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 №7

на тему:

**«**ИНТЕГРАЦИЯ АССЕМБЛЕРНЫХ ПРЕРЫВАНИЙ В ПРОЕКТЫ НА C++**»**

БГУИР 6-05-0612-02 67

|  |
| --- |
| Выполнил студент группы 353503  КОХАН Артём Игоревич |
|  |
| (дата, подпись студента) |
| Проверил ассистент каф. Информатики  РОМАНЮК Максим Валерьевич |
|  |
| (дата, подпись преподавателя) |

Минск 2024

# 1 Индивидуальное задание

**Задание 1. Вариант 7.** Реализовать управление функциями ассемблера: инициализация дерева, добавление и удаление элементов, балансировка дерева, чтобы обеспечить оптимальное время поиска. При каждой операции с деревом (добавление, удаление) автоматически создается резервная копия дерева в файле. Добавить функционал восстановления дерева из последнего бэкапа. Реализацию всего дерева и его функционала выполнить в assembler-коде. Сохранения дерева в файл надо произвести с использованием сериализации. Загрузка дерева из файла надо произвести с использованием десериализации.

# 2 Выполнение работы

Для реализации бинарного дерева поиска в памяти выделим 8 байт под значения узла, 8 байт под адрес левого потомка и 8 байт под адрес правого потомка. Балансировка реализована с помощью отсортированного массива элементов, а далее средний элемент массива выбран в качестве корня, левое поддерево создано из левой части массива, правое поддерево создано из правой части массива. Сложность также вызвала реализация удаления элементов из дерева, так как при удалении элемента необходимо было обновлять указатели на потомков, а также смещать элементы, которые были расположены за удалённым элементом в памяти. При вставке и удалении реализован алгоритм сериализации бинарного дерево поиска в файл, при необходимости можно загрузить дерево из файла путём десериализации дерева из него.

Листинг кода 1 – Функционал программы на языке ассемблера

global bst ; метка bst будет видимой извне

section .data

bufferInput: times 21 db 0, 0Ah

bufferInt db 21 dup(0)

bufferFileNum db 21 dup(0)

msRoot: db "Enter value root in range from -9 223 372 036 854 775 808 until 9 223 372 036 854 775 807: ", 0 ; 91 символов

invalid\_input\_msg: db "Invalid input", 0Ah, 0 ; 14 символов

info: db "Choise the comand:", 0Ah, "0 - to close program", 0Ah, "1 - insert node to tree", 0Ah, "2 - delete node from tree", 0Ah, "3 - output tree", 0Ah, "4 - back up tree", 0Ah, "5 - balance the tree", 0Ah ; 144 символов

insert\_value: db "Enter to insert value node in range from -9 223 372 036 854 775 808 until 9 223 372 036 854 775 807: ", 0 ; 101 символов

delete\_value: db "Enter to delete value node in range from -9 223 372 036 854 775 808 until 9 223 372 036 854 775 807: ", 0 ; 101 символов

myTree: db "Root -> Left -> Right:", 0Ah ; 23 символов

full\_tree\_msg: db "TREE IS FULL !!!", 0Ah ; 17 символов

not\_found\_msg: db "Element not found :(", 0Ah ; 21 символ

balance\_success\_msg: db "Success balance tree", 0Ah ; 21 символов

file\_name: db "tree.txt", 0 ; 8 символов

file\_error\_msg: db "Error to open file", 0Ah ; 22 символа

file\_invalid\_msg: db "Incorrect data in file", 0Ah ; 23 символа

file\_success\_msg: db "Success back up of tree", 0Ah ; 25 символов

size\_file dq 0

key\_file dq 0

separator: db 0Ah

node\_count dw 0 ; Число узлов дерева

last\_node dq 0 ; Адрес последней node

parent\_in\_delete dq 0

max\_nodes\_count equ 256

node\_count\_balance dw 0 ; будем сохранять кол-во элементов перед балансировкой

average\_index dq 0

section .bss

treeNode resb 6144 ; Выделение 6144 байт для 256 узлов

bufferFile resb 5120

sortedArray resq 256 ; Массив для хранения элементов (для балансировки)

arrayIndex resq 0 ; Индекс текущего элемента (для балансировки)

section .text

;;;-------------------------------MAIN PROGRAM-----------------------------------------------Starting

bst:

mov rax, 1 ; 1 - номер системного вызова функции write

mov rdi, 1 ; 1 - дескриптор файла стандартного вызова stdout

mov rsi, msRoot ; адрес строки для вывода

mov rdx, 91 ; количество байтов

syscall

mov rax, 0 ; sys\_read

mov rdi, 0 ; stdin

mov rsi, bufferInput

mov rdx, 21

syscall

mov rsi, bufferInput

call validate\_input

cmp bl, 2

je invalid\_input\_root

mov [rel treeNode], rax

inc byte [rel node\_count] ; Увеличиваем счетчик узлов

lea rax, [rel treeNode] ; mov rax, treeNode

mov [rel last\_node], rax

continue\_execution:

