Национальный исследовательский ядерный университет «МИФИ»

ОТЧЁТ

о выполнении контрольного домашнего задания

«Связные списки и структуры, деревья»

Выполнил: студент 1 курса группы С19-103 Оленников В.Д.

Преподаватель: Пчелинцев А.В.

Москва 2020г.

**Текст** **задачи**

Необходимо разработать программу, которая реализует алгоритм выбора вариантов с помощьюдеревьев, построенных на структурах.

1. Для создания программы должны быть использованы связные списки структур.

2. Структуры образуют дерево с **тремя** и более уровнями, число пунктов меню от 2 до 5 плюс пункт возврата в меню предыдущего уровня.

3. После выбора пункта меню самого нижнего уровня программа завершает работу, и на экран выводится перечень выбранных пунктов. В случае отказа от данного меню (возврат в меню верхнего уровня) необходимо исключить из списка выбора пункт, соответствующий меню, в которое вернулась программа.

4. В программе используются четыре вида деревьев (пункт **«Вид дерева»**). Нужно реализовать один из них.

5. Деревья создаются динамически и подлежат редактированию (пункты **«Листья»** и

**«Поддеревья»**).

6. В созданном дереве провести расчет элементов (пункт **«Расчёт»**).

7. Продемонстрировать «путешествие» по уровням дерева, а также вывести на экран

информацию об элементах (пункт **«Вывод»**).

8. Название элементов должно соответствовать одной тематике (пункт **«Тема»**).

**Вариант 29**

А) **Вид дерева** – двоичное

Б) **Листья** - удалить «лист» дерева

В) **Поддеревья** - выделить поддерево из дерева (удалить все остальные элементы)

Г) **Расчёт** - подсчитать количество ветвей

Д) **Вывод** - ветви с минимальным числом узлов

Е) **Тема** - план работы менеджера футбольной команды

**Алгоритм решения**

1. Разработать алгоритмы обработки данных. Каждый алгоритм реализовать в отдельной

функции. Также должны быть дополнительные функции.

2. Подготовить алгоритм работы меню.

3. Подготовить алгоритм (блок-схему) работы программы.

4. Подготовить текст программы, провести отладку программы, используя возможности

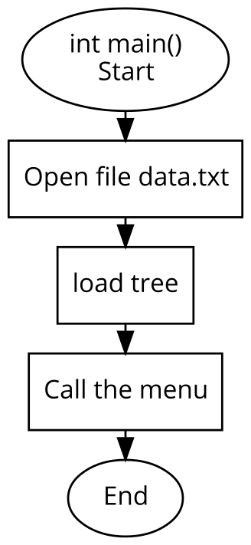
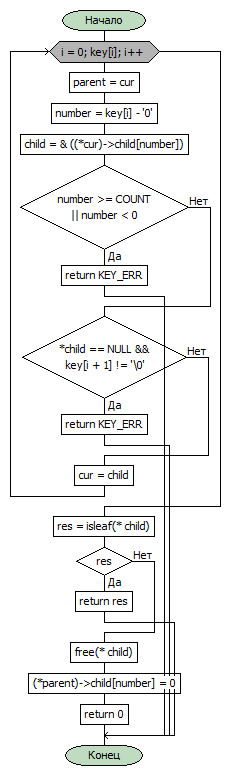
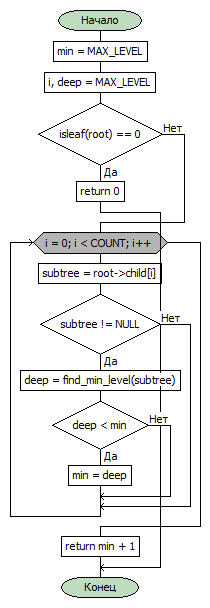
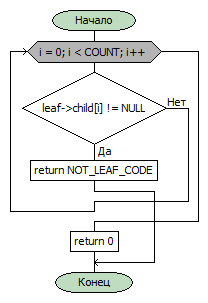
встроенного отладчика интегрированной среды программирования.

5. Подготовить два варианта программы: в виде отдельных функций, расположенных в одном файле; в виде библиотеки функций, расположенных в разных файлах.

6. Подготовить отчёт о проделанной работе

.

