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 С++»**

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 десириализации.

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.



Рисунок 1 – Главное меню программы.

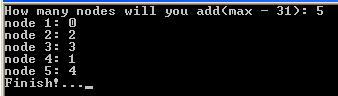


Рисунок 2 – Добавление/удаление нод.



Рисунок 3 – Отображение дерева.

Листнинг №1 – Исходный код задания

.model small

.data

path\_found db "Succes!!...$"

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\_l

not\_empty\_l:

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\_l:

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++.