mov ax, [rel node\_count]

cmp ax, 0

je root\_insert

mov rax, 1

mov rdi, 1

mov rsi, info

mov rdx, 144

syscall

mov rax, 0 ; sys\_read

mov rdi, 0 ; stdin

mov rsi, bufferInput

mov rdx, 21

syscall

xor ax, ax

mov al, [rel bufferInput]

add al, [rel bufferInput + 1] ; num + 0Ah

cmp al, 58

jb .invalid\_input\_main

cmp al, 63

ja .invalid\_input\_main

cmp al, 59

je insertCall

cmp al, 60

je deleteCall

cmp al, 61

je start\_pre\_order\_traversal

cmp al, 62

je restore\_tree\_from\_file

cmp al, 63

je balance\_tree

mov rax, 60 ; Завершение программы

syscall

.invalid\_input\_main:

mov rax, 1

mov rdi, 1

mov rsi, invalid\_input\_msg

mov rdx, 14

syscall

jmp continue\_execution

;;;-------------------------------MAIN PROGRAM-----------------------------------------------Ending

;;;-------------------------------ROOT INSERT-----------------------------------------------Strting

root\_insert:

mov rax, 1 ; sys\_write

mov rdi, 1 ; stdout

mov rsi, msRoot

mov rdx, 91

syscall

mov rax, 0 ; sys\_read

mov rdi, 0 ; stdin

mov rsi, bufferInput

mov rdx, 21

syscall

mov rsi, bufferInput

call validate\_input

cmp bl, 2

je invalid\_input\_node

mov [rel treeNode], rax

inc byte [rel node\_count] ; Увеличиваем счетчик узлов

mov rcx, [rel last\_node]

add rcx, 24 ; Узлы занимают 24 байт

mov [rel last\_node], rcx

jmp continue\_execution

;;;-------------------------------ROOT INSERT-----------------------------------------------Ending

;;;-------------------------------CHECK INPUT------------------------------------------------Starting

invalid\_input\_node:

mov rax, 1

mov rdi, 1

mov rsi, invalid\_input\_msg

mov rdx, 14

syscall

jmp insertCall

invalid\_input\_root:

mov rax, 1

mov rdi, 1

mov rsi, invalid\_input\_msg

mov rdx, 14

syscall

jmp bst

validate\_input:

xor rax, rax

xor rbx, rbx

xor rcx, rcx

xor rdx, rdx

mov bl, 1 ; Флаг для отрицательного числа

lodsb ; Загрузить первый символ из буфера в al

cmp al, '-' ; Проверка на знак минуса

je check\_digits ; Если минус, перейти к проверке цифр

cmp al, 0Ah

je not\_number

mov bl, 0 ; Флаг для положительного числа

back\_to\_digits:

cmp al, 0Ah ; Проверка на конец строки (новая строка)

je end\_check ; Конец строки

cmp al, '0' ; Проверка, является ли это цифрой

jb not\_number

cmp al, '9'

ja not\_number

cmp bl, 1

je convertNEG\_to\_number

convert\_to\_number:

imul rdx, rdx, 10 ; Умножение текущего числа на 10

jo overflow

sub al, '0' ; Конвертация ASCII в число

add rdx, rax ; Добавление к общему

jo overflow

jmp check\_digits

convertNEG\_to\_number:

imul rdx, rdx, 10

jo overflow

sub al, '0'

sub rdx, rax

jo overflow

jmp check\_digits

check\_digits:

lodsb ; Загрузить следующий символ из буфера

jmp back\_to\_digits ; Повторить проверку цифр

not\_number:

mov bl, 2 ; Индикатор некорректного ввода

ret

end\_check:

mov rax, rdx ; Результат в rax

ret

overflow:

mov bl, 2 ; Индикатор переполнения

ret

;;;-------------------------------CHECK INPUT------------------------------------------------Ending

;;;-------------------------------INSERT NODE------------------------------------------------Starting

insertCall:

call start\_pre\_order\_traversal\_to\_file

mov ax, [rel node\_count]

cmp ax, max\_nodes\_count

je .full\_tree

mov rax, 1 ; sys\_write

mov rdi, 1 ; stdout

mov rsi, insert\_value

mov rdx, 101

syscall

mov rax, 0 ; sys\_read

mov rdi, 0 ; stdin

mov rsi, bufferInput

mov rdx, 21

syscall

mov rsi, bufferInput

call validate\_input

cmp bl, 2

je invalid\_input\_node

call insert

jmp continue\_execution

.full\_tree:

mov rax, 1

mov rdi, 1

mov rsi, full\_tree\_msg

mov rdx, 17

syscall

jmp continue\_execution

insert:

lea rsi, [rel treeNode] ;mov rsi, treeNode , Адрес node(начинаем с корня)

call insert\_recursive

ret

insert\_recursive:

mov rdx, rsi

lea rcx, [rel treeNode]

cmp rsi, rcx

jne no\_root\_insert

mov rdx, [rsi]

mov rdi, [rsi]

cmp rax, rdi

jl .left\_branch ; Идем влево, если значение < текущего узла

jge .right\_branch ; Идем вправо, если значение > текущего узла

ret

.left\_branch:

