МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени Н.Э. БАУМАНА

Учебное пособие

Рабочая тетрадь для выполнения лабораторных работ по курсу :

«Основы конструкторско-технологической информатики»

МГТУ имени Н.Э. Баумана

Отчет по лабораторной работе №1 «Основы C++, работа в среде Microsoft Visual Studio»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение принципов работы в среде Microsoft Visual Studio, ознакомление с базовыми функциями C++.

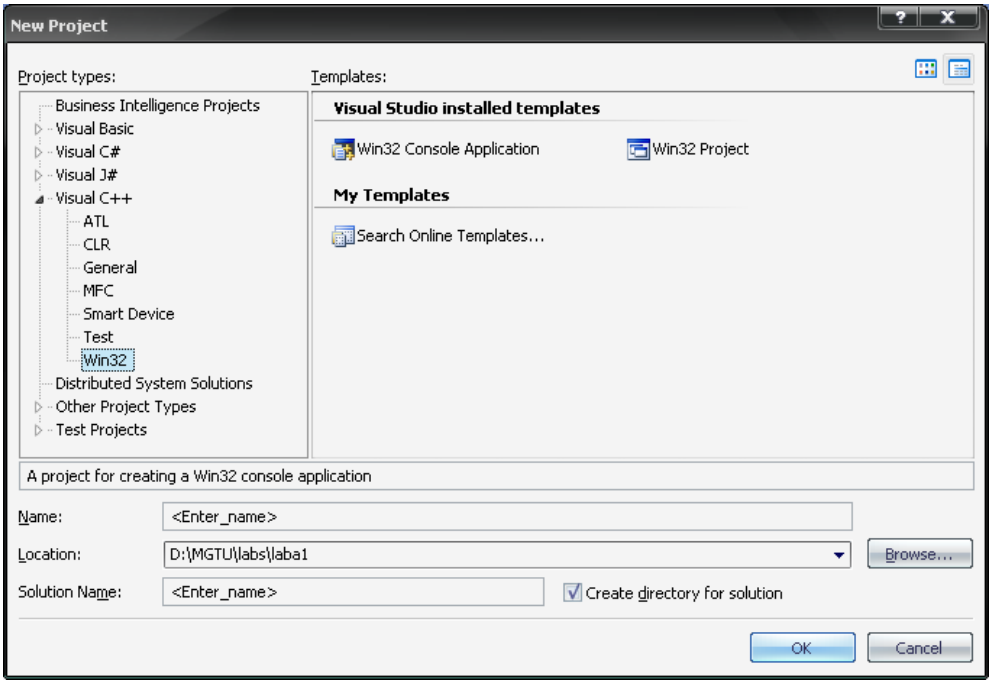
Задачи работы:
-знакомство с работой Microsoft Visual Studio и основами C++
-разработка примеров простейших программ на C++

Краткий конспект теоретической части (ответы на контрольные вопросы)

Понятие проекта и решения? _____
_____
_____
_____
_____
Структура программы? _____
_____
_____
_____
_____
_____
Типы данных в C++?_____
_____
_____
_____
_____
Управляющие конструкции и циклы?_____
_____
_____
_____
_____
Функции ввода/вывода в C++_____
_____
_____
_____
_____
_____
Подключение библиотек_____ _____
_____
_____
_____

Примеры работы в среде Microsoft Visual Studio

Пример 1: Создание проекта в среде Microsoft Visual Studio . Net:



Результат создания проекта

Пример 2: Пример простейшей программы «Hello, world»:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
printf("Hello, world! “);
return 0;
}
```

Результат выполнения

Пример 3: Основные типы данных:

```
int i = 125;
long l = 324;
short s = 65535;
```

```
unsigned short u = 65535;
char ch = ’c’;
double d1 = 3.14159;
float f1, f2 = 5.77e+3;
double d2 = 25.4e+2;
char str[5];
str = ”Tanya”;

prinf(“int = %d long = %d \n”, i, l);
printf("short = %d \n unsigned short = %d ", s, u);
printf(“char = %c \n”, c);
printf(“float f2 = %7.3f \n”,f2);
printf(“d1 = %3.5f \t d2 = %4.2f \n”, d1, d2);
printf(“str = %s”, str);
```

Результат выполнения

Пример 4: Арифметические операции:

```
int x;
x = 100;
printf(“before x = %d \n”, x);
x = -(5 % 3);
y = (-5) % 3;
printf(“after x  =  %d y = %3.2f  \n”, x, y );
x = 5;
y = x;
int a, b;
a = (++x) + 2; // переменной a присваивается значение 8
b = (y++) + 2; // переменной b присваивается значение 7
```

Результат выполнения

Пример 5: Логические операции

```
bool a, b, c, d;
int x, y;
x = 3;
y = 7;
b = true;
c = false;
a = b || c;            // логическое "или"
d = b && c;            // логическое "и"
a = !b;                // логическое "не"
a = (x == y);          // сравнение в правой части
c = (x > 0 && y != 1); // с истинно, когда
                          // оба сравнения истинны
if (x == y) {printf(“x equals y! \n”);} // true, если x равно y,
иначе false
if (x != y) {printf(“x does not equal y! \n”);}
if (x < y) {printf(“x < y \n”);}  // всегда false
```

Результат выполнения

Задание 1:

1. Напишите программу, которая переводит секунды в дни, часы, минуты и секунды.
Количество переводимых секунд запрашивать у пользователя.

```
Результат выводить в виде:
N секунд это:
дней:   d
часов:  h
минут:  m
секунд: s
```

2. Напишите программу, переводящую скорость из километров в час в метры в секундах.
Количество переводимых км/ч запрашивать у пользователя.
Результат выводить в виде: `N км/ч = M м/с`

Исходный код
Результат выполнения

Проверять корректность введенных данных.
Для ввода чисел используйте оператор scanf(“%format”,a), где
format: %d – для переменных типа int, %3.2f – для переменных типа double, float (первое число – количество символов целой части, второе число – количество десятичных разрядов), %c – для переменных типа char, %s – для строковых переменных (массивов из элементов типа char).

Пример 6: Условный оператор

```
int x, y;
x = 4;
y = 9;
if (x > y)
{
    double tmp = x;
    x = y;
    y = tmp;
} else {
         x = -1;
         y = 0;
        }
prinf (“x = &d, y = %d \n”, x, y);

double a, s;
a = 2.0;
if (a < 0.0) {
    s = (-1.0);
}
else if (a > 0.0) {
    s = 1.0;
}
else {
    s = 0.0;
}
prinf (“s = %3.2f \n”, s);
```

Результат выполнения

Пример 7: Операторы цикла while, for

```
int n, p;
p = 1;
n = 65;
while (2*p <= n)
    p *= 2;
printf(“p = %d”,p);

double a[5]; // Массив a содержит не более 5 эл-тов
double sum;    // Переменная для суммы эл-тов массива
int i;         // Переменная цикла

for(i = 0; I < 5; i++)
a[i] = i + 1;

sum = 0.0;
for (i = 0; i < 5; ++i) {
```

```
    sum += a[i]; // Увеличиваем сумму на a[i]
}
printf("Summa = %3.2f  \n", sum);
```

Результат выполнения

Задание 2:

1. Напишите функцию, реверсирующую строку (переставляющую символы в строке в обратном порядке).
2. Вычислить y=x1+x1x2+x1x2x3+:+ x1x2 *** xm,
где m - либо номер первого отрицательного элемента массива x,
либо общее количество элементов n в массиве x.
3. Напишите программу, печатающую таблицу умножения для чисел от 0 до 9 в десятичной системе счисления.

Исходный код
Результат выполнения

Контрольные вопросы

1. Понятие проекта и решения?
2. Структура программы?
3. Типы данных в C++?
4. Управляющие конструкции и циклы?
5. Функции ввода/вывода в C++?
6. Подключение библиотек?

СПИСОК ЛИТЕРАТУРЫ

1. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
2. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
3. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
4. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.
5. Джонсон Б., Скибо К., Янг М.: Основы Microsoft Visual Studio .NET 2003: - М.: Изд-вр Русская редакция, 2003 г. – 464 с.: ил.

Отчет по лабораторной работе №2 «Основы С++, представление программы в виде функций»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение принципов работы в среде Microsoft Visual Studio, ознакомление с базовыми функциями С++ структурным представлением программ.

Задачи работы:
-знакомство с понятием функции, параметров
-разработка примеров простейших программ

Краткий конспект теоретической части (ответы на контрольные вопросы)

Представление программы в виде функций?_____
_____
_____
_____
_____
Прототип функции?_____
_____
_____
_____
Формальные и фактические параметры?_____
_____
_____
_____
_____
_____
Способ передачи параметров: по ссылке и по значению?_____
_____
_____
_____
_____
Вызов функции?_____ _____
_____
_____
_____
Вычисление квадратного корня?_____
_____
_____
_____
_____

Представление программ в виде функций.

Пример 1: Объявление и вызов функции

