Plag

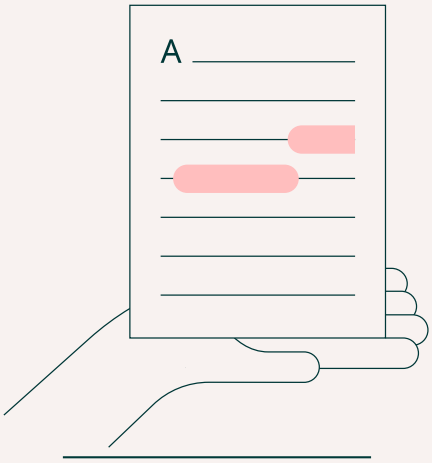
Звіт про оригінальність

● Оцінка схожості 29%

● Ризик плагіату НАЙВИЩИЙ

Olga Kagalo 2025-06-19 22:55

Посилання на звіт: 10mC4 / Посилання користувача: qEAc



Ось вона –
Ваша звіт про оригінальність!

Ми раді повідомити, що перевірка вашого документа завершена, і результати вже готові! Наші алгоритми старанно працювали, щоб знайти збіги в наших базах даних.

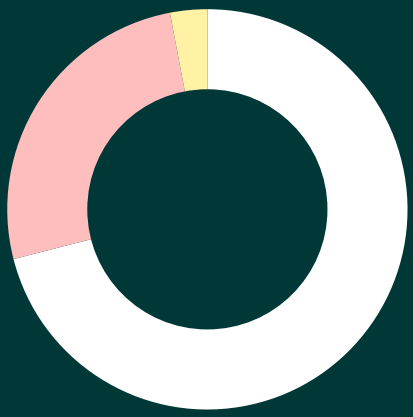
На наступних сторінках ви знайдете результати перевірки:

Бали
Збіги
Посилання

Ваш документ було перевірено за такими джерелами:

- ✓ База даних інтернет-джерел
- ✓ База даних наукових статей
- ○ Глибока перевірка (наш вдосконалений алгоритм)

Бали



Збіги тексту	26%
Перефразування	3%
Цитований текст	0%
Неправильне цитування	0%
Збігів не знайдено	71%

Ризик плагіату НАЙВИЩИЙ

Ризик плагіату вказує, як збіги тексту розподілені по документу. Вищий ризик виникає, коли збіги з'являються близько один до одного, наприклад, у тому самому абзаці або розділі.

Оцінка схожості 29%

Оцінка схожості показує, скільки слів або символів у вашому документі збігаються з текстами інших документів, включаючи перефразовані тексти або неправильні цитати.

Збіги

1 ВИКОРИСТАННЯ ДИНАМІЧНОЇ 12 ПАМ’ЯТІ ДЛЯ ЗБЕРІГАННЯ ДАНИХ

1.1 Огляд динамічних структур даних

Структура даних − це множина 4 елементів даних і множина зв’язків між ними. 4 Фізична структура даних відображає спосіб фізичного представлення даних в пам’яті 16 комп’ютера. Структури даних без врахування їх представлення 4 в пам’яті комп’ютера є 4 логічною структурою. Оскільки 4 між логічною і фізичною 4 структурами існує різниця, то 4 спеціальні процедури 4 здійснюють відображення логічної структури в фізичну і, навпаки.

4 Структури даних класифікуються 16 за різними ознаками. Структури даних є прості, складні і динамічні. Прості дані являються неподільними частинами. Складні 12 структури даних конструюються програмістом з використанням простих даних конкретної мови програмування.

В залежності від наявності або відсутності 12 зв’язків між елементами даних вони діляться на незв’язані структури (масиви, рядки) і зв’язані структури (списки, дерева). В залежності від способу виділення пам’яті структури діляться на статичні і динамічні. По характеру впорядкованості елементів структури їх 12 можна поділити на лінійні і нелінійні структури.

В залежності від характеру взаємного розміщення елементів в пам’яті лінійні структури є з послідовним розміщенням елементів в пам’яті (вектори, рядки, масиви, стеки, черги) і структури с довільним зв’язаним розміщенням елементів в пам’яті (однозв’язні, двозв’язні списки). Прикладом нелінійних структур є багатозв’язні списки, дерева, графи [2, 6, 8]. Схему класифікації структур даних наведено на рис. 1.1.

Рисунок 1.1 – 12 Класифікація структур даних

12 Структури 17 даних тісно пов'язані з типами даних. Будь-які дані, характеризуються типами. Інформація за кожним типом однозначно визначає:

структуру зберігання даних вказаного типу, тобто виділення пам'яті і представлення

даних в ній;

множину допустимих значень, які може мати той або інший об'єкт даного типу;

множину допустимих операцій, які можна виконувати над об’єктами даного типу.

Використання динамічної пам’яті є зручною формою збереження даних, оскільки кількість даних в багатьох практичних задачах наперед невідома. В багатьох задачах необхідно змінювати також склад і конфігурацію набору даних, тобто введення нових та видалення існуючих елементів у вже сформований набір. Поширеними формами динамічних інформаційних структур даних є:

однонапрямлені списки;

двохнапрямлені списки;

двійкові дерева (розгалужені списки);

черги;

стеки.

Динамічні інформаційні структури − це сукупність взаємозв'язаних елементів за певним форматом. Списки базуються на лінійній організації зв’язків між елементами. Кожний елемент лінійного списку зв’язаний через відповідні вказівники з одним або декількома сусідніми елементами. Дерева є розгалуженими ієрархічно організованими динамічними структурами.

Характерною особливістю динамічних структур даних є той факт, що їх елементи розміщені в несуміжних ділянках пам’яті комп’ютера. Цих елементів може бути довільна кількість і вони можуть мати змінний розмір. Окремі елементи динамічних структур розміщуються по адресах пам’яті, котрі наперед невідомі. Тому адреси поточних елементів таких структур не можуть бути обчислені за адресами попередніх або наступних елементів. Для встановлення зв'язків між елементами динамічних структур використовуються вказівники.

Програмно елементи динамічних структур реалізовуються через структурний тип, який складається щонайменше з двох полів: інформаційного, в якому зберігаються дані цього елементу, та адресного, через яке елемент пов'язується з іншим елементом. В основному всі елементи списку мають однаковий структурний тип. В загальному випадку інформаційне поле є інтегрованою структурою – вектором, масивом, записом і т.д.

