МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 8

по дисциплине: Алгоритмы и структуры данных
тема: Структуры данных типа «таблица»»

Выполнил: ст. группы ПВ-202

Аладиб язан

Проверил:

Кабалянц Петр Степанович

Маньшин Илья Михайлович

Лабораторная работа № 8
«Структуры данных типа «таблица»»

Цель работы: изучить формальные теории. Разработать программу, реализующую метод резолюций для логики высказываний.

Задания

1. Для СД типа «таблица» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

1.1.2. Набор допустимых операций.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

1.2.2. Объем памяти, занимаемый экземпляром СД.

1.2.3. Формат внутреннего представления СД и способ его и интерпретации

1.2.4. Характеристику допустимых значений.

1.2.5. Тип доступа к элементам.

1.3. Логический уровень представления СД. Способ описания СД и экземпляра СД на языке программирования.

2. Реализовать СД типа «таблица» в соответствии с вариантом индивидуального (табл.18) задания в виде модуля.

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.18) с использованием модуля, полученного в результате выполнения пункта 2 задания

Выполнение заданий

1. Для СД типа «таблица» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости: множество, динамическая структура

1.1.2. Набор допустимых операций: Инициализация, включение элемента, исключение элемента с заданным ключом, чтение элемента с заданным ключом, изменение элемента с заданным ключом, проверка пустоты, уничтожение.

1.2. Физический уровень представления СД:

1.2.1. Схема хранения: последовательная или связная

1.2.2. Объем памяти, занимаемый экземпляром СД: зависит от базового типа элемента таблицы

1.2.3. Формат внутреннего представления СД и способ его и интерпретации: в динамической памяти (каждый элемент таблицы представляет из себя СД типа «запись» из ключа и информативной части); на массиве

1.2.4. Характеристику допустимых значений: CAR(Table)=

CAR(BaseType)0 + CAR(BaseType)1 + … + CAR(BaseType)max

1.2.5. Тип доступа к элементам: прямой или последовательный

1.3. Логический уровень представления СД. Способ описания СД и экземпляра СД на языке программирования: на дереве

```
typedef struct element {
    KeyAndValue data;
    ptrEl leftSon;
    ptrEl rightSon;
} element;
```

2. Реализовать СД типа «таблица» в соответствии с вариантом индивидуального (табл.18) задания в виде модуля.

Содержимое заголовочного файла:

```
#ifndef CODE_TABLE_H
#define CODE_TABLE_H

#include <stdbool.h>
#include "../Tree/Tree.h" // Смотреть лаб.раб. №7


static const short TableOk = 0;
static const short TableNotSet = 1;
static const short TableNotExistElement = 2;
static const short TableEnd = 3;
short TableError;

typedef char* T_Key;
typedef Tree Table;

// инициализация таблицы
void initTable(Table *T);
// возвращает "истина", если таблица пуста, иначе "ложь"
bool emptyTable(Table *T);
// включает элемент таблицу, возвращает "истина", если элемент включён
// в таблицу, иначе "ложь"
bool putTable(Table *T, BaseType Key);
// исключает элемент, возвращает "истина", если элемент с ключом s был
// в таблице, иначе "ложь"
bool getTable(Table *T, T_Key Key);
// читает элемент. Возвращает "истина", если элемент с ключом s есть в
// таблице, иначе "ложь"
bool readTable(Table T, BaseType *E, T_Key Key);
// изменение элемента. Возвращает "истина", если элемент с ключом s
// есть в таблице, иначе "ложь"
bool writeTable(Table *T, BaseType E, T_Key Key);
// уничтожение таблицы
void doneTable(Table *T);

#endif //CODE_TABLE_H
```