```
#include <stdio.h> //Описания стандартного ввода-вывода

void print_num(i,j){
int i,j;
   printf("значение i=%d. Значение j=%d.", i,j);
}

int main()
{
print_num(6,19);
return 0;
}
```

Результат выполнения

Пример2: Аргументы функции: формальные и фактические параметры

Программа вводит с клавиатуры терминала два целых числа, затем вычисляет и печатает их наибольший общий делитель. Непосредственно вычисление наибольшего общего делителя реализовано в виде отдельной функции

```
int gcd(int x, int y);

#include <stdio.h> // Описания стандартного ввода-вывода

int gcd(int x, int y); // Описание прототипа функции

int main() {
    int x, y, d;
    printf("Введите два числа:\n");
    scanf("%d%d", &x, &y);
    d = gcd(x, y);
    printf("НОД = %d\n", d);
    return 0;
}

int gcd(int x, int y) { // Реализация функции gcd
    while (y != 0) {
        // Инвариант: НОД(x, y) не меняется
        int r = x % y;  // Заменяем пару (x, y) на
        x = y;          // пару (y, r), где r --
        y = r;          // остаток от деления x на y
    }
```

```
    // Утверждение: y == 0
    return x;   // НОД(x, 0) = x
}
```

Результат выполнения

Пример 3: Вычисление квадратного корня

Пусть надо найти квадратный корень из неотрицательного вещественного числа a с заданной точностью ε. Задача сводится к нахождению корня функции $y = x^2-a$ на отрезке [0,b], где b = max(1,a). На этом отрезке функция имеет ровно один корень, поскольку она монотонно возрастает и на концах отрезка принимает значения разных знаков (или нулевое значение при a = 0 или a = 1).

```
#include <stdio.h> // Описания стандартного ввода-вывода

double a;   // Число, из которого извлекается корень

double f(double x) //функция x^2-a
{
double res;
res = x*x-a;

return res;
}


int main() {
    double x, x0, x1;      // [x0, x1] - текущий отрезок
    double y;              // Значение ф-ции в точке x
    double eps = 0.000001; // Точность вычисления корня

    printf("Введите число a:\n");
    scanf("%lf", &a);

    if (a < 0.0) {
        printf("Число должно быть неотрицательным.\n");
        return 1; // Возвращаем код
    }             //     некорректного завершения

    // Задаем концы отрезка
    x0 = 0.0;
    x1 = a;
    if (a < 1.0) {
        x1 = 1.0;
    }
```

```
    // Утверждение: x0 * x0 - a <= 0,
    //              x1 * x1 - a >= 0

    while (x1 - x0 > eps) {
        // Инвариант: x0 * x0 - a <= 0,
        //            x1 * x1 - a >= 0
        x = (x0 + x1) / 2.0; // середина отрезка [x0,x1]
        y = f(x);       // значение ф-ции в точке x

        if (y >= 0.0) {
            x1 = x; // выбираем левую половину отрезка
        }
        else {
            x0 = x; // выбираем правую половину отрезка
        }
    }

    // Утверждение: x0 * x0 - a <= 0,
    //              x1 * x1 - a >= 0,
    //              x1 - x0 <= eps
    x = (x0 + x1) / 2.0; // Корень := середина отрезка

    // Печатаем ответ
    printf("Квадратный корень = %lf\n", x);

    return 0; // Возвращаем код успешного завершения
}
```

Результат выполнения

Задание 3:

1. Дана строка символов S. Напечатайте все входящие в эту строку заглавные латинские буквы 'A', ... , 'Z' по одному разу в алфавитном порядке с указанием числа вхождений каждой буквы в строку. Проверку на признак заглавной буквы и подсчет количества вхождений в строку оформить в виде функции.
2. Напишите программу, содержащую функцию удаления из вектора элемента перед заданным. Заданный элемент определяется по его порядковому номеру.
3. Пусть в векторе вещественных чисел содержатся результаты измерения некоторой величины x в нескольких опытах. Вычислите среднее значение величины x по формуле $M[X]=\sum_{i=0}^{n-1} x_i$ /n, где n – количество измерений, и дисперсию по

формуле:D[X]=$\sum_{i=0}^{n-1}(M[X]-x_i)^2$ /n. Вычисление среднего арифметического M[X] и дисперсии D[X] организовать в виде отдельных функций.

Исходный код
Результат выполнения

Контрольные вопросы

1. Представление программы в виде функций?
2. Прототип функции?
3. Формальные и фактические параметры?
4. Способ передачи параметров: по ссылке и по значению?
5. Вызов функции?
6. Вычисление квадратного корня?

СПИСОК ЛИТЕРАТУРЫ

1. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
2. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
3. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
4. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.
5. Джонсон Б., Скибо К., Янг М.: Основы Microsoft Visual Studio .NET 2003: - М.:

Отчет по лабораторной работе №3 «Основы С++, работа с файлами»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение принципов работы с файлами в среде MS Visual Studio C++.

Задачи работы:
-знакомство с принципами работы с файлами, основными понятиями и функциями С++
-разработка примеров простейших программ работы с файлами на С++

Краткий конспект теоретической части (ответы на контрольные вопросы)

Типы файлов?_____

Работа с файлами: открытие файла? ___

Работа с файлами: чтение данных из файла?_____

Работа с файлами: запись данных в файл?_____

Подсчет символов и строк в текстовом файле ___

Подсчет символов, строк слов в текстовом файле ___

Пример 1: Работа с файлами: открытие, закрытие файла

```
#include <stdio.h> // Описания стандартного ввода-вывода

int main()
{
FILE *f, *g, *h;

// 1. Открыть текстовый файл "abcd.txt" для чтения
f = fopen("abcd.txt", "rt");
fclose(f)

// 2. Открыть бинарный файл "c:\Windows\Temp\tmp.dat"
// для чтения и записи
g = fopen("tmp.dat", "wb+");
fclose(g)

// 3. Открыть текстовый файл "c:\Windows\Temp\abcd.log"
// для дописывания в конец файла
h = fopen("c:\\Temp\\abcd.log", "at");
fclose(h)

//Диагостика ошибок
FILE *k = fopen("filnam.txt", "rt");
if (k == 0) {
    perror("Не могу открыть файл на чтение");
        fclose(k)

}
}
```

Результат выполнения

Пример 2: Работа с файлами: чтение, запись

```
#include <stdio.h>  // Описания функций ввода-вывода
#include <stdlib.h> // Описание функции exit

int main ()
{
FILE *f;
double buff[10];
size_t res;

f = fopen("tmp.dat", "rb"); // Открываем файл
if (f == 0) { // При ошибке открытия файла
```

```
    // Напечатать сообщение об ошибке
    perror("Не могу открыть файл для чтения");
    exit(1);  // завершить работу с кодом 1
}

// Пытаемся прочесть 10 вещественных чисел из файла
res = fread(buff, sizeof(double), 10, f);
// res равно реальному количеству прочитанных чисел
printf("Read bytes: %d \n", res);

FILE *g;

g = fopen("tmp.res", "wb"); // Открываем файл "tmp.res"
if (g == 0) { // При ошибке открытия файла
    // Напечатать сообщение об ошибке
    perror("Не могу открыть файл для записи");
    exit(1);  // завершить работу программы с кодом 1
}

// Записываем 10 вещественных чисел в файл
res = fwrite(buff, sizeof(double), 10, g);
// В случае успеха res == 10
printf("Written bytes: %d \n", res);

return 0;
}
```

Результат выполнения

Пример 3: Работа с файлами: подсчет символом и строк в текстовом файле:

Программа сначала вводит имя файла с клавиатуры. Для этого используется функция scanf ввода по формату из входного потока, для ввода строки применяется формат "%s. Затем файл открывается на чтение как бинарный (это означает, что при чтении не будет происходить никакого преобразования разделителей строк). Используя в цикле функцию чтения fread, мы считываем содержимое файла порциями по 512 байтов, каждый раз увеличивая суммарное число прочитанных символов.