При розв'язанні прикладних задач з використанням зв’язаних даних "видимим" є тільки вміст інформаційного поля, а поле зв'язку використовується програмістом тільки при написанні програми. Зв'язане представлення даних забезпечує мобільність динамічних структур. При зміні логічної послідовності елементів структури не потрібно переміщати всі дані в пам’яті, а тільки здійснити модифікацію вказівників.

Недоліком зв'язаного представлення даних полягає в складності роботи з вказівниками. Адже це вимагає високої кваліфікації програміста. Також використання полів на зв'язок елементів потрібна додаткова пам'ять. Доступ до елементів зв'язної структури є довшим по часу.

Якщо в суміжному представленні даних для обчислення адреси будь-якого елемента достатньо знати початкову адресу, то для зв'язаної структури адреса елементу не може бути обчислена виходячи з початкових даних.

1.2 Процес управління динамічною пам'яттю

Протягом роботи програми пам’ять, яка виділена для статичних даних, не змінюється. Це не є ефективно при опрацюванні великих масивів даних. Недоліком використання статичних даних є їх фіксований розмір. Якщо дані змінюють свій розмір в ході виконання програми, то для їх зберігання використовується динамічна пам’ять. Використання динамічної пам’яті дозволяє міняти розмір даних, контролювати її розподіл і робити його більш ефективним для розв’язуваної задачі [4, 5, 8].

Для роботи з динамічною пам’яттю використовуються адреси ділянок пам’яті – динамічні змінні. З їхньою допомогою здійснюється доступ до ділянок динамічної пам’яті. Виділені ділянки існують або до кінця роботи програми, або до тих пір, поки не будуть звільнені за допомогою спеціальних функцій.

При динамічному розподілі пам’яті можна запрошувати і міняти її розмір в процесі виконання програми. Наявність вільної пам’яті і її розподіл залежить від операційної системи комп’ютера. Операційна система або виділяє ділянку динамічної пам’яті, або видає повідомлення про її відсутність. Засоби роботи з динамічною пам’яттю продемонстровано на прикладі мови програмування С [2].

Динамічні змінні створюються за допомогою стандартних функцій, що містяться в бібліотечному файлі stdlib.h, котрий підключається за допомогою директиви #include. Функції виділення динамічної ділянки пам’яті повертають вказівник на початок виділеної ділянки.

Управлінням динамічною пам’яттю здійснюється операційною системою через простий

інтерфейс стандартних процедур або функцій. Питання виділення і звільнення динамічної пам’яті 9 повинні бути вирішені при проектуванні операційних систем і систем програмування. При розподілі динамічної пам’яті розв’язуються наступні задачі:

облік вільної пам’яті;

виділення ділянки динамічної пам'яті заданого розміру;

звільнення виділеної динамічної пам'яті.

Доступна для розподілу динамічна пам’ять являє собою неперервну ділянку з адресами від до ., де і − початковий та кінцеві номери байтів ділянки пам’яті, призначеної 9 для динамічного розподілу. При 9 кожному 9 запиті на виділення пам'яті видається адреса початку ділянки, де буде розміщуватися виділена пам’ять. Система сама визначає розміщення виділеного блоку динамічної пам’яті. При виділенні пам'яті повертатьсяє адреса цього блоку.

Пам’ять виділяється блоками, тобто неперервними ділянками суміжних байтів. Блоки можуть бути фіксованої або змінної довжини. Фіксований розмір блоку набагато зручніший для управління. В цьому випадку вся 9 доступна для розподілу пам’ять розбивається на ділянки, розмір кожної з яких дорівнює розміру блоку. Динамічні змінні і вказівники автоматично створюються при вході в той блок, в якому вони описуються. Вони існують на протязі роботи всього блоку і знищуються при виході з нього [9, 13].

Проблемою управління пам’яті є фрагментації або дроблення пам’яті. Вона полягає у виникненні ділянок пам’яті, які не можуть бути використані. Система управління пам’яттю повинна мати інформацію про те, які ділянки пам’яті вільні, а які – зайняті.

1.3 Стандартні функції для виділення та звільнення динамічної пам’яті

Створення і 15 використання динамічних структур даних вимагає динамічного розподілу пам’яті, що дає можливість одержувати в процесі роботи програми додаткову пам’ять для зберігання нових елементів структур, і звільняти використані блоки пам’яті [2].

Для динамічного розподілу пам’яті існує ряд стандартних функцій. Ці функції містяться в стандартних бібліотеках конкретної мови програмування. Робота цих функцій підтримується відповідною системою програмування. В дипломному проєкті розглянуто засоби роботи з динамічною пам’яттю на мові програмування С. Цей механізм дозволяє робити запит на виділення додаткових областей оперативної пам’яті в процесі виконання програми. Прототипи стандартних функцій містяться в бібліотечному файлі <stdlib.h> і наведені в табл. 1.1.

Таблиця 1.1 – Стандартні функції для розподілу динамічної пам’яті

Функція

Прототип функції

Призначення 5 функції

5 malloc()

5 void *malloc(size_t s);

Виділення динамічної пам’яті розміром s байт. Пам’ять не заповнюється нулями.

5 calloc()

5 void *calloc (size_t 5 n, size_t s);

5 Виділення 5 динамічної пам’яті для розміщення n елементів по s байт. Заповнення пам’яті нулями.

Функція malloc() виділяє задану кількість байтів неперервної ділянки оперативної пам’яті. Вона 15 повертає вказівник на початок 5 області виділеної ділянки динамічної пам’яті. Якщо потрібний об’єм пам’яті не можна виділити, то функція повертає адресну константу NULL. Виділення динамічної пам’яті за допомогою функції malloc() ілюструє наведений приклад:

u=(int*)malloc(sizeof(int));

Оскільки 18 функція повертає значення типу void*, то його необхідно перетворити до типу вказівника (int*).

Функція calloc () виділяє ділянку 5 динамічної пам’яті для розміщення n елементів по s байт, заповнює виділену пам’ять нулями і повертає вказівник на початок виділеної області. Якщо потрібний об’єм пам’яті функція не змогла виділити, то вона повертає адресну константу NULL.

Для розширення динамічної 5 пам’яті використовується функція realloc(). 5 Ця функція 5 змінює розмір блоку раніше виділеної динамічної пам’яті до заданої кількості байтів. байт. При використанні функції realloc() можливі такі випадки:

Якщо для розширення блоку за адресою block, є достатня кількість пам’яті, то функція виконається і поверне адресу block.

