Зміст

Вступ

1. Бінарні дерева

Відкриття програми

2. Списки

Проводимо компілюцію

3. Вводимо дані стеку

4. Вводимо дані черги

Бінарне дерево

Закінчили виконання практичної частини контрольної роботи

Висновок

Список використаної літератури

# Вступ

В програмуванні та комп'ютерних науках структури даних - це способи організації даних в комп'ютерах. Часто разом зі структурою даних пов'язується і специфічний перелік операцій, які можуть бути виконаними над даними, організованими в таку структуру. Правильний підбір структур даних є надзвичайно важливим для ефективного функціонування відповідних алгоритмів їх обробки. Добре побудовані структури даних дозволяють оптимізувати використання машинного часу та пам'яті комп'ютера для виконання найбільш критичних операцій. Відома формула "Програма = Алгоритми + Структури даних" дуже точно виражає необхідність відповідального ставлення до такого підбору. Тому іноді навіть не обраний алгоритм для обробки масиву даних визначає вибір тої чи іншої структури даних для їх збереження, а навпаки. Підтримка базових структури даних, які використовуються в програмуванні, включена в комплекти стандартних бібліотек найбільш розповсюджених мов програмування, такиї як Standart Template Library для C++, Java API, Microsoft.net

# 1. Бінарні дерева

**Дерево** - в інформатиці та програмуванні одна з найпоширеніших структур даних. Формально дерево визначається як скінченна множина Т з однієї або більше вершин (вузлів, nodes), яке задовольняє наступним вимогам: існує один виокремлений вузол - корень (root) дерева інші вузли (за виключенням кореня) розподілені серед m ≥ 0 непересічних множин T1. Tm і кожна з цих множин в свою чергу є деревом. Дерева T1. Tm мають назву піддерев (subtrees) даного кореня. З цього визначення випливає, що кожна вершина є в свою чергу коренем деякого піддерева. Кількість піддерев вершини має назву ступеня (degree) цієї вершини. Вершина ступеню нуль має назву кінцевої (terminal) або листа (leaf). Некінцева вершина також має назву вершини розгалуження (branch node). Нехай x - довільна вершина дерева з коренем r. Тоді існує єдиний шлях з r до x. Усі вершини на цьому шляху називаються предками (ancestors) x; якщо деяка вершина y є предком x, то x називається нащадком (descendant) y. Нащадки та предки вершини x, що не співпадають з нею самою, називаються власними нащадками та предками. Кожну вершину x, в свою чергу, можна розглядати як корень деякого піддерево, елементами якого є вершини-нащадки x. Якщо вершини x є предком y та не існує вершин поміж ними (тобто x та y з'єднані одним ребром), а також існують предки для x (тобто x не є коренем), то вершина x називається батьком (parent) до y, а y - дитиною (child) x. Коренева вершина єдина не має батьків. Вершини, що мають спільного батька, називаються братами (siblings). Вершини, що мають дітей, називаються внутрішніми (internal). Глибиною вершини x називається довжина шляху від кореня до цієї вершини. Максимальна глибина вершин дерева називається висотою. Якщо існує відносний порядок на піддеревах T1. Tm, то таке дерево називається впорядкованим (ordered tree) або пласким (plane tree). Лісом (forest) називають множину дерев, які не перетинаються. Найчастіше дерева в інформатиці зображують з коренем, який знаходиться зверху (говорять, що дерево в інформатиці "росте вниз"). Важливим окремим випадком кореневих дерев є бінарні дерева, які широко застосовуються в програмуванні і визначаються як множина вершин, яка має виокремлений корінь та два піддерева (праве та ліве), що не перетинаються, або є пустою множиною вершин (на відміну від звичайного дерева, яке не може бути пустим). Важливими операціями на деревах є: ***обхід вершин в різному порядку:***

1***.*** перенумерація вершин,

2. пошук елемента,

3. додавання елемента у визначене місце в дереві,

4. видалення елемента,

5. видалення цілого фрагмента дерева,6. додавання цілого фрагмента дерева,

7. трансформації (повороти) фрагментів дерева, знаходження кореня для будь-якої вершини. Найбільшого розповсюдження ці структури даних набули в тих задачах, де необхідне маніпулювання з ієрархічними даними, ефективний пошук в даних, їхнє структуроване зберігання та модифікація.

**Бінарне дерево:** В програмуванні бінарне дерево - дерево структура даних, в якому кожна вершина має не більше двох дітей. Зазвичай такі діти називаються правим та лівим. На базі бінарних дерев будуються такі структури, як бінарні дерева пошуку та бінарні купи.