```
// Подсчет числа символов и строк в текстовом файле
//
#include <stdio.h>  // Описания функций ввода-вывода
#include <stdlib.h> // Описание функции exit

int main() {
    char fileName[256]; // Путь к файлу
    FILE *f;            // Структура, описывающая файл
```

```
    char buff[512];     // Массив для ввода символов
    size_t num;         // Число прочитанных символов
    int numChars = 0;   // Суммарное число символов := 0
    int numLines = 0;   // Суммарное число строк := 0
    int i;              // Переменная цикла

    printf("Введите имя файла: ");
    scanf("%s", fileName);

    f = fopen(fileName, "rb"); // Открываем файл на чтение
    if (f == 0) { // При ошибке открытия файла
        // Напечатать сообщение об ошибке
        perror("Не могу открыть файл для чтения");
        exit(1);  // закончить работу программы с кодом 1
                  // ошибочного завершения
    }

    while ((num = fread(buff, 1, 512, f)) > 0) { // Читаем
        // блок из 512 символов. num -- число реально
        // прочитанных символов. Цикл продолжается, пока
        // num > 0

        numChars += num; // Увеличиваем число символов

        // Подсчитываем число символов перевода строки
        for (i = 0; i < num; ++i) {
            if (buff[i] == '\n') {
                ++numLines; // Увеличиваем число строк
            }
        }
    }

    fclose(f);

    // Печатаем результат
    printf("Число символов в файле = %d\n", numChars);
    printf("Число строк в файле = %d\n", numLines);

    return 0; // Возвращаем код успешного завершения
}
```

Результат выполнения

Другие полезные функции ввода-вывода

Стандартная библиотека ввода-вывода Си содержит ряд других полезных функций ввода-вывода. Отметим некоторые из них.

Посимвольный ввод-вывод	
int fgetc(FILE *f);	ввести символ из потока f
int fputc(int c, FILE *f);	вывести символ в поток f
Построковый ввод-вывод	
char *fgets(char *line,int size, FILE *f);	ввести строку из потока f
char *fputs(char *line, FILE *f);	вывести строку в поток f

Позиционирование в файле	
int fseek(FILE *f, long offset, int whence);	установить текущую позицию в файле f
long ftell(FILE *f);	получить текущую позицию в файле f
int feof(FILE *f);	проверить,достигнут ли конец файла f

Функция fgetc возвращает код введенного символа или константу EOF (определенную как минус единицу) в случае конца файла или ошибки чтения.

Функция fputc записывает один символ в файл. При ошибке fputc возвращает константу EOF (т.е. отрицательное значение), в случае удачи - код выведенного символа c (неотрицательное значение).

Функция `fgets` с прототипом выделяет из файла или входного потока f очередную строку и записывает ее в массив символов line. Второй аргумент size указывает размер массива для записи строки. Максимальная длина строки на единицу меньше, чем size, поскольку всегда в конец считанной строки добавляется нулевой байт. Функция сканирует входной поток до тех пор, пока не встретит симол перевода строки "\n" или пока число введенных символов не станет равным size - 1. Символ перевода строки "\n" также записывается в массив непосредственно перед терминирующим нулевым байтом. Функция возвращает указатель line в случае успеха или нулевой указатель при ошибке или конце файла.

Определение типов символов

Библиотека Си предоставляет следующие функции для определения типа символов, описанные в стандартном заголовочном файле "ctype.h":

int isdigit(int c);	символ c - цифра;
int isalpha(int c);	c - латинская буква;
int isspace(int c);	c - пробел, перевод строки и т.п.;
int ispunkt(int c);	c - знак препинания;
int isupper(int c);	c - прописная латинская буква;
int islower(int c);	c - строчная латинская буква;
int toupper(int c);	если c -- лат. буква, то преобразовать с к прописной букве;
int tolower(int c);	если c -- лат. буква, то преобразовать с к строчной букве.

Функции, начинающиеся с префикса is, возвращают ненулевое значение (т.е. истину), если символ c кодом c принадлежит указанному классу, и нулевое значение (ложь) в противном случае.

Пример 4: Работа с файлами: подсчет числа символом, строк и слов в текстовом файле.

```
// Подсчет числа символов, слов и строк в текстовом файле
//
#include <stdio.h>  // Описания функций ввода-вывода
#include <ctype.h>  // Описания типов символов

int main() {
    char fileName[256]; // Путь к файлу
    FILE *f;            // Структура, описывающая файл
    int c;              // Код введенного символа
```

```
    int numChars = 0;   // Суммарное число символов := 0
    int numLines = 0;   // Суммарное число строк := 0
    int numWords = 0;   // Суммарное число слов := 0
    bool readingWord = false; // Читаем слово := false

    printf("Введите имя файла: ");
    scanf("%s", fileName);

    f = fopen(fileName, "rb"); // Открываем файл
    if (f == 0) { // При ошибке открытия файла
        // Напечатать сообщение об ошибке
        perror("Не могу открыть файл для чтения");
        return 1; // закончить работу программы с кодом 1
    }

    while ((c = fgetc(f)) != EOF) { // Читаем символ
        // Цикл продолжается, пока c != -1 (конец файла)

        ++numChars; // Увеличиваем число символов

        // Подсчитываем число символов перевода строки
        if (c == '\n') {
            ++numLines; // Увеличиваем число строк
        }

        // Подсчитываем число слов
        if (!isspace(c)) {      // если с не пробел
            if (!readingWord) { //  если не читаем слово
                ++numWords;     //   увеличить число слов
                readingWord = true; // читаем слово:=true
            }                   //  конец если
        } else {                // иначе
            readingWord = false;//  читаем слово:=false
        }                       // конец если
    }

    fclose(f);

    // Печатаем результат
    printf("Число символов в файле = %d\n", numChars);
    printf("Число слов = %d\n", numWords);
    printf("Число строк = %d\n", numLines);

    return 0; // Возвращаем код успешного завершения
}
```

Результат выполнения

Задание 1:

1. Составьте программу, которая распечатывает последние n строк файла ввода.
2. Напишите программу, которая читает 2 файла и печатает их вперемежку: одна строка из первого файла, другая - из второго. Придумайте, как поступить, если файлы содержат разное число строк.
3. Дан текстовый файл f. Найти и распечатать самую длинную строку этого файла.

Исходный код
Результат выполнения

Задание 2:

1. Пусть файл содержит записи о сотрудниках некоторого учреждения. Каждая запись включает порядковый номер записи, фамилию, имя сотрудника, его номер телефона и возраст:

```
1 Иванов Иван 1234567 25
2 Петров Петр 2345678 27
…
```

Напишите программу, которая выводит меню с доступными операциями:

```
1 – добавить новую запись (при добавлении новой записи проверять нет ли уже такой записи в файле)
2 – редактировать запись (вводится номер записи)
```

3 – искать запись по фамилии (если имеются однофамильцы, выводить все записи с одинаковыми фамилиями)
4 – удалить запись по порядковому номеру
5 – завершить работу программы

Каждую из операций оформить в виде отдельной функции.

2. Пусть файл содержит информацию о наличии книг в магазине. Каждая запись включает номер ISBN книги, автора, название, количество книг:

9666965348 Стругацкие Град обреченный 258
5845908914 Керниган, Ритчи Язык программирования С 129
…

Напишите программу, которая выводит меню с доступными операциями:

1 – добавить новую книгу (при добавлении проверять нет ли уже такой книги)
2 – редактировать запись (вводится номер ISBN)
3 – поиск книг одного автора (вывести все книги по введенной фамилии автора)
4 – изменить количество книг (по введенному номеру ISBN)
5 – завершить работу программы

Каждую из операций оформить в виде отдельной функции.

Исходный код
Тестовые данные
Результат выполнения

Контрольные вопросы

1. Типы файлов?
2. Работа с файлами: открытие файла?
3. Работа с файлами: чтение данных из файла?
4. Работа с файлами: запись данных в файл?
5. Подсчет символов и строк в текстовом файле?
6. Подсчет символов, строк слов в текстовом файле?

СПИСОК ЛИТЕРАТУРЫ

1. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
2. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
3. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
4. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.
5. Джонсон Б., Скибо К., Янг М.: Основы Microsoft Visual Studio .NET 2003: - М.: Изд-вр Русская редакция, 2003 г. – 464 с.: ил.

Отчет по лабораторной работе №4 «Основы C++, массивы и указатели»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение базовых операций работы с указателями и массивами в C++.

Задачи работы:
-знакомство с указателями в C++
-написание программ работы с массивами через указатели

Краткий конспект теоретической части (ответы на контрольные вопросы)

Объявление массива?_____
_____
_____
_____
_____
Определение размера массива?_____
_____
_____
_____
_____
Понятие указателя?_____
_____
_____
_____
_____
_____
Объявление указателей?_____
_____
_____
_____
_____
Представление массивов через указатели?_____
_____
_____
_____
_____

Основы языка С++. Массивы, указатели.

Пример 1: Объявление массива:

```
#define Nmax 50
/*директива define указывает процессору на то, что встречающееся в коде обозначение Nmax следует заменить на значение 50 */
int main ()
{
char a1[20],a2[2][80];
int b1[25],b2[Nmax];
char a[7]="Привет";
char b[7]={'П','р','и','в','е','т',0x0};
char c[]="Привет";
float d[10]={1.,2.,3.,4.};
int q[2][3]={{1,2,3},
             {4,5,6}};

printf(“a = %s”,a[7]);
printf("b = %s", b);
printf("c = %s",c);
for(int i = 0; i<10; i++)
printf("d[ %d ] = %3.2f \n", i, d[i]);

for (int i  = 0; i < 2; i++)
for (int j = 0; j < 3; j++)
printf("q[ %d ] [ %d ] = %d \n", i, j, q[i][j]);
}
```

Результат выполнения

Пример 2: Определение размера массива:

/*Оператор sizeof возвращает размер всего массива в байтах, а не в элементах массива. Допишите в предыдущую программу следующие строки: */

```
printf("sizeof a = %d \n", sizeof(a));
printf("sizeof d = %d \n", sizeof(d));
printf("sizeof q = %d \n", sizeof(q));
```

Результат выполнения

Задание 1:

1. Написать программу поиска минимального элемента массива из n элементов.
2.Дана последовательность из 20 целых чисел. Определить количество инверсий в этой последовательности (то есть пар таких элементов, в которых большее число находится слева от меньшего: x[i] > x[j] при i<j).
3. Вычислить скалярное произведения двух векторов.

Исходный код
Результат выполнения

Для ввода чисел используйте оператор scanf(“%format”,a), где
format: %d – для переменных типа int, %3.2f – для переменных типа double, float (первое число – количество символов целой части, второе число – количество десятичных разрядов), %c – для переменных типа char, %s – для строковых переменных (массивов из элементов типа char).

Пример 3: Описание указателей:

```
#include "stdlib.h"
#include "iostream" //подключение заголовочного файла с описанием
//функций cout, cin

using namespace std;
int main ()
```

```
{
double **a; //Адрес массива указателей
int m, n;   // Размеры матрицы: m строк, n столбцов
int i,j;
m = 2;
n = 2;
// Захватывается память под массив указателей
a = new double* [2];
for( i = 0; i < 2; i++ )
a[i] = new double [2];

for(i = 0; i < 2; i++)
for(j = 0; j < 2; j++)
cin >> a[i][j];

for(i = 0; i < 2; i++)
for(j = 0; j < 2; j++)
cout << a[i][j] << endl;

delete []a;

int *p, *q;
int b[15] = {5,7,9,11,13,15,17,19,21,23,25,27,29,31,33};
p = &(b[2]); // записываем в p адрес 3-го
             //     элемента массива b
p += 7;      // p будет содержать адрес 12-го эл-та
q = &(a[10]);
--q;         // q содержит адрес элемента a[9]


return 0;
}
```

Результат выполнения

Задание 2:

Работу с массивами организовать через указатели.
1.Переворот одномерного целочисленного массива – перестановка его элементов в обратном порядке. Выделить в отдельную функцию процедуру инвертирования.
2.Перестановка головы и хвоста массива без использования промежуточного массива. Алгоритм этой процедуры заключается он в том, что надо последовательно выполнить 3 инвертирования – головы массива, хвоста массива и всего массива целиком. Инвертирование выделить в отдельную функцию.

1 2 3 4 5 6 7 8 9 10
Голова: 1 2 3 4 5 6
Хвост: 7 8 9 10

Результат: 7 8 9 10 1 2 3 4 5 6

3.Определить количество разных элементов в целочисленном массиве. Алгоритм, выполняющую основную задачу выделить в отдельную функцию.

Исходный код
Результат выполнения

Контрольные вопросы.

1. Объявление массива?
2. Определение размера массива?
3. Понятие указателя ?
4. Объявление указателей?
5. Представление массивов через указатели?

СПИСОК ЛИТЕРАТУРЫ

1. Г. Шилдт Теория и практика C++: пер. с анг. – СПб.: BHV – Санкт-Петербург, 1996ю – 416 с., ил.
2. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с.
3. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
4. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
5. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
6. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.

Отчет по лабораторной работе №5 «Методы сортировки массивов, понятие сложности алгоритма»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение основных методов сортировки массивов. Понятие сложности алгоритма на примере методов сортировки массивов.

Задачи работы:
-ознакомиться с методами сортировки массивов
-написание программ сортировки массивов

Задание повышенной сложности (бонус за сложность – 10 баллов):
Реализовать многофайловую программу: : в заголовочном файле определить необходимые переменные и заголовки функций, во втором файле должен располагаться код функций сортировки, в главном файле – вызов функции main.

Краткий конспект теоретической части (ответы на контрольные вопросы)

Сортировка массива методом пузырька_____
_____
_____
_____
_____
_____
Сортировка массива методом отбора_____
_____
_____
_____
_____
Сортировка массива методом вставки _____
_____
_____
_____
Сортировка массива методом Шелла_____
_____
_____
_____
Быстрая сортировка _____
_____
_____
_____
_____
Сложность алгоритмов _____
_____
_____
_____

Методы сортировки массивов, понятие сложности алгоритма

Пример 1: Сортировка массива методом пузырька

Идея метода состоит в сравнении двух соседних элементов, в результате чего меньшее число (более легкий "пузырек") перемещается на одну позицию влево. Обычно просмотр организуют с конца, и после первого прохода самое маленькое число перемещается на первое место. Затем все повторяется от конца массива до второго элемента и т.д.

```
void bubble(int *x, int n)
{ register int i,j;
  int tmp;
  for(i=1;i<n;i++)
    for(j=n-1;j>=i; j--)
      if(x[j-1]>x[j])
        { tmp=x[j-1]; x[j-1]=x[j]; x[j]=tmp; }
}
```

Задание 1:

написать программу реализующую сортировку массива методом пузырька. Представить блок схему алгоритма.

Тестовые данные
Результат выполнения

Пример 2: Сортировка массива методом отбора

Идея метода: находится элемент с наименьшим значением и меняется местами с первым элементом. Среди оставшихся элементов ищется наименьший, который меняется со вторым и т.д. Функция select, реализующая такую процедуру, приведена ниже:

```
void select(int *x, int n)
{ register int I,j,k;
  int q,tmp;
  for(i=0; i<n-1;i++)
    { q=0; k=I; tmp=x[i];
      for(j=i+1; j<n; j++)
        { if(x[j]<tmp)
            { k=j; tmp=x[j]; q=1; }
        }
      if(q) { x[k]=x[i]; x[i]=tmp; }
    }
}
```

Задание 2:

написать программу реализующую сортировку массива методом отбора. Представить блок схему алгоритма.

Тестовые данные
Результат выполнения

Пример 3: Сортировка массива методом вставки

Идея метода: последовательное пополнение ранее упорядоченных элементов. На первом шаге сортируются два первые элемента. Затем на свое место среди них вставляется третий элемент. К трем упорядоченным добавляется четвертый, который занимает свое место в четверке и т.д. Функция insert, реализующая описанную процедуру, приведена ниже:

```
void insert(int *x, int n)
{ register int i,j;
  int tmp;
  for(i=1;i<n;i++)
    { tmp=x[i];
      for(j=i-1;j>=0 && tmp<x[j]; j--)
        x[j+1]=x[j];
      x[j+1]=tmp;
    }
```

Задание 3:

написать программу реализующую сортировку массива методом вставки. Представить блок схему алгоритма.

Тестовые данные
Результат выполнения

Пример 4: Сортировка массива методом Шелла

В 1959 году сотрудник фирмы IBM D.L. Shell предложил оригинальный алгоритм сортировки. По его предложению сначала сортируются элементы, отстоящие друг от друга на 3 позиции, затем – на две позиции и, наконец, сортируются смежные элементы. В дальнейшем экспериментальным путем были найдены более удачные расстояния между сортируемыми элементами: 9⟶5⟶3⟶2⟶1. Среднее время работы усовершенствованного алгоритма Шелла порядка $n^{1.2}$. Это существенно лучше, чем характерная для трех предыдущих методов величина порядка n^2.