Якщо для розширення виділеного блоку по його поточній адресі block не досить пам’яті, то створюється новий блок потрібного розміру s і дані копіюються з старого блоку на початок нового. Старий блок звільняється і функція повертає вказівник на розширений блок пам’яті.

Якщо перший аргумент функції є NULL, то функція виділяє блок пам’яті розміром s байт і повертає вказівник на неї, тобто вона працює так, як і функція malloc().

Якщо аргумент s=0, то область пам’яті за адресою block звільняється і функція повертає адресну константу NULL.

Якщо для розширення виділеного блоку недосить пам’яті (не можна ні розширити старий блок, ні розмістити новий), то функція повертає NULL, а початковий блок виділеної пам’яті залишається незмінним.

При помилці функція повертає адресну константу NULL.

Оскільки розмір динамічної пам’яті є обмежений, то пам'ять після її використання потрібно звільняти. Коли програма закінчила роботу з блоком динамічно виділеної пам’яті, її треба звільнити. Звільнену пам’ять можна далі динамічно розподіляти і використовувати для інших даних. Для звільнення пам’яті, динамічно виділеної раніше, використовується функція free().

1.4 Створення вказівників на динамічні змінні

Для роботи з даними, розміщеними в динамічній пам’яті, створюються масив вказівників, кожний елемент якого зберігає адресу одного даного, записаного в пам’ять. Адреси елементів, розташованих в динамічній пам’яті, зберігаються в масиві вказівників ptr. Для звертання до елементів масиву ptr використовуються як індексні вирази, так і окремі вказівники ptr. Вказівник ptr вказує на елементи масиву вказівників, що є молодшими байтами даних.

Недоліком статичного масиву вказівників є його фіксований його розмір. Це не дає змоги опрацьовувати масиви даних будь-якого розміру. Встановлювати достатньо великим розмір масиву вказівників приведе до неефективного використання оперативної пам’яті.

Обмеження, пов’язані з фіксованою розмірністю масиву вказівників, можна усунути, якщо масиви вказівників формувати динамічно. Тоді його розмір можна встановлювати в процесі виконання програми, як цього вимагає конкретна реалізація алгоритму. При потребі динамічний масив вказівників можна розширити за допомогою функції realloc().

Статичний масив вказівників використовується для опрацювання символьних рядків у динамічній пам’яті. Оскільки довжини рядків тексту зазвичай є різними, а їх кількість може змінюватися залежно від їх реалізації, то доцільно зберігати введені рядки в динамічній пам’яті. Для кожного рядка виділяється динамічна пам’ять, що відповідає розміру рядка.

Адреси рядків, зберігаються в статичному масиві вказівників. Для звернення до елементів масиву використано як індексні вирази, так і окремий вказівник pptr. Цей вказівник повинен вказувати на елементи масиву вказівників, що вказують на перші символи рядків. Тому він описується наступним способом:

char ** pptr.

Значенням виразу *pptr буде адреса динамічної пам’яті, за якою записано відповідний символьний рядок. При просуванні по масиву вказівників перевіряється, чи не досягнуто елемент, значення якого рівне константі NULL.

1.5 Робота з динамічними блоками пам’яті

Для занесення в усі байти виділеного блоку пам’яті однакових символів використовується функція memset(). Прототип: функції memset() знаходиться у бібліотечному файлі <stdlib.h> і має вигляд:

void * memset (void * str, int n, unsigned m);

Аргумент str вказує на блок пам’яті, аргумент n задає значення, яке треба помістити в байти блоку, а аргумент m задає кількість байт для заповннення інформацією, починаючи з str. Змінна n хоча має тип int, вона використовується як символьний тип. Тому n приймає тільки значення з діапазону від 0 до 255. Функція memset() заповнює блок пам’яті побайтно символьними значеннями.

Для копіювання блоків пам’яті використовується функція memcpy(). Прототип: функції memcpy()і має вигляд:

void * memcpy (void * str1, void * str2, unsigned m);

Аргумент str1 вказує адресу динамічного блоку куди необхідно копіювати інформацію. Аргумент str2 вказує адресу динамічного блоку, з якого треба брати дані для копіювання. Аргумент m задає кількість байт, які треба копіювати.

Функція memcpy() повертає значення вказівника str1. Якщо обидва блоки співпадають, тобто str1= str2, то функція може працювати некоректно. При використанні функції memcpy() блоки пам’яті не повинні перекриватися.

Для переміщення блоків пам’яті використовується функція memmove(). Прототип функції memmove () має вигляд:

void * memmove (void * str1, void * str2, unsigned m);

Вказівники str1 і str2 вказують на початковий і кінцевий блоки пам’яті. Параметр m задає кількість байт, які необхідно копіювати. Функція повертає вказівник на початковий блок пам’яті str1.

Функція копіює один блок пам’яті в інший. Вона також працює з блоками пам’яті, які перекриваються або співпадають. Якщо блоки накладаються один на один, то функція копіює інформацію з перекриваючої області перше, ніж затре її новими значеннями [2].

Схему алгоритму заповнення і переміщення динамічних блоків даних в пам’яті зображено на рисунку 1.2.

Рисунок 1.2 –Схема алгоритму заповнення і переміщення динамічних

блоків даних в пам’яті

2 ДИНАМІЧНІ ІНФОРМАЦІЙНІ СТРУКТУРИ ДАНИХ

2.1 Основні поняття про лінійні списки

Лінійні списки широко застосовуються при опрацюванні даних. Лінійний список складається з набору елементів, розміщених в пам’яті комп’ютера в певному порядку Кожний елемент списку містить вказівник на наступний елемент. Прикладом лінійного списку можуть бути символи в слові, слова в реченні, речення в тексті. Елементи лінійних списків представляються за допомогою структур, які посилаються самі на себе. Визначення структурного типу для задання елементів лінійних списків:

```
struct spysok {
1 char name[10];
1 struct spysok *next;
1 }
```

Даний тип задає структуру struct spysok, яка складається з двох елементів:

1 масив name з десяти символів;

1 вказівник на структуру того 1 ж типу.

Кожний елемент списку являє собою складний структурний тип. Цей тип містить множину полів, які зберігають інформацію елементу списку і вказівник на наступний елемент. Вказівник визначає зв'язок між елементами в списку.