**Бінарне дерево** - таке кореневе дерево, в якому кожна вершина має не більше двох дітей. Повне (закінчене) бінарне дерево - таке бінарне дерево, якому кожна вершина має нуль або двох дітей. Ідеальне бінарне дерево - це таке повне бінарне дерево, в якому листя (вершини без дітей) лежать на однаковій глибині (відстані від кореня). Бінарне дерево на кожному n-му рівні має від 1 до 2n вершин.

**Обхід бінарного дерева** Докладніше дивись статтю Обхід дерева. Часто виникає необхідність обійти усі вершини дерева для аналізу інформації, що в них знаходиться. Існують декілька порядків такого обходу, кожний з яких має певні властивості, важливі в тих чи інших алгоритмах: прямий (preorder), центрований (inorder) та зворотній (postorder). Реалізація бінарних дерев Реалізація бінарного дерева. Кожна вершина містить вказівники на праву та ліву дитину (left та right) Існують декілька варіантів конструювання бінарних дерев в залежності від задач, які вирішуються цими структурами та можливостей тої чи іншої мови програмування. Реалізація з використанням вказівників передбачає зберігання в кожній вершині дерева x разом з даними двох полів з вказівниками (правим та лівим) right [x] та left [x] на відповідних дітей цієї вершини. Модифікована реалізація бінарного дерева. Кожна вершина містить також вказівник на батьківську вершину. Також іноді додається вказівник p [x] на батьківську вершину. Це виявляється корисним, коли необхідний швидкий доступ до батьківської вершини та спрощує деякі алгоритми. Іноді достатньо тільки вказівника на батьківську вершину. Взагалі будь-яке орієнтоване дерево можна описати, знаючи тільки зв'язки від дітей до батьківської вершини. Деякі різновиди бінарних дерев (наприклад, червоно-чорні дерева або AVL-дерева), вимагають збереження в вершинах і деякої додаткової інформації. Якщо у вершини відсутня одна чи обидві дитини, відповідні вказівники ініціалізуються спеціальними "пустими значеннями.

**Бінарне дерево на базі масиву.** Бінарні дерева також можуть бути побудовані на базі масивів. Такий метод набагато ефективніший щодо економії пам'яті. В такому представленні, якщо вершина має порядковий номер i, то її діти знаходяться за індексами 2i+1 та 2i+2, а батьківська вершина за індексом ( (i-1) /2) (за умов, що коренева вершина має індекс 0). Інший варіант зберігання дерева в масиві - зберігати індекси дітей.

**Представлення n-арних дерев як бінарних.** Існує єдине та взаємооднозначне відображення довільного впорядкованого дерева в бінарне. Для цього слід послідовно зв'язати усіх дітей кожної сім'ї з першою дитиною та та видалити усі вертикальні з'єднання за виключенням з'єднання батька з першою дитиною в сім'ї. Тобто кожна вершина N впорядкованого n-арного дерева відповідає вершині M деякого бінарного дерева. Ліва дитина вершини M відповідає першій дитині вершини N, а права дитина M відповідає першому з наступних братів N (тобто першому з наступних дітей батька вершини N). Така відповідність має назву природної відповідності між n-арним та бінарним деревом.

**Бінарне дерево пошуку:** Бінáрне дéрево пóшуку (англ. binary search tree, BST) в інформатиці - бінарне дерево, в якому кожній вершині x співставлене певне значення val [x]. При цьому такі значення повинні задовольняти умові впорядкованості: нехай x - довільна вершина бінарного дерева пошуку. Якщо вершина y знаходиться в лівомі піддереві вершини x, то val [y] ≤ val [x]. Якщо у знаходиться у правому піддереві x, то val [y] ≥ val [x]. Таке структурування дозволяє надрукувати усі значення у зростаючому порядку за допомогою простого алгоритма центрованого обходу дерева. Бінарні дерева пошуку набагато ефективніші в операціях пошуку, аніж лінійні структури, в яких витрати часу на пошук пропорційні O (n), де n - розмір масиву даних, тоді як в повному бінарному дереві цей час пропорційний в середньому O (log2n) або O (h), де h - висота дерева (хоча гарантувати, що h не перевищує log2n можна лише для збалансованих дерев, які є ефективнішими на пошукових алгоритмах.).

