МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах

Кафедра Информационных систем

№	Дата поступления на кафедру	Подпись отв. за регистрацию	Подпись преподавателя

ОТЧЕТ

по учебной (ознакомительной) практике

в _ ФГАОУ ВО “Севастопольский государственный университет” _

Выполнил Хроменко Д.А._____
(Фамилия И.О. обучающегося)
ИС/б-20-2-о_____
(шифр группы)
Направление / специальность __09.03.02
Информационные системы и технологии
(код, наименование)

Руководитель практики от Университета
_____
(должность)
_____
(Фамилия И.О. руководителя)

Севастополь
2021 г.

1 Программирование операций над строками и файлами

1.1 Описание задания

Написать программу, которая считывает строку из стандартного потока ввода, определяет количество знаков препинания каждого типа (. : ; ! ? ,) и выводит результат. В формате "знак = количество" в отдельных строках. Порядок знаков в выводе должен совпадать с порядком указанном в задании.

1.2 Код программы

```
#include <stdio.h>
int main()
{
    int points = 0;
    int colons = 0;
    int semicolons = 0;
    int exclamation_marks = 0;
    int question_marks = 0;
    int commas = 0;
    char str[999999];
    fgets(str, 999998, stdin);

    unsigned long i;
    for(i=0; str[i] != '\0'; i++)
    {
        switch (str[i])
        {
            case '.' : ++points; break;
            case ':' : ++colons; break;
            case ';' : ++semicolons; break;
            case '!' : ++exclamation_marks; break;
            case '?' : ++question_marks; break;
            case ',' : ++commas; break;
        }
    }

    printf(". = %d\n", points);
    printf(": = %d\n", colons);
    printf("; = %d\n", semicolons);
    printf("! = %d\n", exclamation_marks);
    printf("? = %d\n", question_marks);
```

```
    printf(", = %d", commas);

    return 0;
}
```

1.3 Описание решения

Для программы была подключена библиотека <stdio.h> для поддержки ввода и вывода. В функции main были объявлены переменные счётчики для всех знаков препинания. Далее с помощью функции fgets в строку str считываются с клавиатуры символы. Затем с помощью цикла for каждый символ строки сравнивается со знаками препинания, если какой либо символ совпадает с каким либо знаком препинания, то счётчик этого знака увеличивается на один. В конце программы выводятся знаки препинания и их количество в строке.

2 Программирование операций со структурами

2.1 Описание задания

Дана структура timeStruct (время), состоит из трёх полей целого типа (часы, минуты и секунды). Необходимо написать функцию прибавляющую одну секунду к заданному времени.

2.2 Код программы

```
//typedef struct timeStruct{
//    int h;
//    int m;
//    int s;
//} timeStruct;

timeStruct inc(timeStruct a)
{
    if (a.s != 59)
    {
        a.s += 1;
        return a;
    }

    a.s = 0;
    if (a.m != 59)
```

```
    {
        a.m += 1;
        return a;
    }

    a.m = 0;
    a.h += 1;
    return a;
}
```

2.3 Описание решения

Время состоит из секунд, минут и часов. Секунды и минуты не могут быть больше 59, следовательно если к 59 секундам добавить ещё одну, то секунд становится 0, а к минутам прибавляется единица. Если же минут 59 то аналогично с секундами минуты обнуляются и к часам прибавляется единица. Однако если секунд не 59, то просто прибавляется одна секунда. В конце функции возвращаются данные структуры time.

3 Программирование операций над линейным списком

3.1 Описание задания

Вам дана структура student (студент), содержащая поля: номер, фамилия, оценки (структура с полями типа float физика, история и математика) и структура списка.

Ваша задача – реализовать функции:

init – инициализация пустого списка;

get – получение студента из списка по индексу;

push – добавление студента в конец списка;

marshal – конвертирует переданную структуру в строку через пробел:

"№ Фамилия Оценка.Физика Оценка.История Оценка.Математика";

unmarshal – конвертирует строку из формата выше в структуру студент;

findById – находит студента по номеру.