Структури, які посилаються самі на себе, зв'язуються разом і утворюють складні структури даних – списки. Тому кожна змінна структурного типу spysok може зберігати певну кількість даних і вказівник на іншу аналогічну структуру типу spysok.

Кожний елемент структури spysok вказує на наступний. Вказівнику останнього елемента списку присвоєно значення NULL, так як він не вказує на інший елемент. На рис. 2.1 наведені структури, які посилаються самі на себе, зв'язані між собою і утворюють список.

Рисунок 2.1 – Зв'язок структур, які посилаються на себе

Для розпізнавання першого елементу списку використовується спеціальний вказівник не структурного типу head – початковий вказівник. Він вказує на перший елемент в списку. Перший елемент містить вказівник на другий елемент, другий на третій і т. д. Це продовжується до тих пір, поки не зустрінеться елемент з вказівником NULL – останній елемент.

Так як елементи списку зв'язані між собою вказівниками, то при додаванні або викиданні елементів списку необхідно маніпулювати цими вказівниками. Для організації однозв'язного списку необхідно визначити структуру даних і описати початковий вказівник. Оскільки список створюється пустим, то його початковий вказівник рівний NULL. Вказівник на перший елемент масиву рівний head. За допомогою вказівника new додаються нові елементи в список:

```
struct spysok {

char name[10];

struct spysok *next; }

struct spysok * new;

struct spysok * head;

head = NULL;
```

Зв'язаний список є лінійним набором елементів (вузлів), які посилаються на себе. Доступ до елементів списку забезпечується вказівником на його перший елемент. Доступ до наступних вузлів проводиться через зв'язуючі вказівники, які зберігаються в

кожному елементі. Дані, які зберігаються в списках є динамічними, так як кожний вузол створюється по мірі необхідності. Елементи списків зазвичай не розміщуються в неперервній ділянці пам’яті.

2.2 Класифікація динамічних списків

Лінійні списки класифікують по напряму руху:

Лінійний список, кожний 2 елемент якого містить посилання на один сусідній елемент, 2 називається однозв'язним або однонапрямленим;

2 Лінійний 2 список, кожен елемент якого зберігає адреси двох сусідніх елементів, називається двозв'язним або двонапрямленим лінійним списком.

Оскільки елементи списку організовують 2 ланцюг даних, по якому можна рухатися в одному (однозв'язний список) чи двох 2 (двозв'язний список) напрямах, то 1 для звертання до елементів списку 2 достатньо зберігати адресу тільки одного початкового елемента, який називається 2 вершиною 2 списку. Кінцевий елемент списку, який не має наступників, називається хвостом списку. В його адресне поле заноситься адресна константа NULL.

На рис 2.2 наведено схематичне представлення однозв'язного і двозв’язного лінійного списків.

Рисунок 2.2 – Динамічні списки: а - однозв'язний список; б -двозв'язний список

Список може бути замкненим (кільцевим), 2 коли кінцевий елемент містить посилання на вершину списку - фактично такі списки не мають початку й кінця. Для зв'язку з 2 кільцевим 2 списком можна зберігати адресу довільного елемента цього списку.

2 Списки розрізняють також за 2 способом приєднання нових елементів до 2 них Найпоширенішими є 2 такі способи доповнення списку:

2 Приєднання елементів до вершини списку. 2 При цьому 2 кожен новий елемент приєднується перед початковим і стає в списку першим (вершиною списку). Такий список організації даних організований як стек.

2 Приєднання елементів до кінця списку. 2 При цьому 2 формується список, який називається 2 чергою, оскільки кожен новий 2 елемент стає останнім (кінцевим) елементом списку

2 вставлення елементів у визначені позиції списку. При проходженні по такому списку від вершини до 2 кінцевого елемента дані зчитуються 2 у порядку, зворотному до

порядку їх запису. Цей спосіб дає змогу формувати списки, впорядковані за певними критеріями.

Списки можуть впорядковуватися за значеннями ключів інформаційних полів (ключем називають певну ознаку, за якою здійснюється впорядковування і/або пошук елементів). Новий елемент може вводитись у довільну позицію списку, зокрема всередину між двома вже записаними елементами.

2.3 Однонапрямлені однозв'язні списки

Кожний елемент однонапрямленого списку являє собою структурний тип. Зиінні цього типу містить набір даних, які необхідні для збереження інформації про елементи списку. Крім того, в структурі є ще один додатковий елемент – вказівник. Цей вказівник визначає зв'язок між елементами в списку. Приклад структури:

```
struct spysok {
char name[10];
struct spysok *next;
}
```

Елементами структурного типу spysok є:

- масив name[10] з десяти символів;
- вказівник на структуру того ж типу.

Тому кожна структура типу spysok може зберігати певну кількість даних і вказівник на іншу аналогічну структуру spysok. На рис. 2.3 показано зв'язок елементів в однозв'язному списку.

Рисунок 2.3 – Зв'язок між елементами в однозв'язному списку

Кожна структура spysok вказує на наступну структуру spysok. Останній елемент ні на що не вказує. Тому вказівнику останнього елементу списку присвоєно значення NULL, для того, щоб його відрізнити від інших. Структури, які утворюють однозв'язний список, ще називають ланками або вузлами.

Для розпізнавання першого елементу списку використовується спеціальний вказівник не структурного типу – початковий вказівник. Він завжди вказує на перший елемент в списку. Перший елемент містить вказівник на другий елемент, другий на третій і т. д. до тих пір, поки не зустрінеться елемент з вказівником NULL.

1 Елементи до списку можна додавати, викидати або замінювати. Так як елементи списку зв’язані між собою вказівниками, то при додаванні або викиданні елементів списку необхідно маніпулювати цими вказівниками. Елементи можна додавати на початок списку, в середину та в кінець.

1 Для організації однозв’язного списку необхідно визначити структуру даних і описати початковий вказівник. Оскільки список створюється пустим, то його початковий вказівник рівний NULL.