**Пошук в бінарному дереві**. Процедура пошуку в бінарному дереві отримує на вході значення k, яке необхідно знайти, та вказівник x на корень того піддерева, в якому слід шукати.

SEARCH (x, k) 1 if x = NULL or k = val [x]

2 then return x

3 if k < val [x]

4 then return SEARCH (left [x], k)

5 else return SEARCH (right [x], k)

В процесі пошуку ми рухаємось від кореня, порівнюючи значення k зі значенням в поточній вершині х. Якщо k < val [x], пошук рекурсивно продовжується в лівому піддереві (k може бути тільки там згідно умови впорядкованості), інакше - у правому піддереві. Очевидно, що довжина шляху не перевищує висоти дерева, тому час пошуку є O (h), де h - висота дерева.

**Мінімум, максимум, наступник та попередник.**

Мінімальний елемент в дереві можна знайти, розглянувши послідовно ліве піддерево left [x]:

MINIMUM (x)

1 while left [x] <>NULL

2 do x: =left [x]

3 return x

Процедура знаходження максимуму симетрична. Обидві процедури знову ж таки знаходять елемент за час, пропорційний O (h) Елемент, що є наступним за даним (в сенсі відношення впорядкованості) можна знайти за допомогою такої процедури:

SUCCESSOR (x)

1 if right [x] <> NULL

2 then return MINIMUM (right [x])

3 y: =p [x]

4 while y<>NULL and x = right [y]

5 do x: =y

6 y: =p [y]

7 return y

Процедура знаходження попереднього даному елемента є симетричною. Обидві процедури знову ж таки знаходять елемент за час, пропорційний O (h)

**Додавання елементу.**

Процедура додавання елементу в бінарне дерево T додає елемент, зберігаючи умову впорядкованості.

Параметром тут є вказівник z на нову вершину, в яку поміщене значення val [z], left [z] =right [z] =NULL.

INSERT (T, z)

1 y: =NIL

2 x: =root [T]

3 while x <> NULL

4 do y: =x

5 if val [z] < val [x]

6 then x: =left [x]

7 else x: =right [x]

8 p [z]: =NULL

9 if y = NULL

10 then root [T]: =z

11 else if val [z] <val [y]

12 then left [y]: =z

13 else right [y]: =z

Процедура рухається вниз по дереву, при цьому зберігаючи в y вказівник на батька вершини x. Порівнюючи значення в x та z, процедура вирішує, в яке з піддерев рухатись далі. Процес завершується тоді, коли x=NULL. Саме сюди й слід помістити вершину z (рядки 8-13). Ця операція також потребує часу для виконання, пропорційного O (h). **Видалення елементу** Параметром для процедури є вказівник на вершину, що видаляється. Тут можливі три варіанти дій: якщо у вершини немає дітей, достатньо помістити NULL у відповідне поле його батька (замість вказівника на вершину, що видаляється) якщо у вершини є одна дитина, можна з'єднати батька цієї вершини безпосередньо з її дитиною якщо вершина має двох дітей, слід спочатку знайти слідуючий (в сенсі порядку) елемент y, в якого немає лівої дитини, а потім скопіювати значення та додаткові дані з вершини y на місце вершини, що видаляється, а саму вершину y видалити.

DELETE (T, z)

1 if left [z] = NULL or right [z] =NULL

2 then y: =z

3 else y: =SUCCESSOR (z)

4 if left [y] <> NULL

5 then x: =left [y]

6 else x: = right [y]

7 if x <> NULL

8 then p [x]: =p [y]

9 if p [y]: =NULL

10 then root [T]: =x

11 else if y=left [p [y]]

12 then left [p [y]]: =x

13 else right [p [y]]: =x

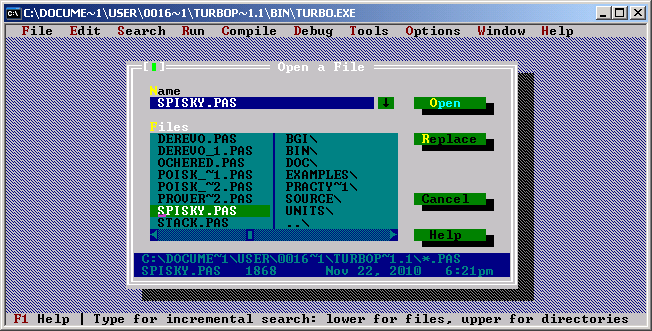
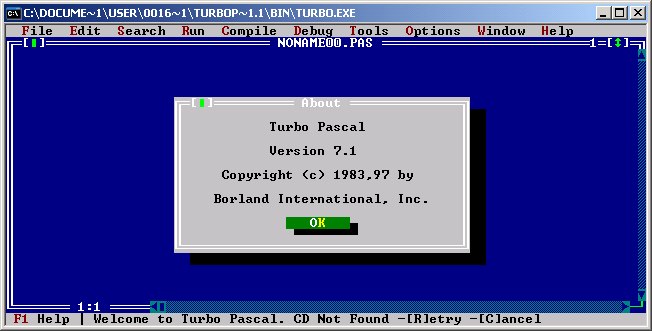
14 if y <> z

