ГУАП

КАФЕДРА № 34

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Старший преподаватель		К.А. Жиданов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

по курсу: ЯЗЫКИ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 3145		Я.А.Скрипников
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Вариант 3.

Бинарное дерево (добавление, поиск)

Цель работы:

Реализовать АТД (абстрактный тип данных) в виде пользовательского типа данных и набора функций, реализующих заданные операции. Помимо стандартных интерфейсов (чтение/добавление/поиск/удаление), требуется реализовать чтение/выгрузку данных из файла.

Ход работы:

- Бинарное дерево — это иерархическая структура данных, в которой каждый узел имеет значение (оно же является в данном случае и ключом) и ссылки на левого и правого потомка. Узел, находящийся на самом верхнем уровне (не являющийся чьим либо потомком) называется корнем. Узлы, не имеющие потомков (оба потомка которых равны NULL) называются листьями
  Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде.

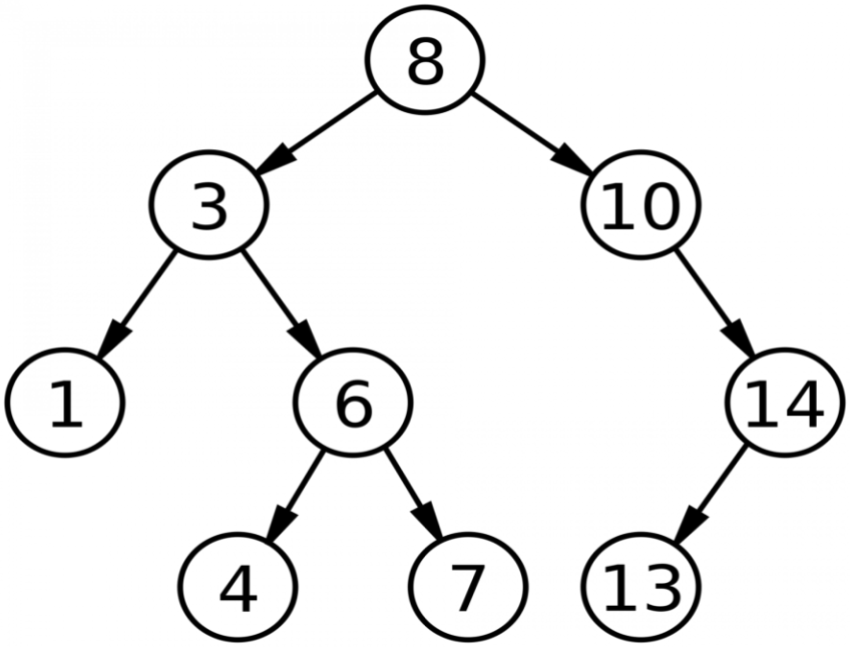

Рис. 1(бинарное дерево)

- Код программы:

```
#include <stdio.h>
#include <malloc.h>

//описание структуры, которая является элементом бинарного дерева
typedef struct binary_tree
{
   int key;                    //значение узла дерева
   struct binary_tree* left;   //адрес левого потомка
   struct binary_tree* right;  //адрес правого потомка
} node;//новый тип переменных




node* create_node(int key)
{
   node* tmp;

   //создание нового элемента типа node
   //переменная tmp хранит адрес этого нововго элемента
   tmp = (node*) malloc(sizeof(node));
   //число в поле key
   tmp->key = key;
   //потомков у нового элемента нет, поэтому 0
   tmp->left = NULL;
   tmp->right = NULL;
   return tmp;
```

```
}
//функция добавления нового элемента
//параметры:
//cr -адрес элемента, при запуске из функции main равен адресу первого элемента root
//key - значение нового элемента
node* add(node* cr, int key) {


   if (key < cr->key) {
      //если значение нового элемента < значения текущего элемента дерева
      if (cr->left == NULL) {
         //и левый адрес  cr = 0,
         //то новый элемент  будет левым потомком
         //создается новый элемент и его адрес в левый адрес
         return cr->left = create_node(key);
      }
      else {
         //если левый адрес  cr не равен 0, т.е. левый потомок уже есть
         //еще раз обращаемся к этой же функции
         //с параметром cr = этому левому адресу
         return add(cr->left, key);
      }
   }

   if (key >= cr->key) {
      //если значение нового элемента >= значения текущего элемента дерева
      if (cr->right == NULL) {
         //и правый адрес  cr = 0,
         //то новый элемент  будет правым потомком
         //создается новый элемент и его адрес в правый адрес
         return cr->right = create_node(key);
      }
      else {
         //если правый адрес  cr не равен 0, т.е. правый потомок уже есть
         //еще раз обращаемся к этой же функции
         //с параметром cr = этому правому адресу
         return add(cr->right, key);
      }
   }

   return NULL;
}

node* search(node* cr, int key) //поиск элемента со значением key
{
   if ((cr == NULL) || (cr->key == key))
      return cr;
   if (key < cr->key)
      return search(cr->left, key);
   else return search(cr->right, key);
}
void print_tree(node* current_node)
```

```
{
   if (current_node->left != NULL)
      //если у очередного элемента адрес левого потомка не равен 0
      //переходим в эту же функцию с адресом этого потомка
      print_tree(current_node->left);
   //сюда попадаем, если левый адрес =0
   //печатаем значение
   printf("Node %d\n", current_node->key);
   //проверяем правый адрес  у этого элемента
   //если он не равен 0
   //переходим в эту же функцию с адресом этого потомка
   if (current_node->right != NULL)
      print_tree(current_node->right);
}


void main(void)
{
   FILE* f;
   int srch, i;
   node* root = NULL;//адрес первого элемента дерева
   node* s;
   int element;
   //открытие массива для чтения
   fopen_s(&f, "massiv.txt", "r");

   //пока не достигнем конца файла
   while (!feof(f))
   {
      // читаем из файла целое число
      if (fscanf_s(f, "%d", &element) > 0)
      {
        // printf("%d\n", element); - для вывода файла чтобы убедиться что он читается
         if (root == NULL)
            //если дерево пустое, добавляем первый элемент
            root = create_node(element);
         else
            //если в дереве есть хотя бы один элемент, добавляем еще
            add(root, element);

      }
   }
   //распечатываем дерево
   print_tree(root);
   //бесконечный цикл
   while (1)
   {
      //приглашение на поиск
      printf("\n search: ");
      //принимаем значение (целое число), которое нужно найти
      scanf_s("%d", &srch);
      //поиск, адрес найденного элемента дерева в переменной s
```

```
        s = search(root, srch);
        if (!s)
           //если адрес = 0, элемент не найден
           printf("no search\n");
        else {
           //элемент найден, печать числа
           printf("%d \n", s->key);
           if (s->left)
              //если адрес его левого потомка не равна 0, печатаем значение этого потомка
              printf("Left %d", s->left->key);
           else
              //если адрес его левого потомка  равн0 0, печатаем прочерк
              printf("Left --");
           if (s->right)
              //если адрес его правого потомка не равна 0, печатаем значение этого потомка
              printf("  Right %d\n", s->right->key);
           else
              //если адрес его правого потомка  равн0 0, печатаем прочерк
              printf("  Right --\n");
        }
     }

}
}
```