1 Вказівник на перший елемент масиву рівний head. За допомогою вказівника new додаються 1 нові елементи в список:

```
1 struct spysok {
1 char name[10];
1 struct spysok *next;
1 }
1 struct spysok * 1 new;
1 struct spysok * head;
head = NULL;
```

2.4. Опис операції додавання елементу на початок лінійного 1 списку

1 Якщо початковий вказівник рівний NULL, то список порожній і його новий елемент стане його першим членом. Якщо початковий вказівник не рівний NULL, то список вже має один або декілька елементів. Тоді процедура додавання елементів на початок списку складається з наступних кроків:

1 Створення 1 екземпляру 1 структури з виділенням для нього пам’яті з допомогою функції malloc();

1 Встановлення 1 вказівника 1 наступного елементу в структурі нового елементу списку рівним текучому значенню початкового вказівника. Якщо список був порожнім, то вказівником доданого елементу в структурі 1 буде NULL, якщо список був не пустий, то вказівником доданого елементу буде 1 адреса текучого першого елемента;

1 Встановлення 1 початкового вказівника 1 рівним адресі нового елементу.

1 Описану операцію реалізує фрагмент програми на мові С:

new=(struct 1 spysok *)malloc(sizeof(struct spysok));

1 new- > 1 next= head;head-> 1 new;

1 Схему додавання нового елементу на початок пустого списку показано на рис. 2.4.

1 Рисунок 2.4 – 1 Додавання нового елемента в пустий список

1 Операцію додавання нового елементу на початок 1 непустого 1 списку показано на рис 2.5.

1 Рисунок 2.5 – 1 Додавання нового першого елементу в непустий однозв'язний список

2.5 Опис операції додавання елементу в кінець лінійного 1 списку

1 Для додавання елемента в кінець списку треба спочатку пройти від початкового вказівника до останнього елементу в списку. Після знаходження останнього елемента необхідно виконати наступні операції:

1 Створити екземпляр структури даних, розподіливши перед тим динамічно пам’ять за допомогою функції malloc();

1 Встановити вказівник в текучому останньому елементі на доданий елемент, адресу якого повернула функція malloc();

1 Встановити вказівник в доданому елементі NULL, щоб зробити його новим останнім елементом в списку.

1 Перераховані операції виконує фрагмент програми на мові С:

struct spysok {

1 char name[10];

1 struct spysok *next; };

1 struct spysok * 1 new;

1 struct spysok * head;

1 struct spysok *sp;

1 sp= head;

1 while (sp-> next!= NULL)

1 sp= sp-> next;

1 new=(1 struct 1 spysok *)malloc(sizeof(struct spysok));

1 sp-> next= new;

1 new- > next= = NULL;

Схематичне представлення операції 1 додавання нового елементу в кінець лінійного списку показано на рис. 2.6.

1 Рисунок 2.6 1 Додавання нового елемента в кінець однозв'язного 1 списку

1 2.6 1 Додавання елемента в середину однозв'язного списку

При додаванні елементу в середину однозв'язного списку спочатку визначається місце додавання елементу, а потім вже процес його. Для додавання елементу в середину списку 1 необхідно виконати наступні операції:

1 Знайти елемент в списку, який будемо називати маркером, після якого треба поставити новий елемент.

1 Створити екземпляр структури даних, виділивши 1 для нього динамічну 1 пам’ять за допомогою функції malloc();

1 Встановити вказівник в маркері на новий елемент, адрес якого повернула функція 1 malloc();

1 Встановити вказівник в новому елементі на цей елемент, на який перед тим вказував маркер.

Перераховані операції 1 реалізує наступний фрагмент програми:

1 struct 1 spysok {

1 char name[10];

1 struct spysok *next;

1 };

1 struct spysok * 1 new;

1 struct spysok * head;

struct spysok *sp;

struct spysok *marker;

new=(struct 1 spysok *)malloc(sizeof(struct spysok));

1 new- > next= marker- > next;

marker- > next= new;

Операція додавання елементу в середину лінійного списку зображено на рис. 2.7.

Рисунок 2.7 – 1 Додавання нового елемента в середину лінійного списку

2.7 Видалення елементу з лінійного списку

Процедура видалення елемента зі списку залежить від розміщення цього елемента в списку. При видаленні першого елементу початковий вказівник стає рівний адресі другого елементу списку. При видаленні останнього елементу вказівник в передостанньому елементі стає рівним NULL. Для видалення елементу зі середини списку вказівник елементу, який передує елементу, що видаляється, стає рівним адресі елементу, розміщеного безпосередньо за елементом, який видаляється.

Пам’ять, яку займав видалений елемент, звільняється 1 за допомогою функції звільнення 1 динамічно виділеної пам’яті free().

1 Фрагмент програми для видалення першого елементу з однозв’язаного списку має вигляд:

free(head);

head= head-> next;

/* Видалення останнього елементу зі списку*/

struct spysok {

1 char name[10];

1 struct spysok *next;

1 };

1 struct spysok *sp1, *sp2;

1 sp1= 1 head;

```
sp2= sp1-> next;
1 while (sp2-> next!= NULL)
1 {sp1= sp2;
sp2= sp-> next;
}
free(sp1-> next);
sp1-> next= NULL;
if ( head = = sp1)
head= NULL;
/* Видалення елементу зі середини списку */
struct spysok {
1 char name[10];
1 struct spysok *next;
1 };
1 struct spysok *sp1, *sp2;
1 /* 1 Фрагмент програми для установки вказівника sp1
на елемент, який стоїть перед видаленим */
sp2= sp1-> next;
free(sp1-> next);
sp1-> next= sp2-> next;
```

Після виконання будь-якого з цих операторів без використання функції free() викинутий елемент залишався б в пам’яті, хоч в списку його вже немає, так як ні один вказівник на нього не вказує. Тому пам’ять елементу, який викинуто зі списку, необхідно звільняти.

Лінійні списки – це ефективний спосіб організації динамічних даних. Оскільки є можливість додавати елементи в будь яке місце списку, то їх можна використати для

сортування даних. Алгоритми для сортування даних за допомогою однозв’язних списків набагато простіші, ніж алгоритми організації відсортованого масиву.

Гнучкість і потужність зв'язаного списку можна використати для пошуку за допомогою лінійного перегляду його елементів. Але число елементів повинно бути обмежене. Якщо список впорядкований, то пошук у впорядкованому списку можна закінчувати тоді, коли буде виявлений перший ключ.

Впорядкованість списку досягається включенням нового елементу у відповідне місце існуючого списку, а не 1 на початок списку. Тому впорядкованість забезпечена. Ні послідовності такої можливості не мають.

Пошук у впорядкованих списках можна використати тоді, коли елемент потрібно включати перед заданим. Новий елемент включається перед першим елементом з ключем, який більший ніж заданий.

