

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: «Конструирование программ»

ОТЧЕТ

к лабораторной работе №8

на тему:

«ИНТЕГРАЦИЯ АССЕМБЛЕРНЫХ ПРЕРЫВАНИЙ В ПРОЕКТЫ НА C++»

БГУИР 1-40 04 01

Выполнил студент группы 253505
БЕКАРЕВ Станислав Сергеевич

(дата, подпись студента)

Проверил ассистент кафедры
информатики
РОМАНЮК Максим Валерьевич

(дата, подпись преподавателя)

Минск 2023

Цель работы. Получить понимание принципов работы DOS-прерываний и методов их использования в программировании на языке C++.

В рамках выполнения лабораторной работы должно быть выполнено следующее задание:

Бинарное дерево поиска с автоматическим бэкапом:

На стороне Assembler:

Реализация алгоритмов для балансировки бинарного дерева поиска, чтобы обеспечить оптимальное время поиска.

При каждой операции с деревом (добавление, удаление) автоматически создается резервная копия дерева в файле.

Добавьте функционал восстановления дерева из последнего бэкапа.

Реализацию всего дерева и его функций выполнить в assembler-коде.

Сохранения дерева в файл надо произвести с использованием сериализации.

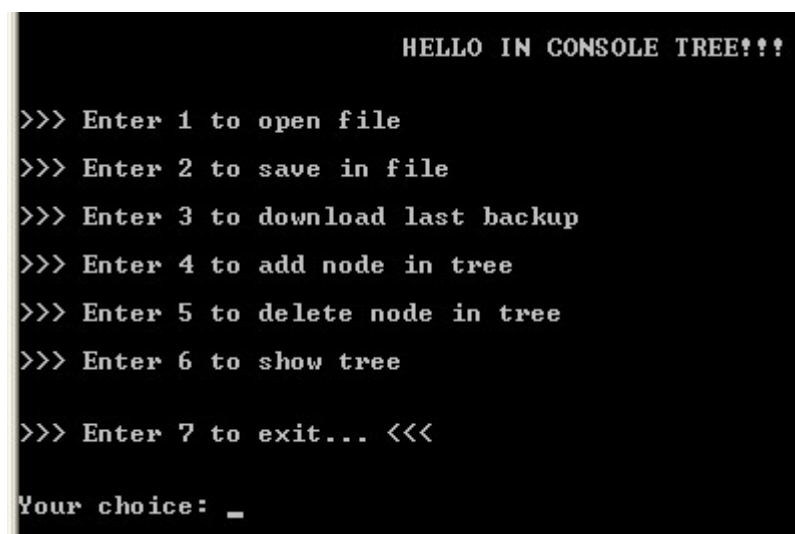
Загрузку дерева из файла надо произвести с использованием десериализации.

На стороне C++:

Управление функциями ассемблера: инициализация дерева, добавление и удаление элементов, балансировка дерева. Сериализация и десериализация дерева при сохранении и загрузке. Функции для создания бэкапов и восстановления из них..

Ход работы: Для выполнения задания был написан программный код на языке Assembler, представленный в листинге №1.

Данный код инициализирует дерево, имеет метод добавления нод в дерево и их удаления. При этом соблюдается условия баланса дерева, при нарушении которых дерево самобалансируется. Так же добавлены функции сохранения дерева в файл и загрузка дерева из файла. Работа программы показана на рисунках 1, 2, 3.



```
HELLO IN CONSOLE TREE!!!

>>> Enter 1 to open file
>>> Enter 2 to save in file
>>> Enter 3 to download last backup
>>> Enter 4 to add node in tree
>>> Enter 5 to delete node in tree
>>> Enter 6 to show tree
>>> Enter 7 to exit... <<<

Your choice: _
```

Рисунок 1 – Главное меню программы.

```

How many nodes will you add(max - 31): 5
node 1: 0
node 2: 2
node 3: 3
node 4: 1
node 5: 4
Finish!..._

```

Рисунок 2 – Добавление/удаление нод.