**Блок-схемы алгоритма**

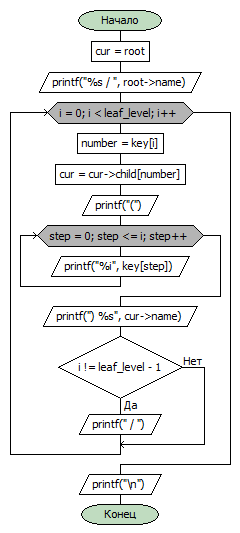
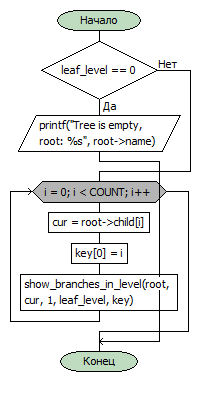
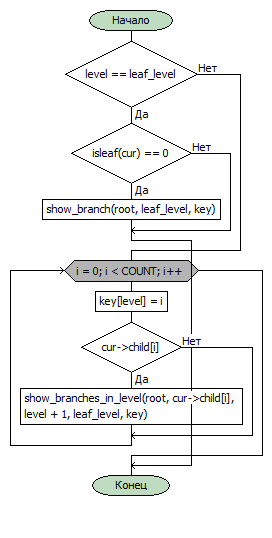
****

isleaf

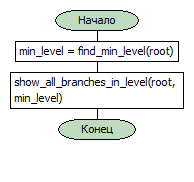
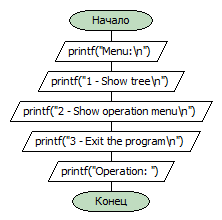
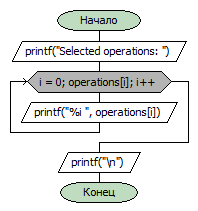
find\_min\_level

delete\_leaf

Функция int main()

****

show\_branch

****

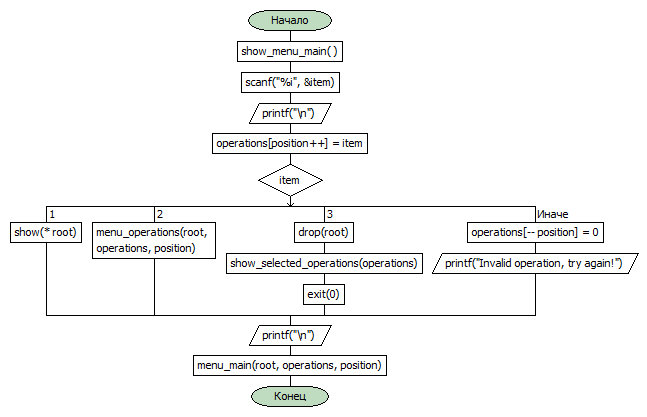
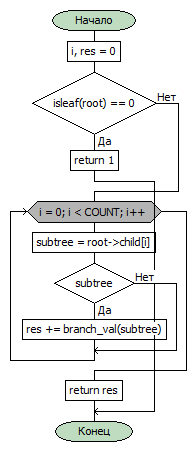
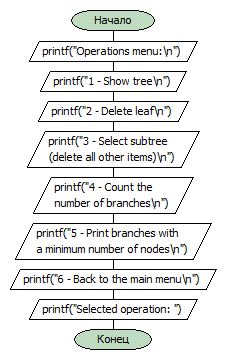
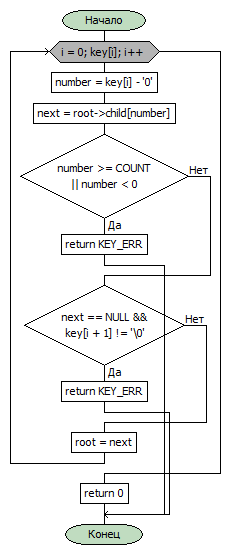
show\_menu\_main

show\_selected\_operations

show\_all\_branches\_min\_nodes

show\_all\_branches\_in\_level

show\_branches\_in\_level

****

check\_key

branch\_val

show\_menu\_operation

menu\_main

**Текст программы**

**Вариант в виде отдельных функций, расположенных в одном файле. Файл main.cpp**

/\*

Name: BDZ3 Trees, variant 29

Author: Vadim Olennikov

\*/

#define COUNT ( 2 ) /\*maximum number of children per node \*/

#define LENGTH ( 100 )/\* max name's length \*/

#define TEMPL\_SIZE ( 20 )/\* characters to read string \*/

#define KEY ( 8 ) /\* max key length \*/

#define NOT\_LEAF\_CODE ( -4 )/\* if node is not leaf \*/

#define KEY\_ERR ( -2 )/\* if key is wrong \*/

#define MEMORY\_ERR ( -1 )

