

Отчёт по лабораторной работе №23
по курсу «Языки и методы программирования».
Выполнил студент группы М8О-111Б-23: Тимофеева Ирина
Александровна
№ по списку 21

Контакты: irinka.timofeeva0505@gmail.com

Работа выполнена: «1» мая 2023 г.

Преподаватель: каф. 806 Никулин Сергей Петрович

Входной контроль знаний с оценкой: _____

Отчет сдан «1» мая 2023 г.

Итоговая оценка: _____

Подпись преподавателя: _____

1. Тема: динамические структуры данных, обработка деревьев.
2. Цель работы: составить программу на языке Си для построения и обработки дерева двоичного вида, содержащего узлы типа int.
3. Задание: 24 — определить глубину дерева.
4. Оборудование: Оборудование ПЭВМ студента, если использовалось: Процессор AMD Ryzen 5 5600H, ОП 16 ГБ, SSD 250 ГБ, мониторы 15" Full Hd Display и 27" BenQ BL2780T. Другие устройства: принтер Canon MG4520S, мышь Logitech g403, наушники HyperX Cloud
5. Программное обеспечение: Программное обеспечение ПЭВМ студента, если использовалось:
Операционная система семейства Ubuntu, наименование версия VirtualBox Ubuntu 20.04.3
интерпретатор команд bash версия 5.0.17. Система программирования C.
Редактор текстов
VI версия 8.1
6. Идея, метод, алгоритм решения задачи [в формах: словесной, псевдокода, графической (блок-схема, диаграмма, рисунок, таблица) или формальные спецификации с пред- и постусловиями]:
Составить программу на языке Си, выполняющую четыре действия: добавление нового узла, визуализация дерева,

удаление узла, вычисление некоторой функции от дерева в соответствии с заданным вариантом.

7. Сценарий выполнения работы (план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию):

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

//дерево общего вида
typedef struct tree
{
    int data;                //данные
    struct tree **childs;    //потомки
    struct tree *parent;     //родитель
    unsigned count;         //число потомков
} Tree;
```

```
int const TreeSize = sizeof(Tree);
```

```
//инициализация дерева
Tree *init(Tree *p, int info)
{
    p->data = info;
    p->count = 0;
    p->childs = NULL;
    p->parent = NULL;
    return p;
}
```

```
// Функция удаления поддеревы
void freemem(Tree *p)
{
    if (p != NULL)
    {
        int i;
        for (i = 0; i < p->count; i++)
        {
            freemem(p->childs[i]);
        }
        if (p->parent != NULL)
        {
            int c = -1;
            for (i = 0; i < p->parent->count; i++)
```

```

    {
        //найдем индекс удаляемого потомка
        if (p->parent->childs[i] == p)
        {
            c = i;
        }
        //перепривяжем потомков после удаляемого узла
        if (c >= 0 && c < p->parent->count-1)
        {
            p->parent->childs[c] = p->parent->childs[c+1];
            c++;
        }
    }
    p->parent->count--;
}
free(p);
}
}

```

//Вывод ошибки создания дерева

```

void error(int c)
{
    printf("\nError: ");
    switch (c)
    {
        case 1:
            printf(" no memory");
            break;
        default:
            printf(" unknown");
            break;
    }
    exit(c);
}

```

//вспомогательный метод вывода ребер дерева

```

void showLine(char *c, int p, int s)
{
    int t = s;
    int i;
    for (i = 0; i < p; i++)
    {
        printf(t & 1 ? "| " : " ");
        t /= 2;
    }
}

```

```

    printf("%s", c);
}

//печать дерева
void showTree(Tree *wood, int p, int s)
{
    if (wood == NULL)
    {
        printf("no data!\n");
        return;
    }
    if (wood->parent)
        printf("%d (parent %d)", wood->data, wood->parent->data);
    else
        printf("%d", wood->data);
    printf("\n");

    int i;
    for (i = 0; i < wood->count; i++)
    {
        showLine("\n", p, s);
        showLine("L: ", p, s);
        showTree(wood->childs[i], p + 1, s + ((i < wood->count-1 ? 1 : 0) << p));
    }
}