Содержимое файла реализации:

```
#include "Table.h"
```

```
#include <cstdio>
#include <cstdlib>

void initTable(Table *T) {
  initTree(T);
}

// возвращает "истина", если таблица пуста, иначе "ложь"
bool emptyTable(Table *T) {
  return T == NULL;
}

// запись данных
void _putTable(Table *T, BaseType E) {
  // проверка, есть ли элементы
  if (*T == NULL) {
    *T = createRoot(E);
    TableError = TreeError;
  } else { // если были элементы, нужно переназначить указатели
    if (strcmp_((*T)->data.key, E->data.key) > 0) {
      _putTable(&(*T)->leftSon, E);
    } else if (strcmp_((*T)->data.key, E->data.key) < 0) {
      _putTable(&(*T)->rightSon, E);
    } else { // если находится такой же элемент, передаётся соответствующее значение переменной
ошибки
      TableError = TableNotSet;
    }
  }
}

// включает элемент таблицу, возвращает "истина", если элемент включён в таблицу, иначе "ложь"
bool putTable(Table *T, BaseType E) {
  if (emptyTable(*T)) { // если таблица пуста, сначала нужно создат корень дерева
    *T = createRoot(E);
    TableError = TreeError;
  } else { // иначе пользуемся функцией для записи в дерево
    _putTable(T, E); // внутри этой функции проверяется на универсальность
  }
  return TableError == TableOk;
}

// удаляет из дерева элемент с ключом Key
void _getTable(Tree *T, T_Key Key) {
  if (*T == NULL) {
    TreeError = TreeEnd;
  }
  if (strcmp_(Key, (*T)->data.key) < 0) { // если удаляемый элемент меньше текущего
    _getTable(&(*T)->leftSon, Key);
  } else if (strcmp_(Key, (*T)->data.key) > 0) { // если удаляемыq элемент больше текущего
    _getTable(&(*T)->rightSon, Key);
  } else if (strcmp_(Key, (*T)->data.key) == 0) { // если удаляемый элемент - текущий
    if ((*T)->leftSon == NULL) { // если левого ребёнка нет
      Tree tmp = *T;
      *T = (*T)->rightSon;
      free(tmp);
```

```
   } else if ((*T)->rightSon == NULL) { // если правого ребёнка нет
     Tree tmp = *T;
     *T = (*T)->leftSon;
     free(tmp);
   } else { // если есть оба ребёнка
     Tree tmp = (*T)->rightSon;
     Tree prev = (*T);
     while (tmp->leftSon) {
       prev = tmp;
       tmp = tmp->leftSon;
     }
     (*T)->data = tmp->data;
     if (prev != *T) {
       prev->leftSon = NULL;
     } else {
       prev->rightSon = tmp->rightSon;
     }
     TableError = TableOk;
     free(tmp);
   }
  } else { // если такого элемента нет вообще
    TableError = TableNotExistElement;
  }
}

// исключает элемент, возвращает "истина", если элемент с ключом s был в таблице, иначе "ложь"
bool getTable(Table *T, T_Key Key) {
  _getTable(T, Key);
  return TableError == TableOk;
}

// чтение данных элемента с ключом s
Tree _readDataTable(Tree T, T_Key Key) {
  if (T == NULL) {
    TableError = TableNotExistElement;
    return T;
  } else if (strcmp_(Key, (*T).data.key) == 0) {
    TableError = TableOk;
    return T;
  }
  if (strcmp_(Key, (*T).data.key) < 0) {
    // если элемент меньше текущего, уходим в левое поддерево
    return _readDataTable(T->leftSon, Key);
  } else if (strcmp_(Key, (*T).data.key) > 0) { // иначе уходим в правое поддерево
    return _readDataTable(T->rightSon, Key);
  } else { // вообще нет такого элемента
    TableError = TableNotExistElement;
    return NULL;
  }
}

// читает элемент. Возвращает "истина", если элемент с ключом s есть в таблице, иначе "ложь"
bool readTable(Table T, BaseType *E, T_Key Key) {
  *E = _readDataTable(T, Key);
  return TableError == TableOk;
```

```
}

// изменение элемента. Возвращает "истина", если элемент с ключом s есть в таблице, иначе "ложь"
bool writeTable(Table *T, BaseType E, T_Key Key) {
 BaseType tmp;
 if (readTable(T, &tmp, Key)) { // если элемент есть
  tmp = E; // перезаписываем в найденный элемент нужный
  TableError = TableOk;
 } else {
  TableError = TableNotExistElement;
 }
 return TableError == TableOk;
}

// уничтожение таблицы
void doneTable(Table *T) {
 delTree(*T);
}
```

№3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.18) с использованием модуля, полученного в результате выполнения пункта 2 задания

Текстовый файл содержит текст на русском языке. В тексте могут встречаться числа, записанные в словесной форме. Преобразовать файл, заменив словесную запись чисел числовой.

Например, файл:
Получил триста двадцать пять рублей пятнадцать копеек.
преобразовать в файл:
Получил 325 рублей 15 копеек.
Для преобразования чисел использовать таблицу. Ключ элемента — словесное название числа («один», «два»,..., «десять», «одиннадцать»,..., «двадцать»,..., «девяносто», «сто», «двести», «триста»,..., «девятьсот»), информационная часть — числовое значение ключа. Информацию в таблицу загрузить из текстового файла.

Содержимое файла реализации:

```
#include <stdio.h>
#include <windows.h>
#include "Table/Table.h"

int main() {
 SetConsoleOutputCP(CP_UTF8);
 // сохранение элементов для таблицы из файла
 element arrayOfElements[256];
 unsigned amountElementsForTable = 0;
 char filename[] = "Keys_and_values_for_Table.txt";
 FILE *f = fopen(filename, "r"); // открытие файла
 // запись из файла в массив элементов
 while (fscanf(f, "%s %u",
        arrayOfElements[amountElementsForTable].data.key,
```

```
            &arrayOfElements[amountElementsForTable].data.s) != EOF) {
  amountElementsForTable++;
 }
 fclose(f);
 // инициализация таблицы
 Table table;
 initTable(&table);
 // заполнение таблицы элементами
 for (int i = 0; i < amountElementsForTable; i++) {
  putTable(&table, &arrayOfElements[i]);
 }
 // проходим по тексту для задания
 char exerciseFile[] = "Text_for_exercise_ASD.txt";
 FILE *exercise = fopen(exerciseFile, "r");
 char s[256];
 BaseType convertNumbersArray[30];
 unsigned i = 0;
 unsigned counter = 0;
 // проходимся до конца файла
 while (!feof(exercise)) {
  // считываем по одному слову
  fscanf(exercise, "%s", s);
  // если нашли число, берём его из таблицы по ключу
  if (readTable(table, &convertNumbersArray[i], s)) {
   // необходимо сложить найденные числа в одно число
   counter += convertNumbersArray[i]->data.s; // складываем и запоминаем
   i++;
  } else { // если слово не число
   if (counter != 0) { // если что-то насчитали
    printf("%d ", counter); // выводим сумму найденных чисел в цифровом эквиваленте
    counter = 0;
   }
   printf("%s ", s); // выводим слово
  }
 }
 return 0;
}
```