15 then val [z]: =val [y]

16 // копіювання додаткових даних з y

17 return y

# Відкриття програми



# 2. Списки

Внести до її тексту зміни та виконати її, створивши список, розмір якого визначено відповідно до варіанту № 15.

**Список (list) -** набір елементів, розташованих у певному порядку. Таким набором бути може ряд знаків у слові, слів у пропозицій у книзі. Цей термін може також ставитися до набору елементів на диску. Використання при обробці інформації списків як типи даних привело до появи в мовах програмування засобів обробки списків. Список черговості (pushup list) - список, у якому останній вступник елемент додається до нижньої частини списку. Список з використанням покажчиків (linked list) - список, у якому кожний елемент містить покажчик на наступний елемент списку. Лінійний список (linear list) - це безліч, що складається з вузлів, структурні властивості якого по суті обмежуються лише лінійним (одномірним) відносним положенням вузлів, тобто тими умовами, що якщо, те є першим вузлом; якщо, те k-му вузлу передує й за ним треба; є останнім вузлом. Операції, які ми можемо право виконувати над лінійними списками, є наступними:

1. Одержати доступ до k-го вузла списку, щоб проаналізувати й/або змінити вміст його полів.

2. Включити новий вузол безпосередньо перед k-им вузлом.

3. Виключити k-й вузол.

4. Об'єднати два (або більше) лінійних списки в один список.

5. Розбити лінійний список на два (або більше) списки.

6. Зробити копію лінійного списку.

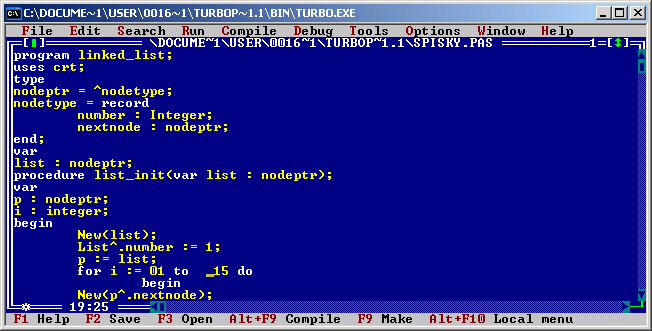
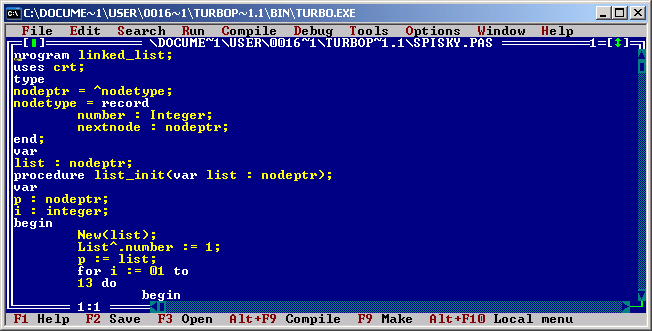
7. Визначити кількість вузлів у списку.

8. Виконати сортування вузлів списку в зростаючому порядку по деяких інформаційних полях у вузлах.