#define FILE\_OPEN\_ERR ( -5 )

#define MAX\_LEVEL ( 100 )/\*max tree level\*/

#define OPERATIONS ( 100 )/\* The number of operations selected by the user \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct \_node node;

struct \_node

{

char name[ LENGTH ];

node \*child[ COUNT ];

};

int add( node \*\*root, char \*key, char \*name);/\* add new element to tree \*/

int load(node \*\* root, char \* filename);/\* load tree from file \*/

node \*create\_leaf( char \*name );/\* create new leaf of tree \*/

void show( node \*root );/\* show tree \*/

void show\_subtree( node \*root, int level );/\* help function for show \*/

void drop( node \*\*cur );/\*delete tree \*/

int load\_exept\_root( FILE \*file, node \*\*root );

int isleaf( node \*leaf );

int delete\_leaf( node \*\*root, char \*key );

int highlight\_subtree( node \*\*root, char \*key );

int check\_key( node \*cur, char \*key );

int branch\_val( node \*root );

int find\_min\_level( node \*root );

void show\_all\_branches\_in\_level( node \*root, int level\_leaf );

void show\_branches\_in\_level( node \*root, node \*cur, int level, int leaf\_level, int \*key );

void show\_branch( node \*root, int leaf\_level, int \*key );

void show\_all\_branches\_min\_nodes( node \*root );

void menu\_main( node \*\*root, int \*operations, int position );

void show\_menu\_main();

void menu\_operations( node \*\*root, int \*operations, int position );

void show\_selected\_operations( int \*operations );

void show\_menu\_operation();

void putNULL( int \*operations );

int main()

{

int res;

int operations[ OPERATIONS ];

node \*root = 0;

char filename[] = "data.txt";

putNULL( operations );

res = load( &root, filename );

if( res == 0 )

menu\_main( &root, operations, 0 );//start position in arr is 0

else

{

printf( "File open error");

exit FILE\_OPEN\_ERR;

}

return 0;

}

void putNULL( int \*operations )

{

int i;

for( i = 0; i < OPERATIONS; i++ )

operations[ i ] = 0;

}

int add( node \*\*cur, char \*key, char \*name)/\* add new element to tree \*/

{

int i, number;

node \*\*next;/\* next level pointer \*/

/\* create new leaf \*/

node \* element;

element = create\_leaf( name );

if( element == NULL )/\* if a memory allocation error occurred \*/

return MEMORY\_ERR;

for( i = 0; key[ i ]; i++ )

{

number = key[ i ] - '0';/\* number of child; convert char to int \*/

next = &( ( \*cur )->child[ number ] );/\* next level pointer \*/

if( number >= COUNT || number < 0 )//if key is wrong

return KEY\_ERR;;

if( \*next == NULL && key[ i + 1 ] != '\0' )/\* if parent of our element doesn't exist \*/

return KEY\_ERR;

cur = next;/\* go to child number key[ i ] \*/

}

( \*cur ) = element;

return 0;/\* successful execution \*/

}

int load (node \*\* root, char \* filename)/\* load tree from file \*/

{

int res\_load;/\* result of scanning subtree\*/

FILE \* file;

char name[ LENGTH ];

char templateSTR[ TEMPL\_SIZE ];/\* to read a given number of characters \*/

sprintf( templateSTR, "%%%is", LENGTH );

file = fopen( filename, "r" );

if( file == NULL )

return -3;/\* file open error \*/

if( fscanf( file, templateSTR, name ) > 0 )/\* put data in root \*/

\*root = create\_leaf( name );

res\_load = load\_exept\_root( file, root );

fclose( file );

return res\_load;/\* result of scanning \*/

}

int load\_exept\_root( FILE \*file, node \*\*root )