//поиск узла дерева по номеру
Tree *searchnode(Tree *t, int info)
{
    if (t->data == info)
        return t;
    else if (t->count > 0)
    {
        int i;
        for (i = 0; i < t->count; i++)
        {
            Tree *p = searchnode(t->childs[i], info);
            if (p != NULL)
                return p;
        }
        return NULL;
    }
    else
        return NULL;
}

```

//добавление узла дерева

Tree *addnode(Tree *ptr, int parentinfo, int info)

```
{
    Tree *p = searchnode(ptr, parentinfo);
    Tree *p_child;
    if (p != NULL)
    {
        if (p->childs == NULL)
        {
            p->childs = (Tree **)malloc(sizeof(Tree *));
            if (p->childs == NULL)
                error(1);
            else
            {
                p->childs[0] = (Tree *)malloc(TreeSize);
                if (p->childs[0] == NULL)
                    error(1);
                (p->count) = 1;
            }
        }
        else
        {
            p->childs = (Tree **)realloc(p->childs, sizeof(Tree *) * (p->count + 1));
            if (p->childs == NULL)
                error(1);
            p->childs[p->count] = (Tree *)malloc(TreeSize);
            if (p->childs[p->count] == NULL)
                error(1);
            (p->count)++;
        }
        p_child = init(p->childs[p->count - 1], info);
        p_child->parent = p;
        return p_child;
    }
    return NULL;
}
```

// глубина дерева

int deepTree(Tree *wood)

```
{
    int deep, d;

    deep = 0;
    if (wood != NULL)
```

```

{
    int i;
    for (i = 0; i < wood->count; i++)
    {
        d = deepTree(wood->childs[i]) + 1;
        if (d > deep)
            deep = d;
    }
}
return deep;
}

```

```

int main()
{
    Tree head, *temp, *ptr;
    ptr = &head;
    char c;
    int i, j, k, l, m, n, o, d1, d2, d3, d4, d5;
    int done = 0;
    //инициализация генератора случайных чисел
    srand(time(NULL));
    while (!done)
    {
        printf("Main menu:\n");
        printf("1 - make binary tree\n");
        printf("2 - print tree\n");
        printf("3 - add node\n");
        printf("4 - delete node\n");
        printf("5 - find deep of tree\n");
        printf("0 - exit\n");
        printf("Enter command: ");
        c = getchar();
        getchar();
        printf("\n-----\n");
        switch (c)
        {
            case '0':
                done = 1;
                break;
            case '1':
                init(ptr, 0);
                for (i = 0; i < rand() % 4 + 1; i++)
                {
                    d1 = rand() % 10 + 1;
                    addnode(ptr, 0, d1);
                }
            }
        }
    }
}

```

```

        for (j = 0; j < rand() % 4 + 1; j++)
        {
            d2 = rand() % 100 + 10;
            addnode(ptr, d1, d2);
            for (k = 0; k < rand() % 4 + 1; k++)
            {
                d3 = rand() % 100 + 100;
                addnode(ptr, d2, d3);
            }
        }
        printf("\n");
        break;
case '2':
    printf("Tree is:\n");
    showTree(ptr, 1, 1);
    printf("\n");
    break;
case '3':
    printf("enter node data to add: ");
    scanf("%d", &d1);
    if (d1)
    {
        //getchar();
        printf("enter root to add: ");
        scanf("%d", &d2);
        if (d2)
        {
            addnode(ptr, d2, d1);
            printf("node added to tree!\n");
            getchar();
        }
    }
    printf("\n");
    break;
case '4':
    printf("enter node data to delete: ");
    scanf("%d", &d1);
    if (d1)
    {
        getchar();
        temp = searchnode(ptr, d1);
        if (temp != NULL)
            freemem(temp);
        printf("node deleted from tree!\n");
    }

```

```

    }
    printf("\n");
    break;
case '5':
    printf("Tree's deep is %d\n", deepTree(ptr));
    printf("\n");
    break;
default:
    printf("incorrect input!\n");
    printf("\n");
}
}

//удалить дерево из памяти
freemem(ptr);
return 0;
}

```

8. Распечатка протокола (подклеить листинг окончательного варианта программы с тестовыми:

```

#листинг программы работы с деревом общего вида
irina@Irina-VivoBook:~/Prog/Prog_C/Lab23$ cat GeneralTree.c
#include <stdio.h>
#include <stdlib.h>

//дерево общего вида
typedef struct tree
{
    int data;                //данные
    struct tree **childs;    //потомки
    struct tree *parent;     //родитель
    unsigned count;          //число потомков
} Tree;

int const TreeSize = sizeof(Tree);

//инициализация дерева
Tree *init(Tree *p, int info)
{
    p->data = info;
    p->count = 0;
    p->childs = NULL;
    p->parent = NULL;
    return p;
}

```



```

}

// Функция удаления поддерева
void freemem(Tree *p)
{
    if (p != NULL)
    {
        int i;
        for (i = 0; i < p->count; i++)
        {
            freemem(p->childs[i]);
        }
        // printf("free node: %d\n", p->data);
        if (p->parent != NULL)
        {
            int c = -1;
            for (i = 0; i < p->parent->count; i++)
            {
                //найдем индекс удаляемого потомка
                if (p->parent->childs[i] == p)
                {
                    c = i;
                }
                //перепривяжем потомков после удаляемого узла
                if (c >= 0 && c < p->parent->count-1)
                {
                    p->parent->childs[c] = p->parent->childs[c+1];
                    c++;
                }
            }
            p->parent->count--;
        }
        free(p);
    }
}

```

```

//Вывод ошибки создания дерева
void error(int c)
{
    printf("\nError: ");
    switch (c)
    {
        case 1:
            printf(" no memory");
            break;
    }
}

```

```

default:
    printf(" unknown");
    break;
}
exit(c);
}

//вспомогательный метод вывода ребер дерева
void showLine(char *c, int p, int s)
{
    int t = s;
    int i;
    for (i = 0; i < p; i++)
    {
        printf(t & 1 ? "| " : " ");
        t /= 2;
    }
    printf("%s", c);
}

//печать дерева
void showTree(Tree *wood, int p, int s)
{
    if (wood == NULL)
    {
        printf("no data!\n");
        return;
    }
    if (wood->parent)
        printf("%d (parent %d)", wood->data, wood->parent->data);
    else
        printf("%d", wood->data);
    printf("\n");

    int i;
    for (i = 0; i < wood->count; i++)
    {
        showLine("\n", p, s);
        showLine("L: ", p, s);
        showTree(wood->childs[i], p + 1, s + ((i < wood->count-1 ? 1 : 0) << p));
    }
}

//поиск узла дерева по номеру
Tree *searchnode(Tree *t, int info)

```

```

{
    if (t->data == info)
        return t;
    else if (t->count > 0)
    {
        int i;
        for (i = 0; i < t->count; i++)
        {
            Tree *p = searchnode(t->childs[i], info);
            if (p != NULL)
                return p;
        }
        return NULL;
    }
    else
        return NULL;
}

```

//добавление узла дерева

Tree *addnode(Tree *ptr, int parentinfo, int info)

```

{
    Tree *p = searchnode(ptr, parentinfo);
    Tree *p_child;
    if (p != NULL)
    {
        if (p->childs == NULL)
        {
            p->childs = (Tree **)malloc(sizeof(Tree *));
            if (p->childs == NULL)
                error(1);
            else
            {
                p->childs[0] = (Tree *)malloc(TreeSize);
                if (p->childs[0] == NULL)
                    error(1);
                (p->count) = 1;
            }
        }
    }
    else
    {
        p->childs = (Tree **)realloc(p->childs, sizeof(Tree *) * (p->count + 1));
        if (p->childs == NULL)
            error(1);
        p->childs[p->count] = (Tree *)malloc(TreeSize);
        if (p->childs[p->count] == NULL)

```

```

        error(1);
        (p->count)++;
    }
    p_child = init(p->childs[p->count - 1], info);
    p_child->parent = p;
    return p_child;
}
return NULL;
}

```

```

// глубина дерева
int deepTree(Tree *wood)
{
    int deep, d;

    deep = 0;
    if (wood != NULL)
    {
        int i;
        for (i = 0; i < wood->count; i++)
        {
            d = deepTree(wood->childs[i]) + 1;
            if (d > deep)
                deep = d;
        }
    }
    return deep;
}

```