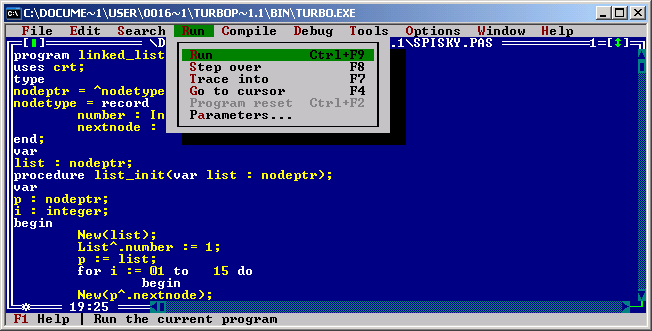
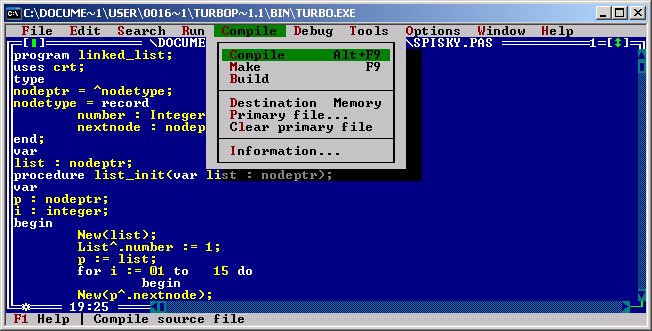
9. Знайти в списку вузол із заданим значенням у деякім полі.

бінарне дерево структура масив

Спеціальні випадки k=1 і k=n в операціях (1), (2) і (3) особливо виділяються, оскільки в лінійному списку простіше одержати доступ до першого й останнього елементів, ніж до довільного елемента.



# Проводимо компілюцію



# 3. Вводимо дані стеку

Виконати її, ввівши дані до стеку, глибина якого визначена відповідно до варіанту №9.

Стек (stack) - лінійний список, у якому всі видалення й доповнення (і звичайно всякий доступ) робляться з одного кінця списку. Стек - частина пам'яті ОЗУ комп'ютера, що призначається для тимчасового зберігання байтів, використовуваних мікропроцесором; при цьому використається порядок запам'ятовування байтів “останнім увійшов - першим вийшов”, оскільки таке добавлення й видалення організовувати простіше всього, то операції здійснюються дуже швидко. Дії зі стеком виконуються за допомогою регістра покажчика стека. Будь-яке ушкодження цієї частини пам'яті приводить до фатального збою. Стек у вигляді списку (pushdown list) - стек, організований таким чином, що останній елемент, що вводить в область пам'яті, розміщується на вершині списку. З стека ми завжди виключаємо "молодший" елемент із наявних у списку, тобто той, котрий був включений пізніше інших. Для черги справедливо в точності протилежне правило: видаляється завжди "старший" елемент; вузли залишають список у тім порядку, у якому вони в нього ввійшли. Стеки дуже часто зустрічаються в практиці. Простим прикладом може послуговувати ситуація, коли ми переглядаємо безліч даних і утворюємо список особливих станів або об'єктів, які повинні оброблятися пізніше; коли первісна безліч оброблена, ми повертаємося до даного списку й виконуємо наступну обробку, видаляючи елементи зі списку, поки список не стане порожнім. Для цієї мети придатні як стек, так і черга, але стек, як правило, зручніший. При вирішенні завдань наш мозок поводиться як "стек": одна проблема приводить до іншої, а та у свою чергу до наступної; ми накопичуємо в стеці ці завдання й підзавдання й забуваємо про них у міру того, як вони вирішуються. Аналогічно процес входів у підпрограми й виходів з них при виконанні машинної програми подібний до процесу функціонування стека. Стеки особливо корисні при обробці мов, що мають структуру вкладень. Взагалі, стеки найчастіше виникають у зв'язку з алгоритмами, що мають явно або неявно рекурсивний характер.

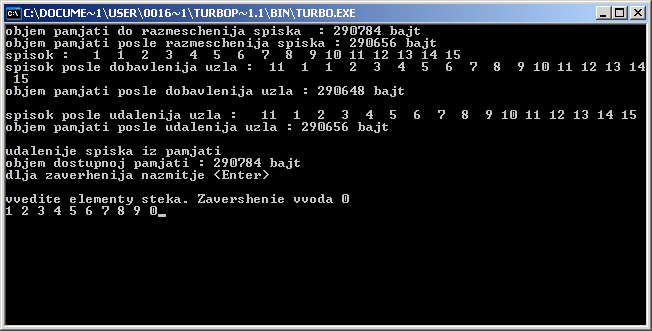
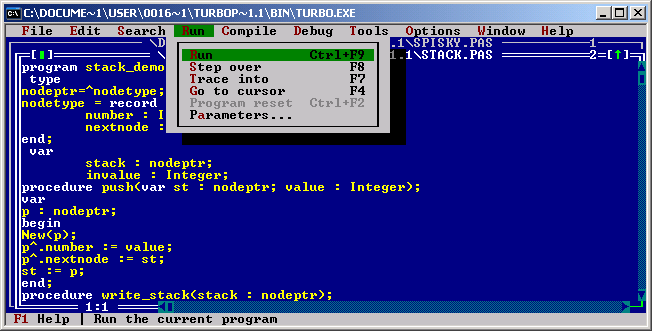
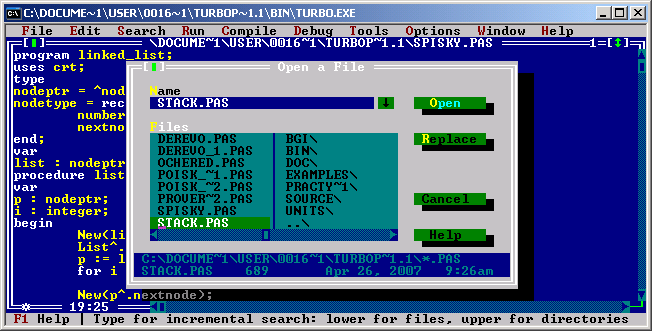
У стеці елемент додаються й віддаляються тільки з одного кінця. На малюнку це елемент N. Тобто якщо він додався, то видалятися може спочатку тільки він, а вже потім всі інші. Для стеку характерні такі властивості:

елементи додаються у вершину (голову) стеку;

елементи видаляються з вершини (голови) стеку;

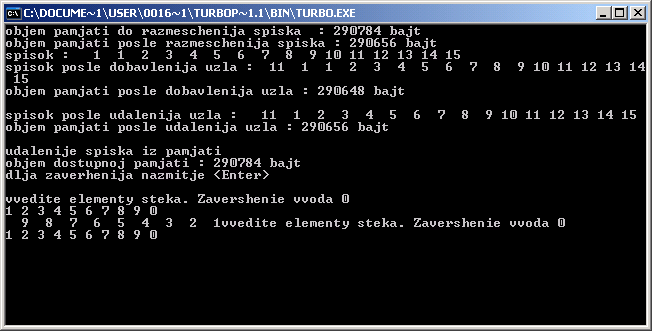
покажчик в останньому елементі стеку дорівнює NULL;

неможливо вилучити елементи стеку, не вилучивши всі елементи, що йдуть попереду. Стек можна уявити собі як коробці, у яку складають які-небудь предмети, щоб дістати самий нижній потрібно попередньо витягти інші. Стек можна вподібнити стопці тарілок, з якої можна взяти верхню й на яку можна покласти нову тарілку. [Інша назва стека в літературі - “магазин” - зрозуміло всякому, хто розбирав автомат Калашникова]. У програмуванні стеки мають широке застосування. Наприклад під час виклику підпрограми адрес повернення до неї зберігається у стеку. У випадку, коли відбувається цілий ряд послідовних викликів, адреси повернення розміщаються в стеці в порядку останнім прийшов - першим вийшов, так що після завершення виконання кожної функції відбувається перехід до функції, її що викликала. Стек підтримує як звичайні нерекурсивні виклики, так і рекурсивний виклик функцій. Стек використовується компілятором під час обчислення виразів, до нього записуються значення локальних змінних тощо.



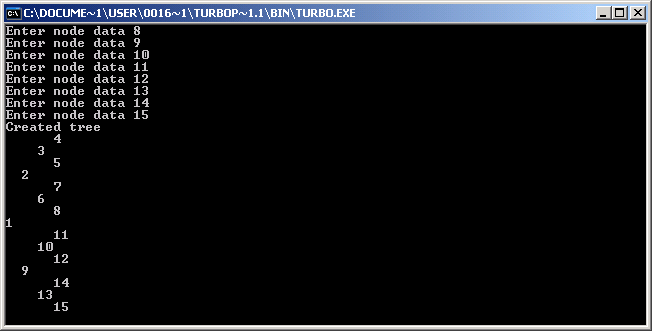
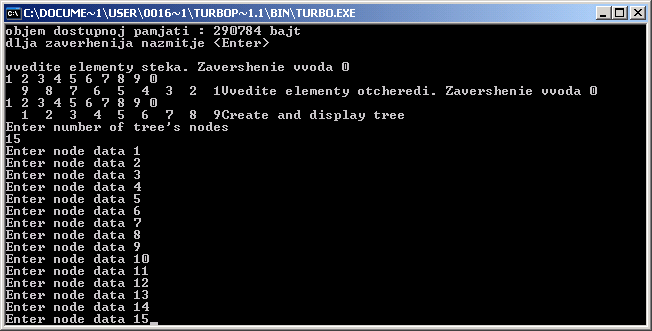
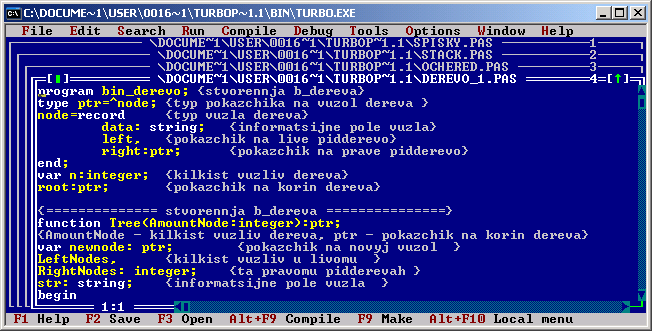
# 4. Вводимо дані черги