{

int res\_add, res\_scan = 1;/\* result of scanning \*/

char key[ KEY ], name[ LENGTH ];

char templateSTR[ TEMPL\_SIZE ];/\* to read a given number of characters \*/

sprintf( templateSTR, "%%%is", LENGTH );

for( ; res\_scan > 0; )

{

res\_scan = fscanf( file, templateSTR, key );

res\_scan = fscanf( file, templateSTR, name );

if( res\_scan > 0 )/\* if name is scanned successful \*/

res\_add = add( root, key, name );/\*root is pointer to pointer and it doesn't change here\*/

if( res\_add != 0 )//not success add

return res\_add;

}

return 0;

}

node \*create\_leaf ( char \*name )/\* create new leaf of tree \*/

{

int i;

node \*element;

element =( node\* ) malloc( sizeof( node ) );

if( element == NULL )/\* if a memory allocation error occurred \*/

return NULL;

strcpy( element->name, name );

for( i = 0; i < COUNT; i++ )

element->child[ i ] = NULL; /\* because element doesn't have children \*/

return element;

}

/\*-------------------------------------------------------------------

Example of result:

-> root

|-> 1

| |-> 4

| | |-> 5

| | |-> 6

| |-> 7

|-> 2

|-> 3

\*/

void show( node \*root )

{

int i;

if( root == NULL )

return;/\* tree is empty \*/

printf( "-> %s\n", root->name );

for( i = 0; i < COUNT; i++ )

show\_subtree( root->child[ i ], 1 );/\* next level is 1 \*/

}

void show\_subtree( node \*root, int level )/\* help function for tree, for formatted output \*/

{

int i;

if( root == NULL )

return;/\* subtree is empty, exit from recursing \*/

for( i = 0; i < level; i++ )/\* print | to display the level \*/

printf( "%4c", '|');

printf( "-> %s\n", root->name );

for( i = 0; i < COUNT; i++ )

show\_subtree( root->child[ i ], level + 1 );/\* next recursion level \*/

}

void drop( node \*\*cur ){

int i;

if( \*cur == NULL )

return;/\* exit from recursion \*/

for( i = 0; i < COUNT; i++ )

drop( &( ( \*cur )->child[ i ] ) );

free( \*cur );

\*cur = 0;

}

int delete\_leaf( node \*\*cur, char \*key )

{

int i, number, res;

node \*\*parent, \*\*child;

for( i = 0; key[ i ]; i++ )//go to element number key

{

parent = cur;

number = key[ i ] - '0';/\* number of child; convert char to int \*/

child = &( ( \*cur )->child[ number ] );/\* next level pointer \*/

if( number >= COUNT || number < 0 )//if key is wrong

return KEY\_ERR;

if( \*child == NULL && key[ i + 1 ] != '\0' )/\* if parent of our element doesn't exist \*/

return KEY\_ERR;

cur = child;

}

res = isleaf( \*child );

if( res )//if key is not for leaf

return res;

free( \*child );//delete leaf

( \*parent )->child[ number ] = 0;

return 0;

}

/\*--------------------------------------------------

Checks if node is leaf (hasn't children)

Returns NOT\_LEAF\_CODE if node is note leaf

Returns 0 if success

leaf - pointer to node

\*/

int isleaf( node \*leaf ){

int i;

for( i = 0; i < COUNT; i++ )

{

if( leaf->child[ i ] != NULL )

return NOT\_LEAF\_CODE;

}

return 0;

}

/\*

Select subtree from tree (delete all other items)

Returns KEY\_ERR if node with code doesn't exist

Returns 0 if success

root - pointer to pointer to tree root

( we have to change pointer to tree )

\*/

int highlight\_subtree( node \*\*root, char \*key )

{

int i, step, number;

node \*parent;

/\*we have no right to make a mistake \*/

if( check\_key( \*root, key ) )

return KEY\_ERR;

for( i = 0; key[ i ]; i++ )

{

number = key[ i ] - '0';/\* number of child we need; convert char to int \*/

for( step = 0; step < COUNT; step++ )

if( step != number )

drop( &( ( \*root )->child[ step ] ) );//delete brothers of element

parent = \*root;