```

int main()
{
    Tree head, *temp, *ptr;
    ptr = &head;
    char c;
    int i, j, k, l, m, n, o, d1, d2, d3, d4, d5;
    int done = 0;
    //инициализация генератора случайных чисел
    srand(time(NULL));
    while (!done)
    {
        printf("Main menu:\n");
        printf("1 - make binary tree\n");
        printf("2 - print tree\n");
        printf("3 - add node\n");
        printf("4 - delete node\n");
    }
}

```

```

printf("5 - find deep of tree\n");
printf("0 - exit\n");
printf("Enter command: ");
c = getchar();
getchar();
printf("\n-----\n");
switch (c)
{
case '0':
    done = 1;
    break;
case '1':
    init(ptr, 0);
    for (i = 0; i < rand() % 4 + 1; i++)
    {
        d1 = rand() % 10 + 1;
        addnode(ptr, 0, d1);
        for (j = 0; j < rand() % 4 + 1; j++)
        {
            d2 = rand() % 100 + 10;
            addnode(ptr, d1, d2);
            for (k = 0; k < rand() % 4 + 1; k++)
            {
                d3 = rand() % 100 + 100;
                addnode(ptr, d2, d3);
            }
        }
    }
    printf("\n");
    break;
case '2':
    printf("Tree is:\n");
    showTree(ptr, 1, 1);
    printf("\n");
    break;
case '3':
    printf("enter node data to add: ");
    scanf("%d", &d1);
    if (d1)
    {
        //getchar();
        printf("enter root to add: ");
        scanf("%d", &d2);
        if (d2)
        {

```

```

        addnode(ptr, d2, d1);
        printf("node added to tree!\n");
        getchar();
    }
}
printf("\n");
break;
case '4':
    printf("enter node data to delete: ");
    scanf("%d", &d1);
    if (d1)
    {
        getchar();
        temp = searchnode(ptr, d1);
        if (temp != NULL)
            freemem(temp);
        printf("node deleted from tree!\n");
    }
    printf("\n");
    break;
case '5':
    printf("Tree's deep is %d\n", deepTree(ptr));
    printf("\n");
    break;
default:
    printf("incorrect input!\n");
    printf("\n");
}
}

//удалить дерево из памяти
freemem(ptr);
return 0;
}

```

#компиляция программы

irina@Irina-VivoBook:~/Prog/Prog_C/Lab23\$ gcc GeneralTree.c

#запуск программы

irina@Irina-VivoBook:~/Prog/Prog_C/Lab23\$./a.out

Main menu:

1 - make binary tree

2 - print tree

3 - add node
4 - delete node
5 - find deep of tree
0 - exit

Enter command: 1

#создание дерева общего вида из случайного набора узлов

Main menu:

1 - make binary tree
2 - print tree
3 - add node
4 - delete node
5 - find deep of tree
0 - exit

Enter command: 2

#печать дерева

Tree is:

0

```
| |
| L: 9 (parent 0)
| | |
| | L: 109 (parent 9)
| | |
| | L: 155 (parent 109)
| | |
| | L: 151 (parent 109)
| | |
| | L: 194 (parent 109)
| |
| L: 2 (parent 0)
| | |
| | L: 77 (parent 2)
| | | |
| | | L: 135 (parent 77)
| | | |
| | | L: 171 (parent 77)
| | | |
| | | L: 185 (parent 77)
| | |
| | L: 74 (parent 2)
| | |
```

```
| | L: 181 (parent 74)
| |
| L: 3 (parent 0)
| |
| L: 14 (parent 3)
| | |
| | L: 171 (parent 14)
| |
| L: 47 (parent 3)
| |
| L: 176 (parent 47)
| |
| L: 173 (parent 47)
```

Main menu:

- 1 - make binary tree
- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 3

#добавление листа 225 в узел дерева 77

enter node data to add: 225
enter root to add: 77
node added to tree!

Main menu:

- 1 - make binary tree
- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 2

#печать получившегося дерева

Tree is:

```
0
| |
| L: 9 (parent 0)
| | |
```



```

| | L: 109 (parent 9)
| | |
| | L: 155 (parent 109)
| | |
| | L: 151 (parent 109)
| | |
| | L: 194 (parent 109)
| |
| L: 2 (parent 0)
| | |
| | L: 77 (parent 2)
| | | |
| | | L: 135 (parent 77)
| | | |
| | | L: 171 (parent 77)
| | | |
| | | L: 185 (parent 77)
| | | |
| | | L: 225 (parent 77)
| | |
| | L: 74 (parent 2)
| | |
| | L: 181 (parent 74)
| |
| L: 3 (parent 0)
| |
| L: 14 (parent 3)
| | |
| | L: 171 (parent 14)
| |
| L: 47 (parent 3)
| | |
| | L: 176 (parent 47)
| | |
| | L: 173 (parent 47)

```

#Вывод глубины дерева

Main menu:

- 1 - make binary tree
- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 5

Tree's deep is 3

Main menu:

- 1 - make binary tree
- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 3

#добавление листа 55 в узел дерева 181

enter node data to add: 55
enter root to add: 181
node added to tree!

Main menu:

- 1 - make binary tree
- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 2

#печать получившегося дерева

Tree is:

0

| |

| L: 9 (parent 0)

| | |

| | L: 109 (parent 9)

| | |

| | L: 155 (parent 109)

| | |

| | L: 151 (parent 109)

| | |

| | L: 194 (parent 109)

| |

| L: 2 (parent 0)

```

| | |
| | L: 77 (parent 2)
| | | |
| | | L: 135 (parent 77)
| | | |
| | | L: 171 (parent 77)
| | | |
| | | L: 185 (parent 77)
| | | |
| | | L: 225 (parent 77)
| | |
| | L: 74 (parent 2)
| | |
| | L: 181 (parent 74)
| | |
| | L: 55 (parent 181)
| |
| L: 3 (parent 0)
| |
| L: 14 (parent 3)
| | |
| | L: 171 (parent 14)
| | |
| L: 47 (parent 3)
| |
| L: 176 (parent 47)
| |
| L: 173 (parent 47)

```

#ВЫВОД глубины дерева

Main menu:

1 - make binary tree

2 - print tree

3 - add node

4 - delete node

5 - find deep of tree

0 - exit

Enter command: 5

Tree's deep is 4

Main menu:

1 - make binary tree

- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 4

#удаление узла с номером 74

enter node data to delete: 74
node deleted from tree!

Main menu:

- 1 - make binary tree
- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 2

#печать получившегося дерева

Tree is:

0

| |

| L: 9 (parent 0)

| | |

| | L: 109 (parent 9)

| | |

| | L: 155 (parent 109)

| | |

| | L: 151 (parent 109)

| | |

| | L: 194 (parent 109)

| |

| L: 2 (parent 0)

| | |

| | L: 77 (parent 2)

| | |

| | L: 135 (parent 77)

| | |

| | L: 171 (parent 77)

| | |

| | L: 185 (parent 77)

```

| | |
| | L: 225 (parent 77)
| |
| L: 3 (parent 0)
| |
| L: 14 (parent 3)
| | |
| | L: 171 (parent 14)
| |
| L: 47 (parent 3)
| |
| L: 176 (parent 47)
| |
| L: 173 (parent 47)

```

Main menu:

- 1 - make binary tree
- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 4

#удаление узла с номером 2

 enter node data to delete: 2
 node deleted from tree!

Main menu:

- 1 - make binary tree
- 2 - print tree
- 3 - add node
- 4 - delete node
- 5 - find deep of tree
- 0 - exit

Enter command: 2

#печать получившегося дерева

 Tree is:

```

0
| |
| L: 9 (parent 0)

```

```

| | |
| | L: 109 (parent 9)
| | |
| | L: 155 (parent 109)
| | |
| | L: 151 (parent 109)
| | |
| | L: 194 (parent 109)
| |
| L: 3 (parent 0)
| |
| L: 14 (parent 3)
| | |
| | L: 171 (parent 14)
| | |
| L: 47 (parent 3)
| |
| L: 176 (parent 47)
| |
| L: 173 (parent 47)

```

9. Дневник отладки должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

10. Замечания автора по существу работы: замечания отсутствуют.

11. Выводы: научился работать с двоичными деревьями в СП Си.

Подпись студента _____