lexan.h і lexan.cpp – модуль для виконання лексичного аналізу;

- token.h і token.cpp – модуль для опису кодування типу лексеми, структури вузла лексеми і організації використання структур або класів лексем в процесі лексичного і синтаксичного аналізу;
- visgrp.h і visgrp.cpp – модуль для відображення рядків або повних таблиць;
- parsP.cpp, parsC.cpp і parsV.cpp – модулі констант для відтворення лексем мов Pascal, C/C++ та Verilog HDL, які використовують однойменні модулі проекту spLb3.dsp;
- spLb7.cpp – модуль контрольних прикладів для тестування функцій обробки графів.

Розділ визначень і декларацій основної частини програми тестування в модулі spLb7.cpp повинен включати виклик функцій ініціалізації таблиць для потрібної мови їх елементів, спеціальні змінні, що зберігають довжину (тобто кількість елементів) таблиці. Розділ визначень модуля visgrp.cpp забезпечує визначення мов реконструкції за варіантом завдання.

Зверніть увагу, що таблиця визначення типів результатів `struct recrdSMAftTbl[179]` в модулі реалізації таблиць semanT.cpp в проекті spLb7.dsp заповнена частково для можливості додавання елементів, потрібних за варіантом.

Завдання на роботу

Завдання на підготовку до роботи на комп'ютері:

1. Визначити варіант завдання для основних задач за таблицею 7.1. Визначити приклади лексем через файл в папці spLb7 модуля тестування spLb7.cpp.

2. Відповісти на контрольні запитання.

3. Підготувати настройки вхідної мови програмування.

4. Використати структуру елементу **`struct`** `lxNode` з файлу index.h шаблону програмного проекту spLb7 для побудови елементу індексу таблиць лексем і визначити поля, що заповнюються при семантичному аналізі.

5. Підготувати програмний модуль контрольної задачі, який виконує заданий варіант з табл. 8.1 і дозволяє перевірити коректність виконання програм. Для цього доповнити таблиці відповідності типів результатів типам операндів **`struct`** `recrdSMAftTbl[179]` в модулі реалізації таблиць semanT.cpp для семантичного аналізу, доповнивши її потрібними для варіанта елементами.

Порядок вибору варіанту:

За останньою цифрою номера залікової книжки або за порядковим номером студента в списку підгрупи з доданим номером групи визначте за табл. 8.1 варіант оброблюваних даних та настройки програм за прикладом.

Завдання на роботу на комп'ютері

6. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

7. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми для перевірки результатів контрольних прикладів, заданих в варіантах з табл. 8.1, що включають помилкові покажчики в описах даних.

8. Одержати результати виконання, проаналізувати виникнення діагностичних повідомлень в них в режимі налагодження і зробити висновки.

Таблиця 8.1

Варіанти завдань для виконання аналізу файлу лексем

№ вар.	Вираз, який відтворюється в графі внутрішнього подання	Мова відтворення
1	**float** b, a, c, *d; b=(2*a +c/d)*2*a;	C
2	**double** *b, a[5]; **int** n; b:=(2*a+c)*2*a;	Pascal
3	**float** *b, a[4]; **int** n; b+=a[n];	C
4	**float** *b, a[3]; **int** n; n:=n-1; b:=b+a[n] **;**	Pascal
5	**float** *b, a[5]; **char** n; b+=a[--n];	C
6	**double** *b, a[4]; **char** n; b:=b+a[n]; n:=n-1 ; n=0;	Pascal
7	**double** *a; **int** b; b=sin(2*a); b=2*a;	C
8	**double** b; **unsigned** *a; b:=sin(2*a); b:=a;	Pascal
9	**float** b, *d, a[5]; **int** c; b=c?d:2*a[n];	C
10	**double** b, a[4]; **unsigned** n,*d; b:=d!=0**;** b:=2*a[n];	Pascal
11	**double** b, a[4]; **short** *n,d; b=2*a[n]; b=d**;**	C
12	**float** b, a[3]; **unsigned** n,*d; b:=2*a[n]; b:=d;	Pascal
13	**float** *b, a[3]; **short** n,d; b=0;b+=a[n];	C
14	**float** b, a[5]; **short** *n; b:=0; n:=n; b:= b+a[n];	Pascal
15	**float** a[3] ,*c; **int** b, n; --n; b=c==a[n]**;**	C
16	**float** b, a[3]; **long** n; n:=n-1; b:=0; b:=n!=a[n]);	Pascal
17	**double** b, a[3]; **long** n; --n; b=n==a[n];	C
18	**double** b, a[3]; **long** *n; b:=a-c=0; c:=(a-c)*2*a;	Pascal
19	**double** b, a[6]; **char** *n; --n; b=n&&b==a[n];	C