\*root = ( \*root )->child[ number ];//go to next level

free( parent );//delete parent

}

return 0;

}

/\*--------------------------------------------------------------------

Checks the correctness of the key

If the node with the given number does not exist,

returns KEY\_ERR

root - pointer to tree root, main node

key - code of node in tree

\*/

int check\_key( node \*root, char \*key ){

int i, number;

node \*next;

for( i = 0; key[ i ]; i++ )

{

number = key[ i ] - '0';/\* number of child; convert char to int \*/

next = root->child[ number ];/\* next level pointer \*/

if( number >= COUNT || number < 0 )//if key is wrong

return KEY\_ERR;

if( next == NULL && key[ i + 1 ] != '\0')/\* if parent of our element doesn't exist \*/

return KEY\_ERR;

root = next;/\* go to child number key[ i ] \*/

}

return 0;

}

/\*----------------------------------------------------------

Return the number of branches in tree

the number of branches is equal to the number of leaves

root - pointer to tree root

\*/

int branch\_val( node \*root ){

int i, res = 0;

node \*subtree;

/\* the number of branches is equal to the number of leaves \*/

if( isleaf( root ) == 0 )

return 1;//exit from recursion

for( i = 0; i < COUNT; i++ )

{

subtree = root->child[ i ];//pointers to subtrees

if( subtree )//if pointer is not NULL

res += branch\_val( subtree );//call next recursion level

}

return res;

}

/\*---------------------------------------------------------------

Shows all branches with a minimum number of nodes

root - pointer to tree root

\*/

void show\_all\_branches\_min\_nodes( node \*root )

{

int min\_level;

min\_level = find\_min\_level( root );

show\_all\_branches\_in\_level( root, min\_level );

}

/\*----------------------------------------------

Finds the minimum branch length in a tree

root - pointer to tree root

\*/

int find\_min\_level( node \*root ){

node \*subtree;

int min = MAX\_LEVEL;

int i, deep = MAX\_LEVEL;

if( isleaf( root ) == 0 )

return 0;/\* exit from recursion \*/

for( i = 0; i < COUNT; i++ )

{

subtree = root->child[ i ];//pointer to child

if( subtree != NULL )//if child exists

{

deep = find\_min\_level( subtree );/\* subtree depth \*/

if( deep < min )

min = deep;

}

}

return min + 1;

}

/\*-----------------------------------------------------

Displays all branches with leaves at the leaf\_level

root - pointer to tree root

leaf\_level - the level at which the leaves of

the tree should be; branch length

\*/

void show\_all\_branches\_in\_level( node \*root, int leaf\_level ){

int i;

int key[ leaf\_level ];

node \*cur;

if( leaf\_level == 0 )//if tree is only root

{

printf( "Tree is empty, root: %s", root->name );

return;

}

for( i = 0; i < COUNT; i++ )

{

cur = root->child[ i ];

key[ 0 ] = i;//code of first node

show\_branches\_in\_level( root, cur, 1, leaf\_level, key );//because next level is 1

}

}

/\*-------------------------------------------------------------

Recursive function shows all branches with leaf on leaf\_level

root - pointer to tree root

cur - pointer to node

level - level of node with pointer cur

leaf\_level - the level at which the leaves of the tree should be

key - code of node with pointer cur

\*/

void show\_branches\_in\_level( node \*root, node \*cur,

int level, int leaf\_level,

int \*key )

{

int i;

if( level == leaf\_level )

{

if( isleaf( cur ) == 0 )//if elem from leaf\_level is leaf

show\_branch( root, leaf\_level, key );

return;//exit from recursion

}

for( i = 0; i < COUNT; i++ )//next recursion level

{

key[ level ] = i;

if( cur->child[ i ] )//if child exists

show\_branches\_in\_level( root, cur->child[ i ],

level + 1, leaf\_level,

key );

}

}

/\*------------------------------------------------------------------

The function displays the branch on the screen

along with the data in the nodes and their code

root - pointer to tree root

leaf\_level - branch length, leaf level

key - pointer to arr with code of leaf on branch

\*/

void show\_branch( node \*root, int leaf\_level, int \*key ){

int i, step, number;

node \*cur = root;

printf( "%s / ", root->name );//show root name

for( i = 0; i < leaf\_level; i++ )