```
void shell(int *x, int n)
{ register int i,j,gap,k;
  int xx;
  char a[5]={9,5,3,2,1};
  for(k=0;k<5;k++)
    { gap=a[k];
      for(i=gap;i<n;i++)
        { xx=x[i];
          for(j=i-gap; xx<x[j] && j>=0; j=j-gap)
            x[j+gap]=x[j];
          x[j+gap]=xx;
        }
     }
}
```

Задание 4:

написать программу реализующую сортировку массива методом Шелла. Представить блок схему алгоритма.

Тестовые данные
Результат выполнения

Пример 5: быстрая сортировка

Известный математик C.A.R. Hoare в 1962 году опубликовал алгоритм быстрой сортировки, за которым закрепилось название quicksort. Основная идея быстрой сортировки напоминает метод поиска делением пополам. Сначала выбирается граничный элемент в сортируемом массиве, его называют компарандом. Все, что больше этого элемента переносится в правую часть массива, а все, что меньше – в левую. После первого шага средний элемент оказывается на своем месте. Затем аналогичная процедура повторяется для каждой половины массива. На каждом последующем шаге размер обрабатываемого фрагмента массива уменьшается вдвое. Количество операций, которое требуется для реализации этой процедуры, оценивается константой $n*\log_2 n$. Это еще быстрее, чем сортировка Шелла. В отличие от предыдущих функций быстрая сортировка оформлена из двух функций – quick, которая допускает принятое в других функциях обращение, и рекурсивной процедуры qs:

```
void quick(int *x, int n)
```

```
{ qs(x,0,n-1); }
//---------------------------------
void qs(int *x,int left,int right)
{ register int i,j;
  int xx,tmp;
  i=left; j=right;
  xx=x[(left+right)/2];
  do { while(x[i]<xx && i<right)i++;
       while(xx<x[j] && j>left) j--;
       if(i<=j)
         { tmp=x[i]; x[i]=x[j];
           x[j]=tmp; i++; j--;
         }
      }
   while(i<=j);
   if(left<j) qs(x,left,j);
   if(i<right)qs(x,i,right);
}
```

Здесь функция quick(int *x, int n) задает вызов основной сортиру.щей функции qs(int *x,int left,int right). Среднее число выполняемых сравнений равно: n*log n. Среднее количество перестановок приблизительно равно: (n*log n)/6. Эти показатели существенно меньше, чем у рассмотренных выше методов сортировки. Следует иметь в виду, если значение-компаранд для каждого из разделов является максимальным, то на практике метод быстрой сортировки вырождается в метод «медленной» сортировки с временем выполнения, пропорциональным квадрату количества элементов.

Задание 5:

написать программу реализующую сортировку массива методом быстрой сортировки. Представить блок схему алгоритма.

Тестовые данные
Результат выполнения

Сложность алгоритмов:
алгоритмы сортировки оцениваются по скорости выполнения и эффективности использования памяти:

- Время — основной параметр, характеризующий быстродействие алгоритма. Называется также вычислительной сложностью. Для упорядочения важны *худшее*, *среднее* и *лучшее* поведение алгоритма в терминах мощности входного множества *A*. Если на вход алгоритму подаётся множество *A*, то обозначим $n = |A|$. Для типичного алгоритма хорошее поведение — это O(n*log(n)) и плохое поведение — это $O(n^2)$. Идеальное поведение для упорядочения — O(n). Алгоритмы сортировки, использующие только абстрактную операцию сравнения ключей всегда нуждаются по меньшей мере в Ω(n*log(n)) сравнениях. Тем не менее, существует алгоритм сортировки Хана (Yijie Han) с вычислительной сложностью O(n*log(log(n))*log(log(log(n)))), использующий тот факт, что пространство ключей ограничено (он чрезвычайно сложен, а за O-обозначением скрывается весьма большой коэффициент, что делает невозможным его применение в повседневной практике). Также существует понятие сортирующих сетей. Предполагая, что можно

одновременно (например, при параллельном вычислении) проводить несколько сравнений, можно отсортировать *n* чисел за $O(\log^2(n))$ операций. При этом число *n* должно быть заранее известно;

- Память — ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. Как правило, эти алгоритмы требуют O(log(n)) памяти. При оценке не учитывается место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы.

Представление кода программы в нескольких файлах

Представим себе, что компоненты программы имеют существенно большие размеры, и зададимся вопросом, как в этом случае распределить их по нескольким файлам. Программу *main* поместим в файл, который мы назовем *main.c*; функции *push, pop* и их переменные расположим во втором файле, *stack.c*; а функцию *getop* - в третьем, *getop.c*. Наконец, функции *getch* и *ungetch* разместим в четвертом файле *getch.c*; мы отделили их от остальных функций.

Существует еще один момент, о котором следует предупредить, - определения и объявления совместно используются несколькими файлами. Нужно, насколько это возможно, централизовать эти объявления и определения так, чтобы для них существовала только одна копия. Тогда программу в процессе ее развития будет легче и исправлять, и поддерживать в нужном состоянии. Для этого общую информацию расположим в заголовочном файле *calc.h*, который будем по мере необходимости включать в другие файлы. В результате получим программу, файловая структура которой показана ниже:

main.c:
```
#include <stdio.h>
#include <stdlib.h>
#include "calc.h"
#define MAXOP 100
main() {
   ...
}
```

calc.h:
```
#define NUMBER '0'
void push(double);
double pop(void);
int getop(char[]);
int getch(void);
void ungetch(int);
```

getop.c:
```
#include <stdio.h>
#include <ctype.h>
#include "calc.h"
getop (){
   ...
}
```

getch.c:
```
#include <stdio.h>
#define BUFSIZE 100
char buf[BUFSIZE];
intbufp = 0;
int getch(void) {
   ...
}
void ungetch(int) {
   ...
}
```

stack.c:
```
#include <stdio.h>
#include "calc.h"
#define MAXVAL 100
int sp = 0;
double val[MAXVAL];
void push(double) {
```

```
   ...
}
double pop(void) {
   ...
}
```

Задание 6:

Методы сортировки должны быть реализованы в виде функций. Работу с массивами организовать через указатели.

1. Написать программу сортировки массивов методом отбора и методом Шелла. Сравнить скорость работы указанных методов сортировки.

2. Написать программу сортировки массивов методом вставки и с помощью быстрой сортировки. Сравнить скорость работы указанных методов сортировки.

3. Написать программу сортировки массивов методом отбора и методом быстрой сортировки. Сравнить скорость работы указанных методов сортировки.

4. Написать программу сортировки массивов методом вставки и методом Шелла. Сравнить скорость работы указанных методов сортировки.

Для вычисление времени работы алгоритма использовать функцию gettimeofday(&tv,NULL) из sys/time.h

```
#include <sys/time.h>
unsigned long GetTickCount()
{
    struct timeval tv;
    gettimeofday(&tv,NULL);
    return (tv.tv_sec*1000+tv.tv_usec/1000);
}
```

Исходный код

Результат выполнения

Контрольные вопросы

1. Сортировка массива методом пузырька?
2. Сортировка массива методом отбора?
3. Сортировка массива методом вставки?
4. Сортировка массива методом Шелла?
5. Быстрая сортировка?
6. Сложность алгоритмов?

СПИСОК ЛИТЕРАТУРЫ

1. Г. Шилдт Теория и практика C++: пер. с анг. – СПб.: BHV – Санкт-Петербург, 1996ю – 416 с., ил.
2. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с.
3. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
4. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
5. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
6. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.

Отчет по лабораторной работе №6 «Основы С++, работа с памятью»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение принципов динамической работы с памятью, работа со строками.

Задачи работы:
-ознакомление с понятием структуры на примере структуры стека и очереди
-разработка программ на С++

Краткий конспект теоретической части (ответы на контрольные вопросы)

Статическая и динамическая память?_____
_____
_____
_____
_____
Захват/освобождение памяти?_____
_____
_____
_____
_____
_____
Печать первых простых n чисел?_____
_____
_____
_____
_____
Понятие структуры данных?_____
_____
_____
_____
_____
Работа со структурами данных: объявление структур?_____
_____
_____
_____
_____
Работа со структурами: нахождение векторного произведения векторов?_____
_____
_____
_____
_____
Работа со структурами данных: поиск числа вершин дерева?_____
_____
_____
_____
_____

Основы C++, работа с памятью.

Пример 1: Выделение и освобождение памяти на примере программы печати первых n простых чисел:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> //подключение модуля math, в котором содержиться описание
//математических фугкций
int main() {
    int n;  // Требуемое количество простых чисел
    int k;  // Текущее количество найденных простых чисел
    int *a; // Указатель на массив найденных простых
    int p;  // Очередное проверяемое число
    int r;  // Целая часть квадратного корня из p
    int i;  // Индекс простого делителя
    bool prime; // Признак простоты

    printf("Введите число простых: ");
    scanf("%d", &n);
    if (n <= 0)   // Некорректное значение =>
        return 1; // завершаем работу с кодом ошибки

    // Захватываем память под массив простых чисел
    a = (int *) malloc(n * sizeof(int));

    a[0] = 2; k = 1;     // Добавляем двойку в массив
    printf("%d ", a[0]); // и печатаем ее

    p = 3;
    while (k < n) {
        // Проверяем число p на простоту
        r = (int)(               // Целая часть корня
            sqrt((double) p) + 0.001
        );
        i = 0;
        prime = true;
        while (i < k && a[i] <= r) {
            if (p % a[i] == 0) { // p делится на a[i]
                prime = false;   // => p не простое,
                break;           // выходим из цикла
            }
            ++i; // К следующему простому делителю
        }
        if (prime) {  // Если нашли простое число,
            a[k] = p; // то добавляем его в массив
            ++k;      // Увеличиваем число простых
            printf("%d ", p); // Печатаем простое число
            if (k % 5 == 0) { // Переход на новую строку
                printf("\n"); // после каждых пяти чисел
            }
        }

        p += 2; // К следующему нечетному числу
    }
    if (k % 5 != 0) {
        printf("\n"); // Перевести строку
    }
    // Освобождаем динамическую память
    free(a);
    return 0;
}
```

