ЛАБОРАТОРНА РОБОТА №10

Тема: Програмування обробки динамічно зв'язаних структур : стеків, черг, списків та дерев.

Мета роботи

• Отримати практичні навички у програмуванні роботи зі зв'язаними динамічними структурами.

Теоретичні відомості

Якщо елементи зв'язані між собою за допомогою покажчиків, то такий спосіб організації даних називається ***динамічними структурами даних,*** їх розмір динамічно змінюється під час виконання програми. З динамічних структур даних найчастіше використовуються лінійні списки, стеки, черги та бінарні дерева.

Лінійні списки

Лінійний список - це скінченна послідовність однотипних елементів (вузлів). Кількість елементів у цій послідовності називається довжиною списку. Наприклад, F=(1,2,3,4,5,6) - лінійний список, його довжина 6.

При роботі зі списками часто доводиться виконувати такі операції:

- додавання елемента в початок списку;
- вилучення елемента з початку списку;
- додавання елемента в будь-яке місце списку;
- вилучення елемента з будь-якого місця списку;
- перевірку, чи порожній список;
- очистку списку;
- друк списку.

Основні методи зберігання лінійних списків поділяються на методи послідовного та зв'язаного зберігання.

Послідовне зберігання списків. Метод послідовного зберігання списків ґрунтується на використанні масиву елементів деякого типу та змінної, в якій зберігається поточна кількість елементів списку. При послідовному зберіганні списків за допомогою масивів елементи списку зберігаються в масиві. Така організація дозволяє легко переглядати вміст списку та дода - вати нові елементи в його кінець. Але такі операції, як вставляння нового елементу в середину списку чи вилучення елементу з середини списку по- требують зсуву всіх наступних елементів. При збільшенні елементів масиву кількість операцій для впорядкування списку стрімко зростає

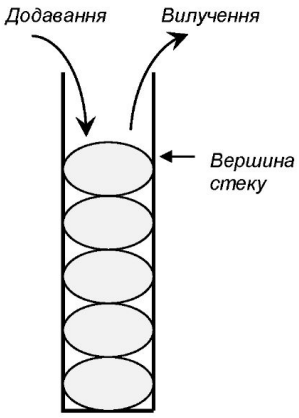
Зв'язане зберігання лінійних списків. Найпростіший спосіб зв'язати множину елементів - зробити так, щоб кожен елемент містив посилання на наступний. Це односпрямований (однозв'язаний) список. Якщо додати в такий список ще й посилання на попередній елемент, то отримаємо

двозв'язаний список. А список, перший та останній елементи якого зв'язані, називається ***кільцевим.***

Стеки

Стек - динамічна структура даних, яка являє собою впорядкований набір елементів, в якому додавання нових елементів і видалення існуючих проходить з одного кінця - *вершини стеку*.

Стек реалізує принцип *LIFO* (last in - first out, останнім прийшов - першим пішов). Найбільш наглядним прикладом організації стеку може бути дитяча пірамідка, де додавання і знімання кілець здійснюється як раз відповідно до цього принципу.



Основні операції над стеками:

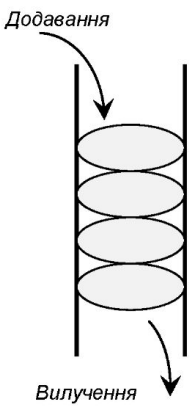
- додавання елемента в стек;
- вилучення елемента із стека
- перевірка, чи порожній стек;
- перегляд елемента у вершині стека без видалення;
- очистка стека.

Стек створюється так само, як і лінійний список, оскільки стек є частковим випадком односпрямованого списку.

Черги

Черга - це лінійний список, де елементи вилучаються з початку списку, а додаються в кінець (як звичайна черга в магазині).

Двостороння черга - це лінійний список, у якого операції додавання, вилучення і доступу до елементів можливі як спочатку так і в кінці списку. Таку чергу можна уявити як послідовність книг, що стоять на полиці так, що доступ до них можливий з обох кінців.



Черга є частковим випадком односпрямованого списку. Вона реалізує принцип *FIFO* (first in - first out, першим прийшов - першим пішов).

Черги створюються аналогічно до лінійних списків та стеків.