Виконати її, ввівши дані до черги, довжина якої визначена відповідно до варіанту №9. Черга (queue) - лінійний список, у якому всі видалення відбуваються на одному кінці списку, а всі включення (і звичайно всякий доступ) робляться на іншому його кінці. Черга - тип даних, при якому нові дані розташовуються слідом за існуючими в порядку надходження; першими дані, що надійшли, при цьому обробляються першими. У деяких розділах математики слово "чергу" використовують у більше широкому змісті, позначаючи будь-який сорт списку, у якому наявні видалення й додавання; зазначені вище спеціальні випадки називаються тоді "чергами з різними дисциплінами". Однак тут термін "черга" використовується у більш вузькому змісті, аналогічному впорядкованим чергам людей, що очікують обслуговування. Правило тут таке ж, як у живій черзі: першим прийшов - першим тебе і обслужений. Прийшов новий покупець, встав (добавився) у кінець черги, а який уже зробив покупки пішов (вийшов) з початку черги. Тобто першим прийшов, першим пішов. Інакше кажучи, у черги є голова (head) і хвіст (tail). Елемент, що додається в чергу, виявляється в її хвості. У черзі новий елемент додається тільки з одного кінця. Видалення елемента відбувається на іншому кінці. Черга, це по суті однонаправлений список, тільки додавання й видалення елементів відбувається на кінцях списку. Черга характеризується такими властивостями: · елементи додаються в кінець черги; · елементи зчитуються та видаляються з початку (вершини) черги; · покажчик в останньому елементі черги дорівнює NULL; · неможливо отримати елемент із середини черги, не вилучивши все елементи що ідуть попереду. Наведемо приклади застосування черг в обчислювальній техніці. У мережній операційній системі процесор сервера обслуговує в певний момент часу тільки одного користувача. Запити інших користувачів записуються до черги. Під час обслуговування користувачів кожен запит просувається до початку черги. Перший в черзі запит підлягає "першочерговому" обслуговуванню. У комп'ютерній мережі за чергою обслуговуються інформаційні пакети. Черги застосовуються також для буферизації потоків даних, що виводяться на друк, якщо в комп'ютерній мережі використовується один принтер. Крім цих структур існують і інші, наприклад деки, двонаправленні списки, кільцеві списки і т. і. Ь На малюнку нище графічно зображено дек (deck) (стек із двома кінцями) - лінійний список, у якому всі додавання й видалення (і звичайно всякий доступ) робляться на обох кінцях списку. Дек по суті двонаправлений список. У зв'язаному списку (linked list) елементи лінійно впорядковані, але порядок визначається не номерами, як у масиві, а покажчиками, що входять до складу елементів списку. Списки є зручним способом зберігання динамічних даних, що дозволяють реалізувати всі операції, (хоча й не завжди ефективно). Інакше кажучи, елемент двостороннє зв'язаного списку (doubly linked list) - це запис, що містить три поля: key (ключ) і два покажчики - next (наступний) і prev (від previous-попередній). Крім цього, елементи списку можуть містити додаткові дані. Якщо х - елемент списку, то next вказує на наступний елемент списку, а prev - на попередній. Якщо prev{х}=nil, то в елемента х немає попереднього: це голова (head) списку. Якщо next{х}= nil, то х - останній елемент списку або, як говорять, його хвіст (tail). Ці дані є неявно загальноприйнятими в програмуванні. Звичайно, динамічні структури даних не обмежуються наведеними вище. Існують і інші, зокрема графи, дерева що займають свою окрему нішу у програмуванні і почасти вирішення певних питань не можливе без їх застосування.