Тестовые данные
Результат работы программы

Пример 2: Выделение и освобождение памяти с помощью операторов new и delete.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
{
double *p = new double(1.5);
//этот фрагмент эквивалентен фрагменту
double *p = new double;
*p = 1.5;

double *a;
int m = 100, n = 101;
a = new double[m * n];

return 0;
}
```

Задание 1:

Модифицировать программу нахождения простых чисел из примера 1 с использованием операторов new и delete.

Исходный код
Тестовые данные

Результат выполнения

Пример 3: Пример объявления структуры:

```
#include "stdafx.h"
#include <stdio.h>
#include <tchar.h>

struct R3Vector { // Вектор трехмерного пространства
    double x;
    double y;
    double z;
};

int _tmain(int argc, _TCHAR* argv[])
{
R3Vector u;

printf("Input coordinate x for u \n");
scanf("%lf",&u.x);
printf("Input coordinate y for u \n");
scanf("%lf",&u.y);
printf("Input coordinate z for u \n");
scanf("%lf",&u.z);

printf("3D vector u:\n u.x = %3.2f u.y = %3.2f u.z = %3.2f \n",
w.x, w.y, w.z);


     return 0;
}
```

Результат выполнения

Пример 4: Нахождение векторного произведения векторов:

```
#include <stdio.h>

struct R3Vector { // Вектор трехмерного пространства
    double x;
    double y;
    double z;
};


int main(int argc, char* argv[])
{
R3Vector u, v, w;

printf("Input coordinate x for u \n");
scanf("%lf",&u.x);
printf("Input coordinate y for u \n");
scanf("%lf",&u.y);
printf("Input coordinate z for u \n");
scanf("%lf",&u.z);

printf("Input coordinate x for v \n");
scanf("%lf",&v.x);
printf("Input coordinate y for v \n");
scanf("%lf",&v.y);
printf("Input coordinate z for v \n");
scanf("%lf",&v.z);

// Вычисляем векторное произведение w = u * v
w.x = u.y * v.z - u.z * v.y;
w.y = (-u.x) * v.z + u.z * v.x;
w.z = u.x * v.y - u.y * v.x;

printf("uXv: w.x = %3.2f w.y = %3.2f w.z = %3.2f \n", w.x, w.y,
w.z);


     return 0;
}
```

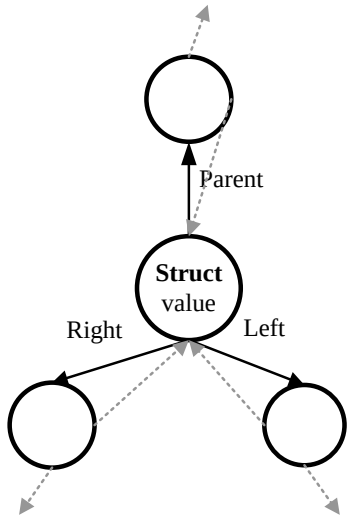
Результат выполнения

Задание 2:

Модифицировать программу нахождение векторного произведения векторов для нахождения скалярного произведения.

Исходный код
Тестовые данные
Результат выполнения

Пример 5: Поиск числа вершин дерева:




```
#include <stdio.h>

     // Описание структуры, представляющей вершину дерева
struct TreeNode {
    struct TreeNode *parent; // Указатель на отца,
    struct TreeNode *left;   //   на левого сына,
    struct TreeNode *right;  //   на правого сына
    int value;             // Значение в вершине
};

int numNodes(const struct TreeNode *root) {
    int num = 0;
    if (root == 0) { // Для нулевого указателя на корень
        return 0;    // возвращаем ноль
    }

    if (root->left != 0) {           // Есть левый сын =>
        num += numNodes(root->left); // вызываем функцию
    }                                // для левого сына

    if (root->right != 0) {          // Есть правый сын =>
        num += numNodes(root->right); // вызываем ф-цию
    }                                  // для правого сына

    return num + 1; // Возвращаем суммарное число вершин
}

int _tmain(int argc, _TCHAR* argv[])
{
TreeNode *root, *left, *right;
root = new TreeNode;
left = new TreeNode;
right = new TreeNode;
root->parent = NULL;
root->value = 1;
```

```
root->left=left;
root->right=right;
left->parent = root;
left->value = 2;
left->left = NULL;
left->right = NULL;
right->parent = root;
right->value = 3;
right->left = NULL;
right->right = NULL;

printf("Num nodes = %d \n",numNodes(root));
     return 0;
}
```

Результат выполнения

Контрольные вопросы

1. Статическая и динамическая память?
2. Захват/освобождение памяти?
3. Печать первых простых n чисел?
4. Понятие структуры данных?
5. Работа со структурами данных: объявление структур?
6. Работа со структурами: нахождение векторного произведения векторов?
7. Работа со структурами данных: поиск числа вершин дерева?

СПИСОК ЛИТЕРАТУРЫ

1. Браунси К. Основные концепции структур данных и реализация в C++: Пер. с англ. – М.: Изд-во Вильямс, 2002 г. – 320 с.: ил.
2. Топп У., Форд У. Структуры данных в C++: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2006 г. – 816 с. :ил.
3. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
4. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
5. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
6. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.

Отчет по лабораторной работе №7 «Основы С++, работа с памятью»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение принципов динамической работы с памятью, работа со строками, написание программы-калькулятора.

Задачи работы:
-ознакомление с понятием структуры на примере структуры стека и очереди
-разработка стекового калькулятора согласно обратной форме записи

Краткий конспект теоретической части (ответы на контрольные вопросы)

Динамические структуры: стек ?____

Динамические структуры: очередь?

Реализация стека на базе массива?

Реализация основных функций работы со стеком?

Стековый калькулятор и обратная польская запись формулы?

Парсер строки ввода калькулятора?

Пример 1: Понятие стека. Реализация стека на базе массива

Базой реализации является массив размера N, таким образом, реализуется стек ограниченного размера, максимальная глубина которого не может превышать N. Индексы ячеек массива изменяются от 0 до N - 1. Элементы стека хранятся в массиве следующим образом: элемент на дне стека располагается в начале массива, т.е. в ячейке с индексом 0. Элемент, расположенный над самым нижним элементом стека, хранится в ячейке с индексом 1, и так далее. Вершина стека хранится где-то в середине массива. Индекс элемента на вершине стека хранится в специальной переменной, которую обычно называют указателем стека (по-английски Stack Pointer или просто SP).

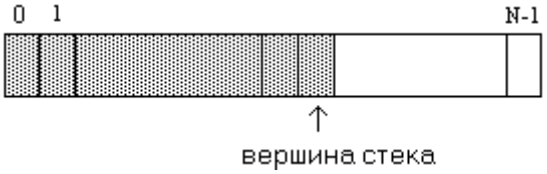


Рис. 3.6 Стек на баземассива.

Когда стек пуст, указатель стека содержит значение минус единица. При добавлении элемента указатель стека сначала увеличивается на единицу, затем в ячейку массива с индексом, содержащимся в указателе стека, записывается добавляемый элемент. При извлечении элемента из стека сперва содержимое ячейки массива с индексом, содержащимся в указателе стека, запоминается во временной переменной в качестве результата операции, затем указатель стека уменьшается на единицу.

В приведенной реализации стек растет в сторону увеличения индексов ячеек массива.

Файл streal.h - описывает интерфейс исполнителя "Стек"
Файл streal.cpp" – реализует функции работы со стеком.
Стек растет в сторону увеличения индексов массива. Пространство под массив элементов стека захватывается в динамической памяти в момент инициализации стека.
Функции инициализации st_init передается размер массива, т.е. максимально возможное число элементов в стеке.
Для завершения работы стека нужно вызвать функцию st_terminate, которая освобождает захваченную в st_init память. Ниже приведено содержимое файла "streal.h", описывающего интерфейс стека.