Розглянемо програмування роботи з однозв'язним списком на прикладі програми, що дозволяє користувачу ознайомитись з формуванням та обслуговуванням черги.

```
#include <stdio.h>              //підключення бібліотеки вводу/виводу

struct elem                     //оголошення структури для створення списку
{
        int value;              //оголошення поля для збереження значення
//оголошення поля для збереження адреси наступного елемента списку
        elem *pnext;
};

//оголошення глобальної змінної для адреси першого елемента списку
elem *pfirst;
void in_val(int);       //оголошення глобальної функції додавання значення
bool out_val(int&);     //оголошення глобальної функції вилучення значення
void show_q();          //оголошення глобальної функції виводу значень буферу

void main()             //оголошення та визначення головної функції
{
        int val;        //оголошення змінної для вводу значення
        char ch;    //оголошення змінної для вводу значення виходу/продовження
        printf("Data Queue usage \r\n");        //вивід константного рядка
//вивід константного рядка
        printf("0-Quit, 1-Add value, 2-Get value, 3-Show Queue \r\n");
        pfirst=NULL;    //визначення змінної-покажчика за умовчуванням
        for(;;)         //вічний цикл
        {
                printf("Make your choice: ");   //вивід запиту до користувача
                scanf("%i", &ch); //отримання значення від користувача
                if(ch==0)       //якщо користувач обрав вихід
                        break;  //вихід з оператора
                else if(ch==1)  //якщо користувач обрав додавання значення
                {
                        printf("Enter new value: ");    //вивід запиту до користувача
                        scanf("%i", &val);//отримання значення від користувача
                        in_val(val);    //виклик функції для запису значення
                }
                else if(ch==2)  //якщо користувач обрав вилучення значення
                {
                        if(out_val(val))    //якщо є значення, то вилучити перше
                                printf("Value is: %i \r\n", val); //вивід значення
                        else                //якщо значення відсутні
//вивід константного рядка
                                printf("No values in Queue \r\n");
                }
                else if(ch==3)  //якщо користувач обрав вивід значень
                        show_q();   //виклик функції виводу значень буфера
                else            //якщо користувач обрав інше значення
//вивід константного рядка
                        printf("Your choice is not valid \r\n");
        }
}
```

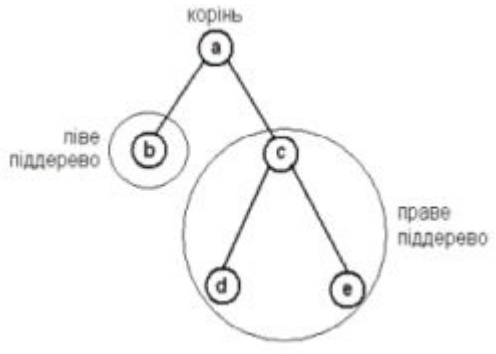
```
void in_val(int val)        //визначення глобальної функції додавання значення
{
        elem *pnew=new elem;  //динамічне створення екземпляра структури
        if(pfirst==NULL)          //якщо створений елемент є першим
                pfirst=pnew;      //запам'ятати його адресу
        else                      //якщо елемент не перший
        {
                elem *pcur; //оголошення змінної-покажчика поточного елемента

//визначення змінної-покажчика адресою першого елемента
                pcur=pfirst;
                while(pcur->pnext!=NULL)    //цикл перебору елементів списку
                        pcur=pcur->pnext;  //перевизначення поточного елемента
                pcur->pnext=pnew; //пов'язати новий елемент з останнім у списку
        }
        pnew->value=val;                //визначення значення елемента
        pnew->pnext=NULL;               //визначення змінної зв'язку
}

bool out_val(int& val)   //визначення глобальної функції вилучення значення
{
        if(pfirst!=NULL)                //якщо у списку є елементи
        {
                val=pfirst->value;    //отримати значення останнього елемента
//оголошення змінної-покажчика для зберігання адреси першого елемента
                elem *ptemp=pfirst;
                pfirst=pfirst->pnext;  //перевизначення адреси першого елемента
                delete ptemp;           //вивільнення пам'яті елемента
                return 1;               //повернути 1
        }
        else                            //якщо у списку елементи відсутні
                return 0;               //повернути 0
}

void show_q()           //визначення глобальної функції виводу значень буфера
{
        if(pfirst!=NULL)                //якщо у списку є елементи
        {
                elem *pcur; //оголошення змінної-покажчика поточного елемента
//визначення змінної-покажчика адресою першого а
                pcur=pfirst;
                do                      //цикл перебору елементів списку
                {
//вивід значення поточного елемента
                        printf("%i   ", pcur->value);
                        pcur=pcur->pnext;  //перевизначення поточного елемента
                }
                while(pcur!=NULL);      //доки є елементи
                printf("\r\n");         //перехід на новий рядок
        }
        else
                printf("No values in Queue \r\n"); //вивід константного рядка
}
```

