Лабораторна робота №6

Технологія розробки алгоритмів розв'язання інженерних задач

Тема: Дерева алгоритми на деревах.

Мета: за варіантом визначити завдання; сформулювати спрощення задачі, описати принцип роботи алгоритму як послідовне спрощення задачі; блок-схему рекурентного алгоритму; довести гарантованість досягнення розв'язку.

Завдання

1. Визначте номер варіанту. З вказаної таблиці по першим літерам прізвища та ім'я визначте дві цифри. Обчисліть номер свого варіанту:

№(літера з прізвища)*6+№(літера з імені) = остання цифра є № вашого варіанту

А	Б	В	Г	Д	Е	Є	Ж	З	І
0	1	2	3	4	5	0	1	2	3
Ї	Й	К	Л	М	Н	О	П	Р	С
4	5	0	1	2	3	4	5	0	1
Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ю	Я
2	3	4	5	0	1	2	3	4	5

2. Запишіть структуру програми для забезпечення роботи з бінарним деревом, яке містить в якості елементів цілі числа.

3. Опишіть дії по пунктам, які потрібно виконати для додавання елементу до дерева так, щоб дерево залишалося впорядкованим.

4. В якому порядку потрібно додати до впорядкованого дерева числа 1,2,...,15 щоб воно було врівноваженим?

5. Вправте помилки в реалізації дерева з прикладу та заповніть його числами від 1 до 15 так, щоб дерево було врівноваженим. Напишіть та налаштуйте програму.

6. За визначеним варіантом з останньої цифри оберіть своє завдання:

№ вар.	Завдання згідно варіанту
0	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом A (див. приклад).
1	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом B (див. приклад).
2	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом C (див. приклад).
3	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом D (див. приклад).
4	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом E (див. приклад).
5	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15.

	Вихідні данні: Числа по порядку обходу методом F (див. приклад).
6	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом A (див. приклад).
7	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом B (див. приклад).
8	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом C (див. приклад).
9	**Вхідні данні:** Врівноважене впорядковане дерево з числами 1,2,...,15. **Вихідні данні:** Числа по порядку обходу методом D (див. приклад).

Запишіть результат роботи програми. Які методи виведуть номера за зростанням, а які за спаданням?

7. Словами описати процес додавання до дерева числа 17. Чи буде нове дерево врівноваженим?

8. Повторити досліди п.6 для дерева, яке заповнено числами в такому порядку, намалюйте отримане дерево:

N=(№ варіанту + 3); N,N-1,N-2,...,1,N+1,N+2,...,15

(Приклад. Номер варіанту 4. N=4+3=7. До дерева додаємо: 7,6,5,4,3,2,1,8,9,10,11,12,13,14,15)

9. Оцінити складність алгоритму.

10. Записати висновки про виконану роботу.

11. Відповісти на контрольні питання (в день виконання роботи усно, при перездачах чи доздачах та ін. — письмово).

12. Додаткове завдання. Доповнити клас дерева функцією малювання дерева на екрані монітора. Копію екрана додати до звіту. (+4 бали)