3.2 Код программы

```
/*
```

```
typedef struct studentMarks {
    float physics;
    float history;
    float maths;
} studentMarks;

typedef struct student {
    int id;
    string lastname;
    studentMarks marks;
} student;

typedef struct node {
    student el;
    node* next;
} node;

typedef struct list {
    node* head;
    int size;
} list;

list* init();
node* get(list* l, int index);
void push(list* l, student el);
string marshal(student el);
student unmarshal(string s);
node* findById(list* l, int id);
*/

list* init()
{
    list *l = (list*)malloc(sizeof(list));
    l -> head = NULL;
    l -> size = 0;
    return (l);
}

node* get(list* l, int index)
{
    int i;
```

```
    node *tmp = l -> head;
    for (i = 0; i != index; i++)
    {
        if (!tmp) break;
        tmp = tmp -> next;
    }

    return tmp;
}

void push(list* l, student el)
{
    node **tail = &(l->head);

    while (*tail)
        tail = &((*tail) -> next);

    (*tail) = (node*)malloc(sizeof(node));

    (*tail) -> el = el;
    (*tail) -> next = NULL;

    l -> size++;

    return;
}

string marshal(student el)
{
    string str = to_string(el.id) + " " + el.lastname + " " +
to_string((int)el.marks.physics) + " " + to_string((int)el.marks.history) + " " +
to_string((int)el.marks.maths);

    return str;
}

student unmarshal(string s)
{
    student stud;
    char buf[30];
```

```
    sscanf(s.c_str(), "%d %s %f %f %f", &(stud.id), buf, &(stud.marks.physics),
&(stud.marks.history), &(stud.marks.maths));
    stud.lastname = buf;

    return stud;
}

node* findById(list* l, int id)
{
    node *tmp = l -> head;

    while (tmp != NULL)
    {
        if (tmp->el.id == id) break;
        tmp = tmp->next;
    }

    return tmp;
}
```

3.3 Описание решения

1). В первой функции init происходит инициализация пустого списка с помощью выделения памяти для одного элемента, затем указатель на голову равен NULL, размер списка равен нулю. Функция возвращает список.

2). Функция get ищет студента в списке по индексу. Индекс передаётся в функцию. В начале объявляется переменная tmp и счётчик итераций. Далее с помощью цикла for каждая итерация сравнивается с индексом. Если индекс и количество итераций совпадают или заканчивается список происходит выход из цилка. На каждой итерации цикла происходит перемещение по списку. В конце функция возвращает указатель на тот элемент списка который совпадает с индексом.

3). Функция push добавляет в конец списка элемент. Сперва создаётся переменная «хвост» которая с помощью цикла while перемещается в конец списка. Затем происходит выделение памяти, в элемент списка вставляются данные и указатель на следующий элемент равен NULL. Размер списка увеличивается на единицу.

4). Функция marshal конвертирует элемент структуры в строку. Для этого создаётся строка в которую сразу записываются элементы структуры с помощью функции to_string. Функция возвращает строку.

5). Функция unmarshal конвертирует строку в элемент списка. Для этого используется функция sscanf библиотеки <stdio.h>. Функция возвращает запись «студент».

6). Функция findByld находит студента по номеру. Работает аналогично функции get, но выходит из цикла при условии совпадении номеров студентов или окончании списка. Возвращает также указатель на тот элемент списка который совпадает с номером.

4 Программирование операций над бинарными деревьями

4.1 Описание задания

Вам дана структура student (студент), содержащая поля: номер, фамилия, оценки (структура с полями типа float физика, история и математика) и структура для бинарного дерева со следующими принципами работы:

Ключевым полем является – фамилия.

Порядок обхода дерева при выводе – left, center, right.

Элемент с меньшим ключом должен располагаться в левой ветви (для сравнения строк используйте strA.compare(strB)).

Гарантируется, что в тестах все фамилии разные.

Ваша задача – реализовать функции:

get – получение студента из дерева по ключу;

push – добавление студента в дерево (возвращает голову получившегося дерева);

printContent – конвертирует переданное дерево в строку, каждая строка которой содержит данные одного студента через пробелы:

"№ Фамилия Оценка.Физика Оценка.История Оценка.Математика";

printStruct – конвертирует переданное дерево в строку, демонстрирующую структуру дерева с помощью отступов (символ отступа – табуляция "\t");

getBestStudent – определение студента с наибольшим средним баллом (если есть студенты с равным количеством баллов – выведите первого из них в порядке обхода).