{

number = key[ i ];

cur = cur->child[ number ];

printf( "(" );

for( step = 0; step <= i; step++ )

printf( "%i", key[ step ] );//show element's code

printf( ") %s", cur->name );//show data from node

if( i != leaf\_level - 1 )//formatted output,

printf( " / " );//not required for last element

}

printf( "\n" );

}

/\*--------------------------------------------------------

Shows transaction information

processes the operation code selected by the user

root - pointer to pointer to tree root( we can change pointer to tree )

operations - pointer to int arr with operations data

position - number of next filling position in operations array

\*/

void menu\_main( node \*\*root, int \*operations, int position )

{

int item;

show\_menu\_main();

scanf( "%i", &item );

printf( "\n" );

operations[ position++ ] = item;

switch( item )

{

case 1 :

show( \*root );

break;

case 2 :/\*Show operation menu\*/

menu\_operations( root, operations, position );

break;

case 3 :/\*Exit the program\*/

drop( root );

show\_selected\_operations( operations );//show list with selected menu items

exit( 0 );

default :

operations[ --position ] = 0;//deselection

printf( "Invalid operation, try again!" );

}

printf( "\n" );

menu\_main( root, operations, position );

}

/\*------------------------------------------------------

Function shows main program menu

\*/

void show\_menu\_main(){

printf( "Menu:\n" );

printf( "1 - Show tree\n" );

printf( "2 - Show operation menu\n" );

printf( "3 - Exit the program\n" );

printf( "Operation: " );

}

/\*--------------------------------------------------------

Shows transaction information

processes the operation selected by the user

root - pointer to pointer to tree root( we can change pointer to tree )

operations - pointer to int arr with operations data

position - number of next filling position in operations array

\*/

void menu\_operations( node \*\*root, int \*operations, int position )

{

int item;/\* selected code \*/

char key[ KEY ];

show\_menu\_operation();

scanf( "%i", &item );

printf( "\n" );

operations[ position++ ] = item;/\*fill operation arr \*/

switch( item ){

case 1 :/\*Show tree \*/

show( \*root );

break;

case 2 :/\*Delete leaf \*/

printf( "Enter key of leaf: " );

scanf( "%s", key );

delete\_leaf( root, key );

break;

case 3 :/\*Select subtree from tree (delete all other items) \*/

printf( "To cancel, enter -1. Enter key of node: " );

scanf( "%s", key );/\* subtree root key \*/

highlight\_subtree( root, key );

break;

case 4 :/\*Count the number of branches \*/

printf( "Number of branches = %i\n", branch\_val( \*root ) );

break;

case 5 :/\* Print branches with a minimum number of nodes \*/

show\_all\_branches\_min\_nodes( \*root );

break;

case 6 :/\* Back to the main menu \*/

if( operations[ position - 1 ] == 2 )/\* delete operation from history \*/

operations[ --position ] = 0;

menu\_main( root, operations, position );

default :

if( operations[ position] )

operations[ --position ] = 0;/\* deselection \*/

printf( "Invalid operation, try again!" );

}

printf( "\n" );

menu\_operations( root, operations, position );

}

/\*----------------------------------------------

Print on display user selected operations

Example: 1 1 2 3 3 5 3

operations - pointer to int arr with operations data

\*/

void show\_selected\_operations( int \*operations ){

int i;

printf("Selected operations: " );

for( i = 0; operations[ i ]; i++ )

printf( "%i ", operations[ i ] );

printf("\n" );

}

/\*------------------------------------------------------

Function shows information about operations in program

\*/

void show\_menu\_operation(){

printf( "Operations menu:\n" );

printf( "1 - Show tree\n" );

printf( "2 - Delete leaf\n" );

printf( "3 - Select subtree (delete all other items)\n" );

printf( "4 - Count the number of branches\n" );

printf( "5 - Print branches with a minimum number of nodes\n" );

printf( "6 - Back to the main menu\n" );

printf( "Selected operation: " );