ДОПОМІЖНА ІНФОРМАЦІЯ. Реалізація бінарного дерева C++

```
class CElement
{
  public:
    int E; //Елемент, який зберігається в дереві
    CElement* Left; //Перехід вліво-вниз по дереву
    CElement* Right; //Перехід вправо-вниз по дереву
    CElement(int i); //Конструктор
    ~CElement(); //Деструктор
};

Celement::CElement(int i=0) //Конструктор
{
  E = i; //Значення елементу дерева
  Left = null; //Елемент новий, тому нижче пусто
  Right = NULL; //Елемент новий, тому нижче пусто
}

CElement::~CElement() //Деструктор
{
  if( Left!=NUll ) delete Left; //Якщо знизу щось є, то
  if( Right!=null) delete Right;//це теж треба знищити
}

//Клас керування деревом
class CTree
```

```
{
  private:
    CElement Root;
  public:
    CTree(); //Конструктор дерева
    ~CTree(); //Деструктор дерева
    addElement(int A); //Додавання елементу
                       //зі збереженням впорядкованості
    void clearTree(); //Вилучає всі елементи з дерева
    void printTreeA(CElement* Base=NULL); //Обхід дерева
    void printTreeB(CElement* Base=NULL); //різним порядком
    void printTreeC(CElement* Base=NULL);
    void printTreeD(CElement* Base=NULL);
    void printTreeE(CElement* Base=NULL);
    void printTreeF(CElement* Base=NULL);
};

CTree::CTree() //Конструктор дерева
{
  Root = NULL;
}
CTree::~CTree() //Деструктор дерева
{
  delete Root;
}
void CTree::addElement(int A) //Додавання елементу
{                          //зі збереженням впорядкованості
  if(Root == Null)
  { //Дерево пусте, додамо елемент як кореневий.
    Root = new CElement(A);
    return;
  }
  //Дерево вже є, потрібно правильно спуститися по ньому
  CElement* Now = null; //Теперішнє положення в дереві
  CElement* Next = Root; //Наступне положення в дереві
                         //почнемо з кореня
  do{
    Now = Next; //Переходимо на наступний елемент вниз
                //(на початку в корінь)
    if( Now->E<A )//Якщо наше число більше, то
    { //далі нам потрібно направо по дереву
      Next = Now->Right;
    }else
    { //інакше — спускаємося наліво, бо вставляємо менший елемент
      Next = Now->Left;
    }
  }while( Next != NULL ); //Повторюємо, доки наступний елемент
                          //не стане пустим — ми спустилися
  if( Now->E<A ) //Тепер останнє рішення про додавання елементу
  { //з правої сторони
    Now->Right = new CElement(A);
  }else
  { //або зліва
```

```
    Now->Left = new CElement(A);
  }
}
void CTree::clearTree() //Вилучає всі елементи з дерева
{
  delete Root; //Все інше вилучиться автоматично
               //деструктором елементу
}
void CTree::printTreeA(CElement* Base=NULL) //Обхід дерева
{
  CElement* Now = Base;
  if(Now == null) Now = Root;
  if(Now == null) return;
  cout << Now->E << endl;
  if( Now->Left != NULL )  printTreeA(Now->Left);
  if( Now->Right != NULL )  printTreeA(Now->Right);
}
void CTree::printTreeB(CElement* Base=NULL)
{
  CElement* Now = Base;
  if(Now == null) Now = Root;
  if(Now == null) return;
  if( Now->Left != NULL )  printTreeA(Now->Left);
  cout << Now->E << endl;
  if( Now->Right != NULL )  printTreeA(Now->Right);
}
void CTree::printTreeC(CElement* Base=NULL)
{
  CElement* Now = Base;
  if(Now == null) Now = Root;
  if(Now == null) return;
  if( Now->Left != NULL )  printTreeA(Now->Left);
  if( Now->Right != NULL )  printTreeA(Now->Right);
  cout << Now->E << endl;
}
void CTree::printTreeD(CElement* Base=NULL)
{
  CElement* Now = Base;
  if(Now == null) Now = Root;
  if(Now == null) return;
  cout << Now->E << endl;
  if( Now->Right != NULL )  printTreeA(Now->Right);
  if( Now->Left != NULL )  printTreeA(Now->Left);
}
void CTree::printTreeE(CElement* Base=NULL)
{
  CElement* Now = Base;
  if(Now == null) Now = Root;
  if(Now == null) return;
  if( Now->Right != NULL )  printTreeA(Now->Right);
  cout << Now->E << endl;
  if( Now->Left != NULL )  printTreeA(Now->Left);
}
```

```
void CTree::printTreeF(CElement* Base=NULL)
{
  CElement* Now = Base;
  if(Now == null) Now = Root;
  if(Now == null) return;
  if( Now->Right != NULL )  printTreeA(Now->Right);
  if( Now->Left != NULL )  printTreeA(Now->Left);
  cout << Now->E << endl;
}
```