4.2 Код программы

```
/*
typedef struct studentMarks {
    float physics;
    float history;
    float maths;
} studentMarks;

typedef struct student {
    int id;
    string lastname;
    studentMarks marks;
} student;

typedef struct node {
    student el;
    node* left;
    node* right;
} node;

node* get(node* head, string lastname);
node* push(node* head, student el);
string printContent(node* head);
string printStruct(node* head, string prefix);
node* getBestStudent(node* head);
*/

node* get(node* head, string lastname)
{
    if (!head) return NULL;

    if (head -> el.lastname == lastname)
        return head;

    if  (head -> el.lastname > lastname)
        return get(head -> left, lastname);
```

```
    else
        return get(head -> right, lastname);
}

node* push(node* head, student el)
{
    if (!head)
    {
        head = (node*)malloc(sizeof(node));
        head -> left = NULL;
        head -> right = NULL;
        head -> el = el;
    }

    if (head -> el.lastname > el.lastname)
    {
        if (head -> left)
            return  push(head -> left, el);
        else
        {
            head -> left = (node*)malloc(sizeof(node));
            head -> left -> left = NULL;
            head -> left -> right = NULL;
            head -> left -> el = el;
        }
    }

    if (head -> el.lastname < el.lastname)
    {
        if (head -> right)
            return push(head -> right, el);
        else
        {
            head -> right = (node*)malloc(sizeof(node));
            head -> right -> left = NULL;
            head -> right -> right = NULL;
            head -> right -> el = el;
        }
    }
    return head;
}
```

```
string printContent(node* head)
{
    if (!head) return "";

    return printContent(head -> left) +
           to_string(head -> el.id) + " " + (head -> el.lastname) + " " +
to_string((int)head -> el.marks.physics) + " " + to_string((int)head ->
el.marks.history) + " " + to_string((int)head -> el.marks.maths) + "\n" +
           printContent(head -> right);
}

string printStruct(node* head, string prefix)
{
    if (!head) return "";

    return printStruct(head -> left, prefix + "\t") +
           prefix +
           to_string(head -> el.id) + " " + (head -> el.lastname) + " " +
to_string((int)head -> el.marks.physics) + " " + to_string((int)head ->
el.marks.history) + " " + to_string((int)head -> el.marks.maths) + "\n" +
           printStruct(head -> right, prefix + "\t");
}

node* getBestStudent(node* head)
{
    if(!head) return NULL;

    node *tmp = getBestStudent(head -> left);

    if(!tmp)
        tmp = head;

    else if( ((head -> el.marks.history + head -> el.marks.physics + head ->
el.marks.maths) / 3) > ((tmp -> el.marks.history + tmp -> el.marks.physics + tmp ->
el.marks.maths) / 3) )
        tmp = head;


    node *tmpRight = getBestStudent(head -> right);
```

```
    if(!tmpRight)
       return tmp;

    else if( ((tmpRight -> el.marks.history + tmpRight -> el.marks.physics + tmpRight
-> el.marks.maths) / 3) > ((tmp -> el.marks.history + tmp -> el.marks.physics + tmp
-> el.marks.maths) / 3) )
        tmp = tmpRight;


    return tmp;
}
```

4.3 Описание решения

1). Функция get получает студента из дерева по ключу. Сперва идёт проверка существует ли head, если нет то возвращается NULL, также если фамилии совпадают возвращается указатель на данный элемент дерева. Если же фамилия не найдена то идёт сравнение искомой фамилии и фамилии в head. Если head больше, то идём в левую ветвь дерева, иначе в правую.

2). Функция push добавляет элемент в дерево. Если это первый элемент (головы нет), то выделяется память под один элемент, указатель на левое и правое поддерево равен NULL, содержимое узла равно el. Если же это не первый элемент, то происходит сравнение фамилий. Если в голове фамилия больше то идём в левое поддерево. Если там уже есть элемент то вызываем функцию push с головой указывающей на левый элемент от текущего. Если же в левом поддереве нет элемента то мы выделяем память, указатели на левое и правое поддерево делаем равными NULL и вставляем el в содержимое узла. Аналогично происходит с правым поддеревом если фамилия в голове меньше. В конце возвращается указатель на голову.

3). Функция printContent конвертирует дерево в строку. Так как обход дерева происходит в порядке слева, по центру, справа, то сперва функция проводит обход слева с сопутствующим переводом элементов узла в строку с помощью функции to_string, а затем по правым элементам дерева. После всего обхода дерева функция возвращает строку.

4). Функция printStruct работает аналогично printContent, но при этом имеет дополнительный параметр prefix, отвечающий за уровень глубины элемента дерева.

5). Функция getBestStudent ищет студента с наибольшим средним баллом. Для этого с помощью рекурсии мы обходим дерево и сравниваем узел с левой дочерней вершиной, а затем с правой. В конце функция возвращает указатель на студента с наибольшим средним баллом

Выводы

При выполнении задания №1 были закреплены навыки работы со строками, как с массивами символов и файлами в языке С. Также закреплены знания для работы с вводом и выводом из стандартного потока. Изучена функция «fgets» для работы со стандартным потоком ввода. Также были повторно закреплены навыки работы с условными операторами. Полученные навыки помогут создавать более сложные программы для последующих заданий.

При выполнении задания №2 получены и применены на практике знания связанные со структурами в языке С. Полученные навыки помогут при создании программ, работающих с динамическими структурами данных.

При выполнении задания №3 были изучены основные принципы работы с указателями в языках С и С++. Также были повторно закреплены навыки работы с выделением памяти. Были изучены динамические структуры данных, а именно линейный список способы его представления на языке С и С++. Полученные навыки помогут создавать программы, работающие с бинарными деревьями.

При выполнении задания №4 были изучены и применены в решении задачи знания о бинарных деревьях и операциях над ними. Повторно закреплены навыки работы с рекурсивными функциями.

Все полученные во время разработки знания и опыт помогут разрабатывать более сложные программы в будущем.