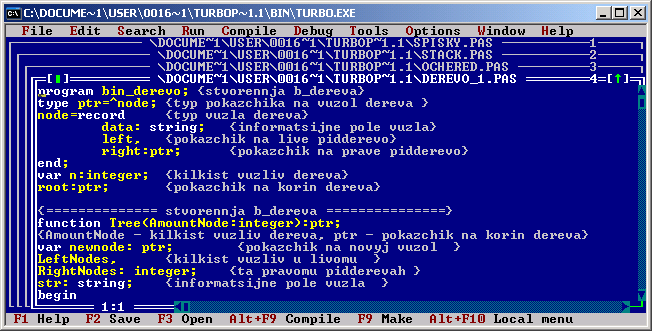


# Бінарне дерево

Виконати її, створивши бінарне дерево, кількість вузлів якого визначена відповідно до варіанту №15. Навести зображення дерева. Деревоподібна архітектура має найменший діаметр серед всіх існуючих, який дорівнює для бінарного дерева 2lg ( (P+1) /2) - відстань між двома листами, шлях між якими проходить через корінь. Для k-арних дерев діаметр зменшується зі збільшенням k. Недоліком деревоподібних мереж є те, що обмін даними між процесами відбувається за лінійний час, а процес-корінь є вузьким горлом при передачі інформації. Багатопроцесорний комп'ютер з паралельною обробкою інформації називається деревовидною машиною (tree machine) [33], якщо його процесори сполучені зв'язками так, що утворюється топологія повного бінарного дерева. Такий комп'ютер має 2d-1 процесорних елементів для деякого d, які розбиті на d рівнів, пронумерованих від 1 до d. Кожний процесор на рівні j, 1ЈjЈd, зв'язаний з єдиним процесором на рівні j-1 (батьком), та з двома процесорами на рівні j+1 (синами). Зв'язки між процесами розташовані таким чином, що безпосередньо обмінюватися інформацією можуть лише батько з сином. Єдиний процесор на першому рівні називається коренем в топології дерева, а процесори на рівні d - листами. Корінь не має батька, а листи не мають синів. На малюнку 15 зображена топологія деревовидної машини з d=4 рівнями, яка містить 24-1 = 15 процесорних елементів.



# Закінчили виконання практичної частини контрольної роботи



# Висновок

У виконаній роботі було розглянуто процеси пошуку інформацій та розроблено структури даних для ефективного зберігання та обробки інформації. Як приклад розглянуто бінарне дерево. Це динамічна структура даних, розмір якої обмежується тільки розміром віртуальної пам'яті комп'ютера. Бінарні дерева забезпечують пошук конкретного значення, максимуму, мінімуму, попереднього, наступного, операції вставки та видалення елемента. Розглянуті у роботі бінарні структури широко використовуються у житті, наприклад це різноманітні "ієрархічні структури", які нині широко використовуються в багатьох комп'ютерних завданнях. На даний час також розвивається граматичний аналіз, в основі якого і знаходяться принципи бінарних дерев. Граматичний аналіз на даний час широко використовується у сучасних пошукових алгоритмах.

# Список використаної літератури

1. Баррон Д. Рекурсивные методы в программировании, - М.: 1974

2. Дмитриева М.В., Кубенский А.А. Элементы современного программирования, - СПб.: 1991.

3. Барашенков В.В. Анализ и преобразование операторных схем алгоритмов: учебное пособие. - Л.: ЛЭТИ, 1979.

4. Кинг Д. Создание еффективного программного обеспечения. - М.: Мир, 1991.

5. Барашенков В.В. Интерпретация операторных схем алгоритмов. - Л.: ЛЭТИ, 1978.

6. Бентли Дж. Жемчужины творчества программистов. - М.: Мир, 1990.

7. Шауман А.М. Основы машинной арифметики. - 1979.

8. Ахо, Альфред В. и др. Построение и анализ вычислительных алгоритмов. - М.: Мир, 1979

9. Криницкий Н.А. Программирование и алгоритмические языки. - 1975.

10. Прикладные вопросы системного анализа. Межвузовский тематический сборник. Вып.2. Куйбышев, 1976.

11. Автоматическое построение ассоциативного списка со сжатием информации. - К., 1976.

12. Квиттнер П. Задачи, программы, вычисления, результаты. - М.: Мир, 1980.

13. Шауман А.М. Основы машинной арифметики. - 1979.

14. Малоземов В.Н. Певный А.Б. Рекуррентные вычисления. - Л., изд. ун-та, 1976.