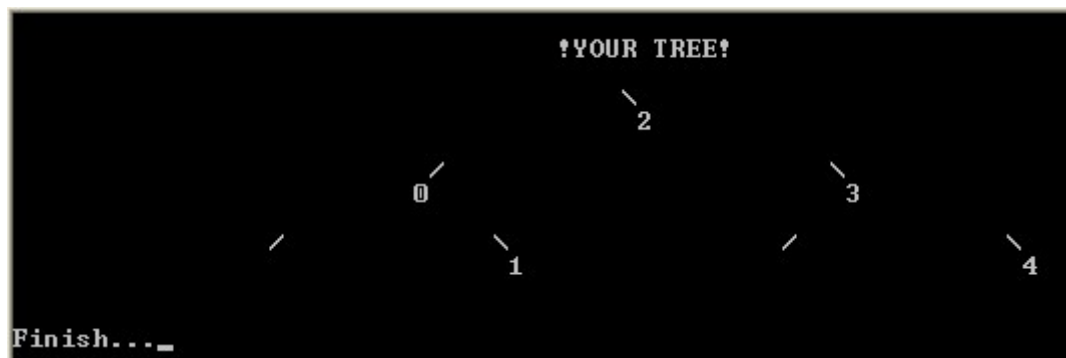


Рисунок 3 – Отображение дерева.

Листнинг №1 – Исходный код задания

```

.model small
.data
path_found db "Success!!...$"
path_unfound db "ERROR: Path Not Found. Try Again Please...$"
file_number db 2 DUP(0) ; - number for read/write in file
width_tree dw 255 DUP(0) ; - width_traversal (after save in file)
first_node dw 1 DUP(0) ; = offset tree
last_node dw 1 DUP(0); = first_node + size
control_block dw 4 DUP(0); this node control exist of root
tree dw 1020 DUP(0) ; 255 - max amount of nodes, after -
UndefindBehavior
;struct node:
; left*
; right*
; value
; height - FFFFh(ABCD): A - left exist?, B - right exist?, CD - height

.code
add_rec proc ; reg di - *node
    mov ax, [bp + 6]
    cmp ax, [di + 4]
    jng left

    right:
        mov ax, [di + 6]
        shl ax, 4
        shr ax, 12
        cmp ax, 0001

```

```

jne no_node_r
node_r:
    push di
    mov di, [di + 2]
    call add_rec
    pop di
    jmp end_no_node_r
no_node_r:
    cmp [di + 2], 0000h
    jne not_empty_r
    ;empty:
    mov ax, last_node
    mov [di + 2], ax
    add last_node, 8
    add [di + 6], 0100h
    push di
    mov di, [di + 2]
    mov [di], 0000h
    mov [di + 2], 0000h
    mov ax, [bp + 6]
    mov [di + 4], ax
    mov [di + 6], 0000h
    pop di
    jmp end_no_node_r
not_empty_r:
    add [di + 6], 0100h
    push di
    mov di, [di + 2]
    mov ax, [bp + 6]
    mov [di + 4], ax
    mov [di + 6], 0000h
    pop di
end_no_node_r:
    jmp end_left
left:
    mov ax, [di + 6]
    shr ax, 12
    cmp ax, 0001
    jne no_node_l
node_l:
    push di
    mov di, [di]
    call add_rec
    pop di
    jmp end_no_node_l
no_node_l:
    cmp [di], 0000h
    jne not_empty_l
    ;empty:
    mov ax, last_node
    mov [di], ax
    add last_node, 8
    add [di + 6], 1000h

```

```

        push di
        mov di, [di]
        mov [di], 0000h
        mov [di + 2], 0000h
        mov ax, [bp + 6]
        mov [di + 4], ax
        mov [di + 6], 0000h
        pop di
    jmp end_no_node_1
not_empty_1:
    add [di + 6], 1000h
    push di
    mov di, [di]
    mov ax, [bp + 6]
    mov [di + 4], ax
    mov [di + 6], 0000h
    pop di
end_no_node_1:
end_left:
    call get_height ; update height
    mov bx, [di + 6]
    mov bl, al
    mov [di + 6], bx
    call balance
    ret
add_rec endp

balance proc ; reg di - *node
    call get_balanced
    cmp ax, -2 ; r_h > l_h --> left_rotate
    jne r_rotate
        ; right_left_rotate?
        push di
        mov di, [di + 2]
        call get_balanced
        pop di
        cmp ax, 1
        jne only_l_rotate
            ; right_left_rotate:
            push di
            mov di, [di + 2]
            call right_rotate
            pop di
        only_l_rotate:
        call left_rotate
        jmp end_balance
    r_rotate:
    cmp ax, 2 ; r_h < l_h --> right_rotate
    jne end_balance
        ; left_right_rotate?
        push di
        mov di, [di]
        call get_balanced