Програма дозволяє користувачу обирати одну з чотирьох можливостей: завершення, додавання нового значення до черги, отримання значення черги та вивід на консоль всіх значень черги. Останні три дії реалізовано через виклик відповідних до підзавдань функцій. Програма виконується у вічному циклі доки користувачем не буде обрано варіант завершення. Черга базується на структурі ***elem,*** *що має два поля:* ***value*** *- для зберігання значення елементу, та* ***pnext*** *- адресу наступного за чергою елемента. Функція* ***in_val()*** *у якості параметру приймає значення, що необхідно додати у чергу. Перший її рядок реалізує динамічне створення нового екземпляра структури. Потім виконується перевірка значення глобальної змінної* ***pfirst,*** *що зберігає адресу першого елемента списку, і у залежності від нього створений елемент додається як перший чи останній. Поля елемента заповнюються наприкінці функції. Функція* ***out_val()*** *через параметр посилання повертає значення першого елемента списку. При відсутності елементів параметр, що повертається приймає значення* ***0,*** *в іншому випадку -* ***1*** *.* ***out_val()*** *записує значення першого елементу у посилання, зберігає у* ***pfirst*** *адресу зв'язку і вивільнює пам'ять, що задіяна під перший елемент, видаляє зі списку. Функція* ***show_q()*** *при наявності елементів у списку перебирає їх з першого до останнього, поступово виводячи на консоль значення, що в них збережені. Захист від похибки дозволяє запобігти вводу користувачем невідповідного значення вибору та отримання значення з порожнього списку.*

Треба зазначити, що у вищенаведеній програмі зручніше було б, окрім змінної ***pfirst,*** також використовувати змінну ***plast,*** щоб не виникало необхідності при додаванні значень у чергу кожного разу виконувати перебір всього списку у пошуках останнього елемента.

Двійкові дерева

Бінарне дерево - це динамічна структура даних, що складається з вузлів (елементів), кожен з яких містить, окрім даних, не більше двох посилань на інші бінарні дерева. На кожен вузол припадає рівно одне посилання. Початковий вузол називається *коренем дерева.*



Якщо дерево організоване таким чином, що для кожного вузла всі ключі його лівого піддерева менші за ключ цього вузла, а всі ключі його правого піддерева - більші, воно називається *деревом пошуку.* Однакові ключі в деревах пошуку не допускаються.

В дереві пошуку можна знайти елемент за ключем, рухаючись від кореня і переходячи на ліве або праве піддерево в залежності від значення ключа в кожному вузлі. Такий спосіб набагато ефективніший пошуку по списку, так як час виконання операції пошуку визначається висотою дерева.

Дерево є рекурсивною структурою даних, так як кожне піддерево є також деревом. Дії з такими структурами даних простіше всього описувати

за допомогою рекурсивних алгоритмів, хоча можна створити і його нерекурсивний еквівалент.

Для бінарних дерев визначені наступні операції:

- включення вузла у дерево;
- пошук по дереву;
- обхід дерева;
- видалення вузла.

Приклад програмного коду обробки бінарного дерева

```
# include <iostream.h>
# include <conio.h>

//Наша структура
struct node
{
    int info; //Інформаційне поле
    node *l, *r;//Ліва і Права частина дерева
};

node * tree=NULL; //Оголошуємо змінну, тип котрої структура Дерево

/*ФУНКЦІЯ ЗАПИСУ ЕЛЕМЕНТА В БІНАРНЕ ДЕРЕВО*/
void push(int a,node **t)
{
    if ((*t)==NULL) //Якщо дерева не існує
    {
        (*t)=new node; //Виділяємо пам'ять
        (*t)->info=a; //Кладемо у виділене місце аргумент a
        (*t)->l=(*t)->r=NULL; //Очищаємо пам'ять для наступного росту
        return; //Заложили насіннячко, виходимо
    }
       //Дерево є
        if (a>(*t)->info) push(a,&(*t)->r); //Якщо аргумент a
більше ніж поточний елемент, кладемо його вправо
        else push(a,&(*t)->l); //Інакше кладемо його уліво
}
/*ФУНКЦІЯ ВІДОБРАЖЕННЯ ДЕРЕВА НА ЕКРАНІ*/
void print (node *t,int u)
{
    if (t==NULL) return; //Якщо дерево порожнє, то відображати
нема чого, виходимо
    else //Інакше
    {
    print(t->l,++u);//За допомогою рекурсії відвідуємо ліве піддерево
    for (int i=0;i<u;++i) cout<<"|";
    cout<<t->info<<endl; //І показуємо елемент
    u--;
    }
    print(t->r,++u); //За допомогою рекурсії відвідуємо праве піддерево
}
```

```
void main ()
{
    int n; //Кількість елементів
    int s; //Число, передане в дерево
    cout<<"уведіть кількість елементів  ";
    cin>>n; //Уводимо кількість елементів

    for (int i=0;i<n;++i)
    {
    cout<<"уведіть число  ";
    cin>>s; //Зчитуємо елемент за елементом

    push(s,&tree); //И кожен кладемо в дерево
    }
    cout<<"ваше дерево\n";
    print(tree,0);
    getch();
}
```