; Переход к левому потомку

add rsi, 8 ; Левый потомок

mov rdi, [rsi]

test rdi, rdi

jz create\_node

call insert\_recursive

ret

.right\_branch:

; Переход к правому потомку

add rsi, 16 ; Правый потомок

mov rdi, [rsi]

test rdi, rdi

jz create\_node

call insert\_recursive

ret

no\_root\_insert:

mov rdi, [rdx] ; значение ноды

test rdi, rdi

jz create\_node\_no\_root

mov rbx, [rdi]

mov rdi, rbx

cmp rax, rdi

jl .left\_branch\_no\_root ; Идем влево, если значение < текущего узла

jge .right\_branch\_no\_root ; Идем вправо, если значение > текущего узла

.left\_branch\_no\_root:

mov rdx, [rsi] ; aдрес ноды

add rdx, 8

mov rsi, rdx

call no\_root\_insert

ret

.right\_branch\_no\_root:

mov rdx, [rsi] ; aдрес ноды

add rdx, 16

mov rsi, rdx

call no\_root\_insert

ret

create\_node:

mov rcx, [rel last\_node]

add rcx, 24 ; Узлы занимают 24 байт

mov [rel last\_node], rcx

mov [rcx], rax ; Сохраняем значение

mov [rsi], rcx

inc byte [rel node\_count] ; Увеличиваем счетчик узлов

ret

create\_node\_no\_root:

mov rcx, [rel last\_node]

add rcx, 24 ; Узлы занимают 24 байт

mov [rel last\_node], rcx

mov [rcx], rax ; Сохраняем значение

mov [rdx], rcx

inc byte [rel node\_count] ; Увеличиваем счетчик узлов

ret

;;;-------------------------------INSERT NODE------------------------------------------------Ending

;;;-------------------------------OUTPUT TREE------------------------------------------------Starting

start\_pre\_order\_traversal:

mov rax, 1

mov rdi, 1

mov rsi, myTree

mov rdx, 23

syscall

xor rdi, rdi

lea rdi, [rel treeNode]

call .pre\_order\_traversal

jmp continue\_execution

.pre\_order\_traversal:

test rdi, rdi

jz done\_pre\_order\_traversal

; Обрабатываем текущий узел

mov rax, [rdi]

call print\_node\_value

; Обходим левое поддерево

add rdi, 8

mov rsi, [rdi]

sub rdi, 8

push rdi

test rsi, rsi

jz .skip\_left

mov rdi, rsi

call .pre\_order\_traversal

.skip\_left:

pop rdi

; Обходим правое поддерево

mov rsi, [rdi + 16]

test rsi, rsi

jz done\_pre\_order\_traversal

mov rdi, rsi

call .pre\_order\_traversal

done\_pre\_order\_traversal:

ret

print\_node\_value:

mov r8, rdi

; Преобразуем число в строку

mov rsi, bufferInt ; Указатель на буфер

call int\_to\_string

; Выводим строку на экран

mov rdx, rax ; Длина строки

mov rax, 1 ; sys\_write

mov rdi, 1 ; stdout

syscall

mov rax, 1 ; sys\_write

mov rdi, 1 ; stdout

mov rsi, separator

mov rdx, 1

syscall

mov rdi, r8

ret

int\_to\_string:

; Преобразует число в строку

; Вход: rax = значение

; Выход: rsi = строка, rax = длина строки

xor rbx, rbx ; по умолчанию число положительное

mov rcx, 10 ; Основание системы счисления

lea rdi, [rsi + 20] ; Конец буфера

mov byte [rdi], 0 ; Null-terminator

test rax, rax

jns .convert\_loop

neg rax

mov bl, '-'

.convert\_loop:

dec rdi

xor rdx, rdx

div rcx

add dl, '0'

mov [rdi], dl

test rax, rax

jnz .convert\_loop

; Если число было отрицательным, добавляем минус

cmp bl, '-'

jne .done

dec rdi

mov [rdi], bl

.done:

add rsi, 21

mov rax, rsi

sub rsi, 21

sub rax, rdi ; Длина строки

mov rsi, rdi ; Указатель на начало строки

ret

;;;-------------------------------OUTPUT TREE------------------------------------------------Ending

C++ позволяет вызывать функции assembler-a NASM в программе. Это позволяет оптимизировать код, написанный на С++. Для этого нам необходимо скомпилировать файл кода \*.asm в объектный файл: nasm -f elf64 BST.asm -o BST.o. С помощью компилятора gcc линкуем и запускаем программу на языке С++: gcc main.cpp BST.o -o main.

Листинг кода 2 – Функционал программы на языке С++

#include <iostream>

extern "C" void bst();

int main()

{

bst();

return 0;

}

# Вывод

В ходе лабораторной работы получено понимание принципов работы для интеграции ассемблерных вставок в проекты, написанные на С++ и принципы работы с прерываниями на операционной системе linux.