```
// Файл "streal.h"
// Стек вещественных чисел, интерфейс
//
#ifndef ST_REAL_H
#define ST_REAL_H

// Прототипы функций, реализующих предписания стека:

void st_init(int maxSize); // Начать работу (вх: цел
                           //     макс. размер стека)
void st_terminate();    // Закончить работу
void st_push(double x); // Добавить эл-т (вх: вещ x)
double st_pop();        // Взять элемент: вещ
double st_top();        // Вершина стека: вещ
int st_size();          // Текущий размер стека: цел
bool st_empty();        // Стек пуст? : лог
int st_maxSize();       // Макс. размер стека: цел
bool st_freeSpace();    // Есть свободное место? : лог
void st_clear();        // Удалить все элементы
double st_elementAt(int i); // Элемент стека на
                            //   глубине (вх: i): вещ
#endif
```

```
// Конец файла "streal.h"
```

Отметим, что директивы условной трансляции

```
#ifndef ST_REAL_H
#define ST_REAL_H
. . .
#endif
```

используются для предотвращения повторного включения h-файла: при первом включении файла определяется переменная препроцессора ST_REAL_H, а директива "#ifndef ST_REAL_H" подключает текст, только если эта переменная не определена. Такой трюк используется практически во всех h-файлах. Нужен он потому, что одни h-файлы могут подключать другие, и без этого механизма избежать повторного включения одного и того же файла трудно.

Пример2: Реализация основных функций работы со стеком:

Файл "streal.cpp" описывает общие статические переменные, над которыми работают функции, соответствующие предписаниям стека, и реализует эти функции.

```
// Файл "streal.cpp"
// Стек вещественных чисел, реализация
//
#include <stdlib.h>
#include <assert.h>
#include "streal.h" // Подключить описания функций стека

// Общие переменные для функций, реализующих
// предписания стека:
static double *elements = 0; // Указатель на массив эл-тов
                             //     стека в дин. памяти
static int max_size = 0;     // Размер массива
static int sp = (-1);        // Индекс вершины стека

// Предписания стека:

void st_init(int maxSize) { // Начать работу (вх:
                            //     макс. размер стека)
    assert(elements == 0);
    max_size = maxSize;
    elements = (double *) malloc(
        max_size * sizeof(double)
    );
    sp = (-1);
}

void st_terminate() { // Закончить работу
    if (elements != 0) {
        free(elements);
    }
}

void st_push(double x) { // Добавить эл-т (вх: вещ x)
    assert(              // утв:
        elements != 0 && //   стек начал работу и
        sp < max_size-1  //   есть своб. место
```

```
    );
    ++sp;
    elements[sp] = x;
}

double st_pop() { // Взять элемент: вещ
    assert(sp >= 0); // утв: стек не пуст
    --sp;            // элемент удаляется из стека
    return elements[sp + 1];
}

double st_top() { // Вершина стека: вещ
    assert(sp >= 0); // утв: стек не пуст
    return elements[sp];
}

int st_size() { // Текущий размер стека: цел
    return (sp + 1);
}

bool st_empty() { // Стек пуст? : лог
    return (sp < 0);
}

int st_maxSize() { // Макс. размер стека: цел
    return max_size;
}

bool st_freeSpace() { // Есть своб. место? : лог
    return (sp < max_size - 1);
}

void st_clear() { // Удалить все элементы
    sp = (-1);
}

double st_elementAt(int i) { // Элемент стека на
                             //   глубине (вх: i): вещ
    assert(                     // утв:
        elements != 0 &&        // стек начал работу и
        0 <= i && i < st_size() // 0 <= i < размер стека
    );
    return elements[sp - i];
}
// Конец файла "streal.cpp"
```

В реализации стека неоднократно использовалась функция assert. Фактическим аргументом функции является логическое выражение. Если оно истинно, то ничего не происходит; если ложно, то программа завершается аварийно, выдавая диагностику ошибки.

Задание 1:

на базе представленного выше описания основных функция работы со стеком написать основную часть программы (содержание функции main), демонстрирующую использование написанных операций, демонстрирующих работу со стеком.

Исходный код функции main
Тестовые данные
Результат выполнения

Обратная польская запись формулы.

В 1920 г. польский математик Ян Лукашевич предложил способ записи арифметических формул, не использующий скобок. В привычной нам записи знак операции записывается между аргументами, например, сумма чисел 2 и 3 записывается как 2 + 3. Ян Лукашевич предложил две другие формы записи: префиксная форма, в которой знак операции записывается перед аргументами, и постфиксная форма, в которой знак операции записывается после аргументов. В префиксной форме сумма чисел 2 и 3 записывается как + 2 3, в постфиксной — как 2 3 +. В честь Яна Лукашевича эти формы записи называют прямой и обратной польской записью.

В польской записи скобки не нужны. Например, выражение

(2+3)*(15-7)

записывается в прямой польской записи как

* + 2 3 - 15 7,

в обратной польской записи — как

2 3 + 15 7 - *.

Если прямая польская запись не получила большого распространения, то обратная оказалась чрезвычайно полезной. Неформально преимущество обратной записи перед прямой польской записью или обычной записью

можно объяснить тем, что гораздо удобнее выполнять некоторое действие, когда объекты, над которыми оно должно быть совершено, уже даны.
Обратная польская запись формулы позволяет вычислять выражение любой сложности, используя стек как запоминающее устройство для хранения промежуточных результатов.
Для вычисления выражения надо сначала преобразовать его в обратную польскую запись (при некотором навыке это легко сделать в уме). В приведенном выше примере выражение (2+3)*(15-7) преобразуется к
2 3 + 15 7 - *
Затем обратная польская запись просматривается последовательно слева направо. Если мы видим число, то просто вводим его в калькулятор, т.е. добавляем его в стек. Если мы видим знак операции, то нажимаем соответствующую клавишу калькулятора, выполняя таким образом операцию с числами на вершине стека.
Изобразим последовательные состояния стека калькулятора при вычислении по приведенной формуле. Сканируем слева направо ее обратную польскую запись:
2 3 + 15 7 - *
Стек вначале пуст. Последовательно добавляем числа 2 и 3 в стек.

| | вводим число 2 ⟶ | 2 | вводим число 3 ⟶ | 3 |
| | | 2 |

Далее читаем символ + и нажимаем на клавишу + калькулятора. Числа 2 и 3 извлекаются из стека, складываются, и результат помещается обратно в стек.

| 3 | выполняем сложение ⟶ | 5 |
| 2 |

Далее, в стек добавляются числа 15 и 7.

5	вводим число 15 ⟶	15	вводим число 7	7
	5		15
		5

Читаем символ - и нажимаем на клавишу - калькулятора. Со стека при этом снимаются два верхних числа 7 и 15 и выполняется операция вычитания. Причем уменьшаемым является то число, которое было введено раньше, а вычитаемым — число, введенное позже. Иначе говоря, при выполнении некоммутативных операций, таких как вычитание или деление, правым аргументом является число на вершине стека, левым — число, находящееся под вершиной стека.

| 7 | выполняем вычитание ⟶ | 8 |
| 15 | | 5 |
| 5 |

Наконец, читаем символ * и нажимаем на клавишу * калькулятора. Калькулятор выполняет умножение, со стека снимаются два числа, перемножаются, результат помещается обратно в стек.

| 8 | выполняем умножение ⟶ | 40 |
| 5 |

Число 40 является результатом вычисления выражения. Оно находится на вершине стека и высвечивается на дисплее стекового калькулятора.

Задание 2:

Написать программу, реализующую калькулятор. В проект должны входить три файла: "streal.h", "streal.cpp" и "stcalc.cpp". Первые два файла реализуют стек вещественных чисел, эта реализация уже рассматривалась выше. Файл "stcalc.cpp" реализует стековый калькулятор на базе стека, его и нужно добавить к проекту.
Пользователь вводит строку из чисел и арифметических операций – программа должна распарсить строку, записать числа в стек, а при наличии арифметической операции выполнить ее и поместить результат обратно в стек. Пользователь может ввести число с клавиатуры, это число просто добавляется в стек. При вводе одного из четырех знаков арифметических операций +, -, *, / программа извлекает из стека два числа, выполняет указанное арифметическое действие над ними и помещает результат обратно в стек. Значение результата отображается также на дисплее. Кроме арифметических операций, пользователь может ввести название одной из стандартных функций: sin, cos, exp, log (натуральный логарифм). При этом программа извлекает из стека аргумент функции, вычисляет значение функции и помещает его обратно в стек. При желании список стандартных функций и возможных операций можно расширить. Каждую команду стекового

калькулятора нужно реализовать в виде отдельной функции. Например, вычитание реализуется с помощью функции onSub():