Структура ***Node*** - представляє одну ланку нашого майбутнього дерева. У структурі оголошений інформаційний елемент і покажчики на саму праву і на саму ліву частину майбутнього дерева.

Оголошуємо покажчик на нашу структуру (на ланку дерева) і щоб уникнути непередбачених помилок відразу ініціалізуємо цей покажчик в нуль.

Після опису структури-ланки бінарного дерева і оголошення на неї покажчика, в головній функції пишемо стандартні рядки (скільки елементів ввести) і виконуємо цикл по введенню елементів вказане число разів. При цьому нам потрібно кожен новий елемент відразу записувати в дерево, з цією метою пишемо окрему функцію.

У наведеному коді функція приймає 2 параметра. Перший - записується число, другий покажчик на покажчик ланки дерева.

При занесення інформації, двійкове дерево або містить інформацію, або ще не заповнювалося, тому перший крок - перевірка на наявність даних у дереві за наступними ознаками.

- *Якщо дерево порожнє, то треба створити перший елемент. Цей елемент буде коренем дерева. Після створення першого елемента треба виділити пам'ять для можливих гілок.*
- *Якщо дерево щось містить, то перевіряється умова, згідно з якою ми розміщуємо в дереві елементи.*
- ***Бінарне дерево** - це впорядковане дерево, кожна вершина якого має не більше двох піддерев, причому для кожного вузла виконується правило: в лівому піддереві містяться тільки ключі, що мають значення, менші, ніж значення даного вузла, а в правому піддереві містяться тільки ключі, що мають значення , більші, ніж значення даного вузла.*

При цьому, якщо елемент, який ми хочемо помістити в дерево більше ніж кореневий, то за допомогою рекурсивного виклику функції відбувається послідовне переміщення елемента в праву частину. Наш елемент стрибає

через один, поки не знайде своє місце. А його місце це те місце, де праворуч елемент або більше ніж він сам або якщо стрибати більше немає через кого.

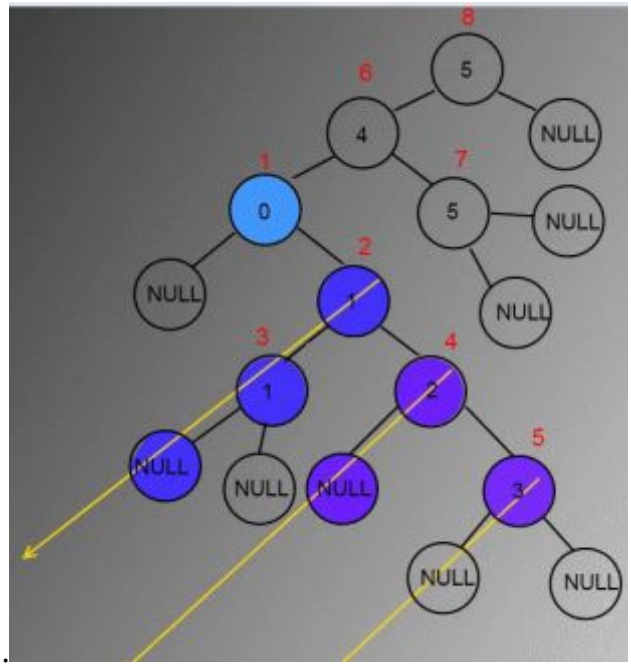
Якщо записується елемент менше ніж кореневий, то виконується така ж операція тільки в ліву сторону.