Якщо слова зустрічаються в тексті з однаковою частотою, то їх впорядкованість є несуттєвою. Адже положення слова в списку не має суттєвого значення, оскільки всі слова зустрічаються з однаковою частотою.

Пошук у впорядкованих списках можна використати для пошуку слів в текстах. Але включенні будь-якого нового слова у впорядкований список варто використовувати лише для побудови словника з різною частотою повторення слів в словниках.

З ОПИС АЛГОРИТМУ І ПРОГРАМНИХ ДОДАТКІВ ДЛЯ СОРТУВАННЯ ДАНИХ З ВИКОРИСТАННЯМ ЛІНІЙНИХ СПИСКІВ

3.1 Алгоритм побудови однонапрямлених списків

При роботі з лінійними однонапрямленими списками використовуються вказівники. З їх допомогою здійснюється доступ до -го елементу списку. Динамічні вказівники створюються за допомогою спеціальних функцій та операцій. Вони існують або до кінця роботи програми, або до тих пір, поки не будуть звільнені за допомогою спеціальних функцій.

Форма звернення 1 до елементів структури за 1 допомогою вказівників 1 ефективна, коли для роботи з масивом структур використовують зовнішні вказівники, базовий тип яких збігається з типом структур – елементів масиву.

1 Лінійні зв'язні списки є простими динамічними структурами даних. Вони являють собою впорядкований масив, що складається із змінного числа елементів. Якщо обмеження на довжину списку, тобто на кількість елементів в ньому, не накладається, то список представляється в пам'яті у вигляді зв'язаної структури.

Якщо елемент списку не пов'язаний ні з яким іншим, то в полі вказівника цього елементу записується значення NULL, що не вказує ні на один елемент.

Переглядати елементи однонапрямленого списку можна тільки в одну сторону. При перегляді здійснюється послідовний доступ до елементів списку. Елементи переглядаються або до кінця списку або до знаходження потрібного елементу. Обробка однозв'язного списку не завжди ефективна, оскільки відсутня можливість перегляду елементів в протилежному напрямі.

Кожний елемент однонапрямленого списку являє собою структуру, що містить набір змінних, які необхідні для збереження інформації про елемент, і ще один обов’язковий елемент – вказівник. Цей вказівник визначає зв’язок між елементами в списку. Структурну схему алгоритму побудови лінійних списків для зберігання і опрацювання даних наведено на рис. 3.1.

Рисунок 3.1 – Структурна схема алгоритму для побудови лінійних однонапрямлених списків

Для організації однозв’язного списку необхідно визначити структуру даних і описати початковий вказівник. Оскільки список створюється пустим, то його початковий вказівник рівний NULL. Алгоритм побудови лінійного зв’язаного списку складається з таких елементів:

Визначення структури елементу списку;

Виділення пам’яті для розміщення першого елементу списку;

Додавання першого елементу в пустий список;

Виділення пам’яті для розміщення чергового елементу списку;

Додавання елементу до непустого списку;

Для розпізнавання першого елементу списку використовується спеціальний початковий вказівник. Він завжди вказує на перший елемент в списку. Перший елемент містить вказівник на другий елемент, другий на третій і т. д. до тих пір, поки не зустрінеться елемент з вказівником NULL.

3.2 Дослідження операції додавання елементу до лінійного списку

При виконанні операцій над елементами лінійного списку важливим фактором є послідовність модифікації вказівників окремих елементів списку, яка забезпечує коректну зміну елементів списку, не змінюючи інших елементів. При Неправильна порядок модифікації вказівників може привести до втрати частини списку.

Для включення нового елементу до лінійного списку необхідно одержати доступ до його -го елементу. Елементи можна безпосередньо включати 1 на початок списку, всередину списку після заданої позиції, або 3 в кінець списку. Складно ефективно реалізувати доступ до -го елементу списку при великій кількості його елементів.

Над списками також можна виконувати операцію конкатенації. При конкатенації лінійних списків останній елемент першого списку містить порожній вказівник, що є ознакою кінця списку. В останній елемент першого списку заноситься вказівник, який вказує 1 на перший елемент другого списку. Відповідно, другий список стає продовженням першого.

Бакалаврська робота присвячена вивченню методів роботи з даними, організованими у вигляді зв'язаного списку. В роботі розроблено алгоритм, що використовує зв'язаний лінійний список для зберігання текстової інформації у вигляді символьних рядків. Складність алгоритму полягає в сортування текстових даних по мірі їх додавання до лінійного списку. Дані в списку завжди знаходяться в відсортованому порядку.

Для реалізації операцій над динамічними даними, організованими за допомогою лінійних списків, над елементами лінійних списків виконуються операції додавання нових елементів. Елементи додаються на початок, середину або 3 кінець списку в залежності від їх значення. Розрізняють такі типи операцій:

Додавання елементу 1 на початок списку;

1 Додавання елементу в середину списку.

Додавання елементу 3 в кінець списку.

3 При додаванні елементу в середину списку необхідно організувати пошук позиції для додавання елементу.

Структурну схему алгоритму, що демонструє процес додавання нових елементів до лінійного 1 списку, показано на рис. 3.2.

1 Рисунок 3.2 □ Структурна схема алгоритму для включення нового елементу

у відсортований лінійний список

3.3 Опис алгоритму для сортування даних з використанням лінійних списків

7 У практиці програмування часто виникає необхідність у впорядкуванні елементів використовуваних структур даних за якою-небудь ознакою, або за 7 яким-небудь критерієм. 7 В цьому випадку необхідно відсортувати множину елементів,

використовуючи задане відношення порядку.

7 Інший клас важливих та часто використовуваних алгоритмів складають алгоритми, що реалізують різні методи пошуку заданого елемента серед множини певної структури даних.

Серед структур даних необхідність у сортуванні виникає для масивів та файлів. Сортуванням це впорядкування 8 елементів деякої структури даних, на якій визначено відношення порядку, тобто за якою-небудь ознакою. В 8 залежності розміщення елементів 8 структур даних у оперативній пам’яті або на зовнішніх пристроях, розрізняють внутрішнє та зовнішнє сортування.

8 Якщо елементи деякої структури даних позначити через , через , – ключ елементу , , то сортуванням елементів є їх перестановка у таку послідовність, для якої їх ключі задовольняють відношенням:

.

Методи сортування масивів класифікують за часом їх роботи, який залежить від числа необхідних порівнянь ключів та числа перестановок елементів.