20	**double** *b, a[3]; **short** n; b=--n&&b==a[n]);	C
21	**float** *b, a[2]; **long** n; b:=n>0 **and** b=a[n]; n:=n-1;	Pascal
22	**double** b, a[3]; **long** *n; b=c<a; b=sin(2*a); b=2*a;	C

9. Одержати результати виконання, проаналізувати коректність сформованих типів результатів в режимі налагодження і зробити висновки.

10. Продемонструвати результати викладачам

Контрольні запитання про особливості семантичного аналізу

1. Як визначають в компіляторах основні типи даних?

2. Яким чином забезпечується відповідність типів даних операндів та результатів операцій при семантичному аналізі?

3. Які поля повинні входити до таблиць семантичного аналізатора при реалізації комп'ютерної мови?

4. Які поля структур лексем термінальних вузлів заповнюються на етапі семантичного аналізу?

5. Які поля структур лексем вузлів роздільників та ключових слів заповнюються на етапі семантичного аналізу?

6. Які дані включаються до таблиць семантичної відповідності операндів?

7. Як організується рекурсивний алгоритм узагальненої семантичної обробки у застосуванні до семантичного аналізу?

8. Які особливості необхідно визначити для семантичного аналізу операцій присвоювання?

9. Які таблиці треба використовувати для визначення типів даних в операторах описів типів і примірників даних?

10. Які помилки можуть розпізнаватися при семантичному аналізі?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Вирт Н.* Алгоритмы + структуры данных = программы: Пер. с англ. - М.: Мир, 1985. - 406 с., ил.
2. *Вирт Н.* Алгоритмы и структуры данных: Пер. с англ. - М.: Мир, 1989. - 317 с., ил.
3. *Пустоваров В.И.* Ассемблер: программирование и анализ корректности машинных программ. – К: BHV, 2000, 480 с.
4. *Пустоваров В.И.* Язык ассемблера в программировании информационных и управляющих систем. М:. "Энтроп", К: "Век", 1996, 304 с. – К.: Юниор, 1997. – 304 с.
5. *Бек Л.* Введение в системное программирование: Пер. с англ.- М.: Мир, 1988. - 448 с., ил.
6. *Проценко В.С., Чаленко П.Й., Ставровський А.Б.* Техніка програмування мовою Сі: Навчальний посібник. – К.: Либідь, 1993, 224 с.
7. *Ахо А., Сети Р., Ульман Дж.* Компиляторы: принципы, технологии, инструменты: Пер. с англ. – М.: Издательский дом Вильямс, 2001. – 768 с.
8. *Стобо Дж.* Язык программирования Пролог: Пер. с англ. – М.: Радио и связь, 1993. – 386 с.
9. Metzger R.C., Zhaofang W. Automatic algorithm recognition and replacement: a new approach to program optimization / The MIT Press, Cambridge, 2000. 219 p.
10. Muchnick S.S. Advanced compiler design and implementation – San Francisco, Morgan Kaufmann Publishers, 1997. – 856 p.