Для того щоб відобразити Бінарне дерево на екрані спочатку перевіряємо взагалі чи є воно. Якщо ми не заповнимо дерево і не закладемо насіннячко (не створимо корінь), то дерева в принципі існувати не може і тому відображати буде нічого, але якщо створили, то використовуючи рекурсивний виклик функції послідовно відображаємо всі елементи. Крім того, ця перша перевірка буде і сигналом для завершення рекурсивного виклику функції. Адже якщо зустрінеться Нульова ланка, то відбувається ***return,*** а ми після створення кожної ланки очищали пам'ять для наступного зростання

Пояснення навіщо використаний параметр ***u:*** при виведенні на екран ви будете бачити палички, які виводяться за допомогою циклу. Знайти в коді це місце неважко. Ці палички символічно позначають кількість вузлів

Слід нагадати, ***що вузол дерева, який не має нащадків називається лист***. У нашому випадку у дерева два листа 0 і 9.

Нехай користувач вводить числа і з них будується дерево. Те що вводить користувач з'являється по черзі і шукає своє місце. Коли число знаходить свою позицію, то бронює її. (Права частина заповнюється так само, як і ліва). Після заповнення дерева, пройтися по ньому не складе труднощів. Кожен крок проходу позначений в кінці червоною цифрою. Якщо зустрічаються рівні елементи, то вони створюють щось схоже на гілку і ця гілка з кожним елементом збільшується (в кінці позначено стрілками). При зчитуванні інформації з дерева спочатку все збирається з однієї гілки, потім з наступної і так поки «гілки» не закінчаться.



Примітка: Якщо користувач уведе число «4», то воно запишеться праворуч від числа «3», або зліва від «5». При додаванні чисел, вони будуть розташовуватись по жовтим стрілкам (зростаючі гілки).

Порядок виконання роботи

1. Засвоїти теоретичний матеріал. Використовуючи лекційний матеріал і інші літературні джерела, дослідити функції роботи з різними динамічними структурами.
2. Розробити програму згідно індивідуального завдання. Для кожного режиму роботи програми повинні бути розроблені окремі функції. Вхідні і вихідні дані повинні зберігатись у файлах.
3. Підготувати звіт:
   - варіант і текст завдання;
   - схема взаємодії функцій програми (з вказанням даних, що передаються і повертаються функціями);
   - схема даних програми;
   - лістинг програми і результати виконання;
   - висновки.

Варіанти індивідуальних завдань

Постановка задачі.

Створити динамічну структуру - список студентів, що містить не менше 10 студентів. Для кожного студента зазначити рік народження і оцінки по 4 екзаменам.

Реалізувати різні функції:

- додавання нових елементів у структуру;
- видалення елементу із заданим номером зі структури;
- виведення на екран інформації, яка зберігається у заданому елементі;
- виведення на екран усієї інформації зі структури.

Організувти:

- для списку та дерева – пошук та видалення елементів,
- для черги та стеку – відбір елементів у окремий масив,

що відносяться до студентів, у яких:

1. Середній бал менше середнього бала групи.
2. Середній бал менше 4.5.
3. Середній бал менше 4.
4. Середній бал менше 3.5.
5. Всі оцінки 5.
6. Одна 4, а інші 5.
7. Оцінка, отримана на першому іспиті, - 2.
8. Оцінка, отримана на другому іспиті, - 5.
9. Немає 3 і 2.
10. Більше однієї 2.
11. Одна 3, а інші 4 і 5.
12. Прізвище починається на букву Т
13. Прізвище починається на букву С

14.Рік народження нижче 1992.
15.Рік народження вище 1992

Результати представити у файлі Out_Sp.txt

- Дані про вилучених (відібраних) студентів;

- Дані про студентів, що лишились у списку (дереві), або дані про всіх, вилучених зі стеку або черги студентів).

№ варіанту	Вид динамічної структури
1, 5, 9, 13	Бінарне дерево
2, 6, 10, 14	Черга
3, 7, 11, 15	Стек
4, 8, 12, 16	Однозв'язаний список

Контрольні питання

1. Що таке динамічні структури? З чого вони складаються?
2. Наведіть різни приклади опису динамічних структур.
3. Які види динамічних структур існують?
4. Які види зберігання лінійних списків ви знаєте?
5. Які операції над лінійними списками можна виконувати?
6. Поясніть, що таке стеки.
7. Назвіть основні операції над стеками і поясніть їх виконання схематично.
8. Як можна отримати доступ до будь-якого елементу стека, окрім вершини?
9. Наведіть фрагмент коду для додавання елементу у стек.
10. Поясніть, що таке черги і правила їх організації.
11. Назвіть основні операції над чергами і поясніть їх виконання схематично.
12. Наведіть фрагмент коду для ініціалізації черги.
13. В чому сутність бінарного дерева?
14. Які операції над елементами бінарного дерева можна здійснювати?
15. Покажіть на прикладі створення бінарного дерева, елементи якого цілі числа або символи абетки.