}

**Данные текстового файла**

**Файл data.txt**

Plan

0 finance

00 players

000 defenders

0000 midfielders

0001 goalkeeper

01 inventory

010 balls

011 uniform

1 management

10 match

100 comand

101 briefing

11 media

110 advertising

111 interview

**Результат работы программы**

*Данные, введенные пользователем, подчеркнуты*

**Демонстрация главного меню, меню операций и вывода дерева:**

Menu:

1 - Show tree

2 - Show operation menu

3 - Exit the program

Operation: 1

-> Plan

|-> finance

| |-> players

| | |-> defenders

| | | |-> midfielders

| | | |-> goalkeeper

| |-> inventory

| | |-> balls

| | |-> uniform

|-> management

| |-> match

| | |-> comand

| | |-> briefing

| |-> media

| | |-> advertising

| | |-> interview

Menu:

1 - Show tree

2 - Show operation menu

3 - Exit the program

Operation: 2

**Демонстрация операции удаления листа (здесь и далее для краткости не показан переход в меню операций)**

Operations menu:

1 - Show tree

2 - Delete leaf

3 - Select subtree (delete all other items)

4 - Count the number of branches

5 - Print branches with a minimum number of nodes

6 - Back to the main menu

Selected operation: 2

Enter key of leaf: 0000

**Изменения дерева, удаленный лист выделен:**

|  |  |
| --- | --- |
| До операции | После |
| -> Plan  |-> finance  | |-> players  | | |-> defenders  | | | |-> midfielders  | | | |-> goalkeeper  | |-> inventory  | | |-> balls  | | |-> uniform  |-> management  | |-> match  | | |-> comand  | | |-> briefing  | |-> media  | | |-> advertising  | | |-> interview | -> Plan  |-> finance  | |-> players  | | |-> defenders  | | | |-> goalkeeper  | |-> inventory  | | |-> balls  | | |-> uniform  |-> management  | |-> match  | | |-> comand  | | |-> briefing  | |-> media  | | |-> advertising  | | |-> interview |

**Демонстрация операции выделения поддерева**

Operations menu:

1 - Show tree

2 - Delete leaf

3 - Select subtree (delete all other items)

4 - Count the number of branches

5 - Print branches with a minimum number of nodes

6 - Back to the main menu

Selected operation: 3

To cancel, enter -1. Enter key of node: 0

**Изменения дерева, выбранный пользователем корень поддерева выделен:**

|  |  |
| --- | --- |
| До операции | После |
| -> Plan  |-> finance  | |-> players  | | |-> defenders  | | | |-> midfielders  | | | |-> goalkeeper  | |-> inventory  | | |-> balls  | | |-> uniform  |-> management  | |-> match  | | |-> comand  | | |-> briefing  | |-> media  | | |-> advertising  | | |-> interview | -> finance  |-> players  | |-> defenders  | | |-> midfielders  | | |-> goalkeeper  |-> inventory  | |-> balls  | |-> uniform |

**Подсчет количества ветвей в исходном дереве:**

Operations menu:

1 - Show tree

2 - Delete leaf

3 - Select subtree (delete all other items)

4 - Count the number of branches

5 - Print branches with a minimum number of nodes

6 - Back to the main menu

Selected operation: 4

Number of branches = 8

**Демонстрация операции вывода ветвей с минимальным числом узлов:**

Operations menu:

1 - Show tree

2 - Delete leaf

3 - Select subtree (delete all other items)

4 - Count the number of branches

5 - Print branches with a minimum number of nodes

6 - Back to the main menu

Selected operation: 5

finance / (1) inventory / (10) balls

finance / (1) inventory / (11) uniform

**Выход из программы:**

Operations menu:

1 - Show tree

2 - Delete leaf

3 - Select subtree (delete all other items)

4 - Count the number of branches

5 - Print branches with a minimum number of nodes

6 - Back to the main menu

Selected operation: 6

Menu:

1 - Show tree

2 - Show operation menu

3 - Exit the program

Operation: 3

Selected operations: 1 2 2 1 3 1 4 5 6 3

Process returned 0 (0x0) execution time : 64.618 s

Press any key to continue.

**Заключение**

Цель работы - научиться составлять отчеты. В результате работы создан отчет о проделанной работе на 22 страницы. Недостатком программы является наличие синтаксиса языка C в блок-схемах.