Алгоритм для сортування текстових даних, розміщених в динамічних структурах лінійних зв’язаних списках, складається з таких елементів:

завантаження початкового відсортованого списку в пам’ять. Дані в списку завжди знаходяться у відсортованому порядку;

пошук режиму включення нового елементу у відсортований список. Режим включення нового елементу визначається з умови одержання відсортованого списку. Список перебирається елемент за елементом. Ознакою кінця списку є рівність константі NULL вказівника 1 останнього елементу списку. Результатом перевірки є один з трьох можливих випадків при додаванні запису .;

виділення пам’яті для включення нового елементу у відсортований список. Вказівник доданого елементу встановлюється рівним значенню, яке повертає функція malloc ( ) при виділення динамічної пам’яті. Якщо пам'ять виділена помилково, то на екран виводиться повідомлення про помилку. В цьому випадку програма завершує роботу. При успішному виділенні пам'яті, програма продовжує роботу;

задання режиму включення нового елементу у відсортований список. Може бути три режими включення нового елементу: 1 на початок списку, в середину списку та в кінець списку;

3 режим включення 1 нового елементу на початок списку. Якщо новий запис

повинен стати першим, то вказівник в новому елементі встановлюється рівним адресі першого запису попереднього списку. Початковому вказівнику присвоюється адреса доданого елементу. В результаті новий запис додається 1 на початок списку;

1 режим включення нового елементу в середину списку. Якщо новий запис необхідно додати в середину списку, то значенню вказівника елементу, що передує доданому, присвоюється адреса нового елементу, яку повертає функція malloc() при виділення динамічної пам'яті. Вказівнику елементу, що додається, присвоюється адреса елементу, що слідує за доданим в список;

визначення номера для включення нового елементу 1 в список. Для визначення цього номера перебираються всі інші елементи в пошуках місця для нового елементу. Виконується перевірка, чи не є новий символьний рядок менший або рівний, ніж поточний. Якщо ця умова вірна, то саме сюди і слід помістити новий запис, користуючись розглянутим прийомом 1 додавання елемента в середину зв'язаного списку. Якщо ж новий символьний рядок більший, ніж поточний, то перегляд списку продовжується. Вказівник попереднього елементу рівний адресі доданого. Вказівник доданого елементу встановлено на адресу наступного;

режим включення 1 нового елементу в кінець списку. 3 Якщо при перегляді елементів списку, досягнуть його кінець, то виконується 1 додавання елемента в кінець списку. Якщо новий запис слід додати після останнього, то вказівнику останнього запису присвоюється адреса нового елементу, яку повертає функція malloc( ) при виділення динамічної пам'яті. Вказівник доданого нового елементу стає рівним NULL, що є ознакою кінця списку;

новий елементу включається у лінійний список. Формується відсортований список, який записується на зовнішні носії.

3.4 Опис програмного забезпечення для додавання нових елементів в лінійний список і його сортування

Розроблено програмне забезпечення на мові програмування С для включення елементів до лінійного списку і їх сортування по мірі додавання. Програма демонструє основні операції включення елементів до лінійного списку і їх опрацювання. Програма визначає режим 1 додавання елементів в список за критерієм сортування по зростанню. Вона додає елементи в таких режимах: додавання елементу 1 на початок списку; додавання елементу в середину списку; додавання елементу 3 в кінець списку.

3 При додаванні елементу всередину списку здійснюється пошук позиції за критерієм сортування по зростанню, після якої необхідно включити елемент. Розроблена програма вставляє слова в різні позиції списку, 1 елементами якого є символьні рядки

(слова). Програма виконує наступні функції:

1 1 Підключення бібліотечних файлів для використання стандартних функцій файлового вводу-виводу, функцій обробки символьної інформації та функцій роботи операційної системи:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

2 Опис імені файлу і режимів його відкриття:

FILE *fp;

char mode_w[4]="w";

char mode_r[4]="r";

char file_name[20]; /* Ім’я файлу, в якому є список */

3 Визначення структури даних для зв'язаного 1 списку:

1 struct data {

1 char name[12];

1 struct data *next;

1 }*marker;

1 4 1 Визначення функції f() для додавання 1 пропусків при виводі:

1 void f(int k)

1 {char d=' '; int i;

1 for (i=1; i<=k; i++)

1 printf("%c", d); return;}

1 5 Визначення функції free_memory() для звільнення пам’яті, зайнятої елементами списку:

1 void free_memory(LINK first)

1 {LINK cur_ptr;

1 LINK nex_rec;

1 cur_ptr = first; /* Початок очистки пам'яті */

while (cur_ptr != NULL) /* Перевірка на кінець списку */

{nex_rec = cur_ptr->next; /* 1 Одержання адреси наступного запису */

free(cur_ptr); /* Звільнення пам'яті поточного запису */

cur_ptr = nex_rec; /* зміщення на один запис */}}

6 Опис вказівників на перший, новий і текучий елементи зв’язаного списку:

LINK head = NULL;/*Вказівник 1 на перший елемент пустого списку */

LINK new = NULL; /* Вказівник на новий елемент */

LINK current = NULL; /* Вказівник на текучий елемент */

7 Введення імені файлу і його відкриття для читання лінійного списку:

printf("Vvedit file_name \n");

gets(file_name);

fp=fopen(file_name,mode_r);

if (fp!=NULL) {printf("file %s open mode %s\n",file_name, mode_r); }

else {printf("file %s not open mode %s\n", file_name, mode_r); exit(1);}

8 Виділення пам’яті для нового елементу, що додається. Вказівник new встановлюється рівним значенню, яке повертає функція malloc ( ). Якщо динамічна пам’ять не виділена, то на екран виводиться повідомлення про помилку, і виконання програми завершується. Якщо ж пам'ять була видвленв, то програма продовжує роботу.

new = (LINK)malloc(sizeof(PERSON));

9 Читання даних з файлу:

fscanf(fp, "%11s", c);

strcpy(new->name, c);

/* Занесення адреси передостаннього елементу списку */

current = head;

/* Заповнення списку елементами */

for (i=1; i<kil_word; i++)

{ while (current->next != NULL)

{current=current->next; }

/* Виділення пам’яті для наступного елементу */

new = (LINK)malloc(sizeof(PERSON));

current->next = new;

new->next = NULL;