```

```

        pop di
        cmp ax, -1
        jne only_r_rotate
            ; left_right_rotate:
            push di
            mov di, [di]
            call left_rotate
            pop di
        only_r_rotate:
        call right_rotate
    end_balance:
    ret
balance endp

right_rotate proc ; di - *node
    mov si, [di] ; si - *left_n (== node->left)
    mov ax, 1
    call swap_value_node
        ; node->left = left_n->left
    mov ax, [si]
    mov [di], ax
        ; left_n->left = left_n->right
    mov ax, [si + 2]
    mov [si], ax
        ; left_n->right = node->right
    mov ax, [di + 2]
    mov [si + 2], ax
        ; node->right = left_n (== node->left)
    mov [di + 2], si

    ; update height
    push di
    mov di, si
    call get_height ; update height left_n
    mov bx, [di + 6]
    mov bl, al
    mov [di + 6], bx
    pop di
    call get_height ; update height node
    mov bx, [di + 6]
    mov bl, al
    mov [di + 6], bx
    ret
right_rotate endp

left_rotate proc ; di - *node
    mov si, [di + 2] ; si - *right_n (== node->left)
    mov ax, -1
    call swap_value_node
        ; node->right = right_n->right
    mov ax, [si + 2]
    mov [di + 2], ax
        ; right_n->right = right_n->left

```

```

    mov ax, [si]
    mov [si + 2], ax
        ; right_n->left = node->left
    mov ax, [di]
    mov [si], ax
        ; node->left = right_n (== node->right)
    mov [di], si

    ; update height
    push di
    mov di, si
    call get_height ; update height right_n
    mov bx, [di + 6]
    mov bl, al
    mov [di + 6], bx
    pop di
    call get_height ; update height node
    mov bx, [di + 6]
    mov bl, al
    mov [di + 6], bx
    ret
left_rotate endp

find_node proc ; di - *node, si - *parent, [bp + 6] - deleted value
    mov ax, [bp + 6]
    cmp ax, [di + 4]
    je delete_node
        ; not equal
    go_next:
        mov ax, [bp + 6]
        cmp ax, [di + 4]
        jg go_right
    go_left:
        mov ax, [di + 6]
        shr ah, 4
        cmp ah, 1
        jne end_find_node
        mov ax, di
        mov di, [di]
        mov si, ax
        push si
        call find_node
        pop di
        jmp end_go_right
    go_right:
        mov ax, [di + 6]
        shl ah, 4
        shr ah, 4
        cmp ah, 1
        jne end_find_node
        ;push si
        push di
        mov ax, di

```

```

        mov di, [di + 2]
        mov si, ax
        call find_node
        pop di
        ;pop si
end_go_right:
jmp end_delete_node

; node found
delete_node:
    mov ax, [di + 6]
    cmp ah, 11h

    je rec_remove
        cmp ah, 00h
        jne parent_node_remove

    null_node_remove:
        call remove_null_node
        jmp end_rec_remove

    parent_node_remove:
        call remove_parent_node
        jmp end_rec_remove

    rec_remove:
        call remove_full_node
    end_rec_remove:
    jmp end_height_update

end_delete_node:

; update height and balance
end_find_node:
    call get_height
    mov bx, [di + 6]
    mov bl, al
    mov [di + 6], bx
    call balance
end_height_update:
    ret
find_node endp

```

Выводы: Была написана программа, которая реализует бинарное (АВЛ) дерево поиска с автоматическим бэкапом. Были изучены принципы работы DOS-прерываний и методов их использования в программировании на языке C++.