/* Читання даних з файлу */ fscanf(fp, "%11s", c);

strcpy(new->name, c);}

10 Задання слова для включення в список:

puts("ENTER string char[12] for insert\n");

scanf("%s", input);

11 Визначення режиму додавання елементу. При rez=1 елемент додається 1 на початок списку, при rez=2 – всередину списку, при rez=3 3 в кінець списку. При rez=2 визначається значення параметра p, який задає номер позиції для додавання елементу, виходячи з критерію сортування списку по зростанню.

current = head;

strcpy(a,current->name);

k= strcmp(input,a);

if (k<=0) rez=1;

else while (current != NULL)

{ n=n+1; /* Обчислення номера позиції для вставки слова */

current = current->next;

if (current != NULL)

{ strcpy(a,current->name);

k= strcmp(input,a);

if (k<=0) { rez=2; break; }} else rez=3;}

12 Для додавання елементу в список у різних режимах використовується оператор вибору варіанту switch (). В залежності від значення параметра rez виконуються різні варіанти оператора.

case 1: /* Додавання 1-го елементу в список, список порожній */

{ new = (LINK)malloc(sizeof(PERSON));

new->next = head;/* Занулення вказівника поки що остан. ел. Списку */

head = new;

strcpy(new->name, input); /*Вказівник 1 на початок списку*/ break; }

case 2: /* Додавання елементу в середину списку */

p=n; /* Номер позиції для вставки слова */

if (p==1){

new = (LINK)malloc(sizeof(PERSON));

new->next = head->next ; head->next = new; }

/*Додавання 3 елементу в задану позицію списку */

/* Виділення пам'яті для наступного елементу */

else /*Пошук заданої позиції в списку */

{ marker=head;

for (j=1; j<p; j++) marker= marker->next;

new = (LINK)malloc(sizeof(PERSON));

new->next = marker->next;

marker->next = new; }

strcpy(new->name, input); break ;

case 3: /* додавання елементу 3 в кінець списку */

/* Припускається, шо список має хоч би один елемент*/

current=head;/*Занесення адреси передостаннього елементу списку*/

while (current->next != NULL)

{current = current->next;}

new = (LINK)malloc(sizeof(PERSON));

current->next = new;

new->next = NULL;

/* Вказівник 1 на початок списку */

strcpy(new->name, input); break;

13 Відкриття файлу для запису відсортованого списку:

fp=fopen(file_name,mode_w);

if (fp!=NULL) {printf("file %s open mode %s\n",file_name, mode_w); }

else {printf("file %s not open mode %s\n", file_name, mode_w); exit(1);}

14 Вивід на екран і запис у файл 1 значень елементів списку після 1 сортування:

1 current = 1 head;

1 printf("%lld", current->name); i=8;

1 while (current != NULL)

{ printf("\n"); f(i) ;

printf("%s",current->name);

fprintf(fp, "%11s\t",current->name);

printf("\n"); f(i) ;

1 printf("%lld", (current->next)); i=i+8;

1 current = 1 current->next; }

1 15 1 Звільнення виділеної пам’яті для елементів однозв’язного списку:

1 current = head;

free_memory(current);

Звільнену пам’ять можна використати для збереження інших змінних. Якщо динамічну пам’ять не звільняти, то, із за недостачі вільної пам’яті в деякий момент часу програма не зможе створити чергову динамічну змінну.

Результатом роботи програми є відсортований список слів і їх адреси. 1 Повний текст програми наведено 19 в 19 додатку А.

1 3.5 1 Результати роботи програми сортування даних

У файлі задана кількість слів. На рис. 3.3 показано вмістиме початкового файлу.

Рисунок 3.3 – Вмістиме початкового файлу

Для включення нових елементів в сформований список з клавіатури вводиться 1 додавання нового елементу в список. Номер позиції визначається з критерію сортування слів по зростанню. На екран виводиться відсортований список слів. На рис. 3.4 показано вигляд екрану зі заданим списком.

Рисунок 3.4 – Вигляд екрану зі заданим списком

Після запиту ввести нове слово, вводимо з клавіатури слово «end». Результат додавання слова «end» в алфавітному порядку до списку, показаному на рис. 3.3, наведено на рис. 3.5.

Рисунок 3.5 – Процес додавання слова «end» у відсортований список

Згідно алфавіту слово вводиться в кінець заданого списку.

Результат додавання слова «and» до списку наведено на рис. 3.6.

Рисунок 3.6 – Процес додавання слова «end» у відсортований список

Згідно алфавіту слово додається на початок списку. Результат додавання слова «break» до списку наведено на рис. 3.7.

Рисунок 3.7 – Процес додавання слова «break» у відсортований список

Згідно алфавіту слово додається в середину списку.

На рис. 3.8 показано вмістиме кінцевого сформованого файлу з доданими словами.

Рисунок 3.8 – Вмістиме кінцевого сформованого файлу

Програма формує динамічний список, впорядкований по алфавіту. В список введені дані додаються таким способом, щоб список завжди був впорядкований по алфавіту. При додаванні нового елементу створюється динамічна змінна-запис. Її полям присвоюються значення, які відповідають значенням полів вводу. Програмний блок сортування знаходить потрібну позицію для слова і додає це слово в список. При цьому коректуються значення вказівника вузла next, після якого повинне бути поміщене нове слово.

Посилання

Це джерела виділених збігів у вашому документі. Кожен збіг позначено темно-зеленим числом, яке відповідає вказаному тут джерелу. Джерела впорядковані за схожістю — чим вищий бал, тим сильніше збіг.

#	Джерело	%
1	antibotan.com	17.1%
2	pdf.lib.vntu.edu.ua	2.6%
3	bestprog.net	0.7%
4	nadoest.com	0.6%
5	ela.kpi.ua	0.6%
6	myshared.ru	0.6%
7	socrates.vsau.org	0.5%
8	studfile.net	0.5%
9	studopedia.org	0.5%
10	metod.suitt.edu.ua	0.5%
11	kgemt.org.ua	0.4%
12	infopedia.su	0.4%
13	sambir.wunu.edu.ua	0.4%
14	ir.nmu.org.ua	0.2%
15	irttri.com	0.2%
16	slideplayer.com	0.2%
17	studopedia.com.ua	0.1%
18	web.posibnyky.vntu.edu.ua	0.1%
19	diser.ntu.edu.ua	0.0%

Plag

Дякуємо, що перевірили
свій документ за допомогою
Plag!