```
static void onSub() {
    double y, x;
    if (st_size() < 2) {
        printf("Stack depth < 2.\n");
        return;
    }
    y = st_pop();
    x = st_pop();
    st_push(x - y);
    display();
}
```

Исходный код файла stcalc.cpp
Тестовые данные
Результат выполнения

Контрольные вопросы

1. Динамические структуры: стек?
2. Динамические структуры: очередь?
3. Реализация стека на базе массива?
4. Реализация основных функций работы со стеком?
5. Стековый калькулятор и обратная польская запись формулы?
6. Парсер строки ввода калькулятора?

СПИСОК ЛИТЕРАТУРЫ

1. Браунси К. Основные концепции структур данных и реализация в C++: Пер. с англ. – М.: Изд-во Вильямс, 2002 г. – 320 с.: ил.
2. Топп У., Форд У. Структуры данных в C++: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2006 г. – 816 с. :ил.
3. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
4. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
5. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
6. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.

Отчет по лабораторной работе №8 «Основы C++, работы с двоичными файлами, RLE енкодер/декодер»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение основ работы с двоичными данными, ознакомление с алгоритмом сжатия RLE.

Задачи работы:
- закрепить знания по работе с файлами в C++
-реализовать енкодер и декодер для алгоритма сжатия RLE

Краткий конспект теоретической части (ответы на контрольные вопросы)

Операции чтения и записи с двоичными файлами_____
_____
_____
_____
_____
Первый вариант алгоритма сжатия RLE: енкодер_____
_____
_____
_____
_____
Первый вариант алгоритма сжатия RLE: декодер_____
_____
_____
_____
_____
_____
Второй вариант алгоритма сжатия RLE: енкодер_____
_____
_____
_____
_____
Второй вариант алгоритма сжатия RLE: декодер___
_____
_____
_____
_____
Характеристики алгоритма RLE_____
_____
_____
_____

Работа с файлами. Енкодер/декодер RLE.

Пример 1: Рассмотрим программу, которая создает двоичный файл для записи с именем c_bin и записывает в него 4*10 порций данных в машинном формате (строки, целые и вещественные числа). После записи данных файл закрывается и вновь открывается для чтения. Для демонстрации прямого доступа к данным информация из файла считывается в обратном порядке – с конца. Контроль записываемой и считываемой информации обеспечивается дублированием данных на экране дисплея.

```
#include <stdlib.h>
#include <math.h>
#include <conio.h>
int  main( )
{ FILE *f1;            //указатель на блок управления файлом
  int j,k;
  char s[]="Line";
  int n;
  float r;
  f1=fopen("c_bin","wb");      //создание двоичного файла для записи
  for(j=1;j<11;j++)
    { r=sqrt(j);
      fwrite(s,sizeof(s),1,f1);              //запись строки в файл
      fwrite(&j,sizeof(int),1,f1);     //запись целого числа в файл
      fwrite(&r,sizeof(float),1,f1);  //запись вещественного числа
      printf("\n%s %d %f",s,j,r);     //контрольный вывод
    }
  fclose(f1);                  //закрытие файла
  printf("\n");
  f1=fopen("c_bin","rb");      //открытие двоичного файла для чтения
  for(j=10; j>0; j--)
    {//перемещение указателя файла
      fseek(f1,(j-1)*(sizeof(s)+sizeof(int)+sizeof(float)),SEEK_SET);
      fread(&s,sizeof(s),1,f1);              //чтение строки
      fread(&n,sizeof(int),1,f1);      //чтение целого числа
      fread(&r,sizeof(float),1,f1);   //чтение вещественного числа
      printf("\n%s %d %f",s,n,r);     //контрольный вывод
    }
  getch();
  return 0;
}
```

Результат создания проекта

Пример 2: Приведенная ниже программа является модификацией предыдущего примера. Единственное ее отличие состоит в использовании структуры (записи) b, состоящей из символьного (b.s, 5 байт, включая нулевой байт – признак конца строки), целочисленного (b.n, 2 байта в BC и 4 байта в BCB) и вещественного (b.r, 4 байта) полей.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <conio.h>
  main( )
{ FILE *f1;
  int j,k;
```

```
  struct {
    char s[5];
    int n;
    float r;
  } b;
  strcpy(b.s,"Line");
  f1=fopen("c_rec","wb");
  for(j=1;j<11;j++)
    { b.n=j;    b.r=sqrt(j);
      fwrite(&b,sizeof(b),1,f1);
      printf("\n%s %d %f",b.s,b.n,b.r);
    }
  fclose(f1);
  printf("\n");
  f1=fopen("c_rec","rb");
  for(j=10; j>0; j--)
    { fseek(f1,(j-1)*sizeof(b),SEEK_SET);
      fread(&b,sizeof(b),1,f1);
      printf("\n%s %d %f",b.s,b.n,b.r);
    }
  getch();
}
```

Результат выполнения

Задание 1:

1. создать копию бинарного файла путем последовательного чтения данных из оригинального файла и записи их в копию.
2. написать программу побайтного сравнения двух файлов.

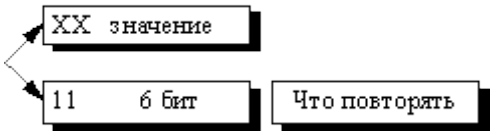
Исходный код
Результат выполнения

Пример 3: Первый вариант алгоритма сжатия без потерь RLE:

Сжатие в RLE происходит за счет того, что в исходном файле встречаются цепочки одинаковых байт. Замена их на пары <счетчик повторений, значение> уменьшает избыточность данных. Рассмотрим, например, следующую строку: `abbbbzzzzddaaaaafffuu` после применения группового сжатия эта последовательность будет преобразована в: `1a4b4z2d5a4f2u.` В данном алгоритме признаком счетчика (counter) служат единицы в двух верхних битах считанного байта. Соответственно оставшиеся 6 бит расходуются на счетчик, который может принимать значения от 1 до 64. Строку из 64 повторяющихся байтов мы превращаем в два байта, т.е. сожмем в 32 раза. Если же кодируемая последовательность содержит единичный байт, значение которого больше 191, то он так же кодируется парой <счетчик, значение>.

Алгоритм декомпрессии при этом выглядит так: считываем счетчик

```
Initialization(...);
do {
    byte = ImageFile.ReadNextByte();
    if(является счетчиком(byte)) {
        counter = Low6bits(byte)+1;
        value = ImageFile.ReadNextByte();
        for(i=1 to counter)
            DecompressedFile.WriteByte(value)
        }
    else {
        DecompressedFile.WriteByte(byte)
} while(ImageFile.EOF());
```



Задание2:

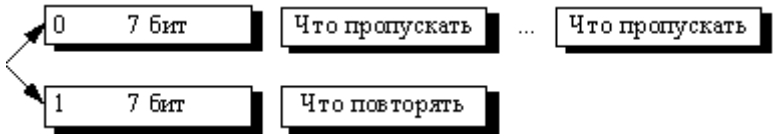
Начертить блок-схему первого варианта алгоритма RLE

Блок-схема енкодера

Блок-схема декодера

Пример 3: второй вариант алгоритма сжатия без потерь RLE:

Второй вариант этого алгоритма имеет больший максимальный коэффициент архивации и меньше увеличивает в размерах исходный файл. Признаком повтора в данном алгоритме является единица в старшем разряде соответствующего байта:



Алгоритм декомпрессии для него выглядит так: считываем счетчик повторяемых или оригинальных байт. Если первый бит счетчика установлен в 1, значит следующий байт следует повторить n раз, n={0;127}. Как извлечь младшие 7 разрядов из байта см. в описании лабораторной работы 1 (побитовые логические операции). Если первый бит счетчика установлен в 0, значит следующие n байт (n={0;127}) следует записать по одному разу.

```
Initialization(...);
do {
    byte = ImageFile.ReadNextByte();
    counter = Low7bits(byte)+1;
    if(если признак повтора(byte)) {
        value = ImageFile.ReadNextByte();
        for (i=1 to counter)
        CompressedFile.WriteByte(value)
    }
    else {
    for(i=1 to counter){
        value = ImageFile.ReadNextByte();
        CompressedFile.WriteByte(value)
    }
    CompressedFile.WriteByte(byte)
} while(ImageFile.EOF());
```

Задание 3:

Начертить блок-схеме для второго варианта алгоритм RLE

Блок-схема енкодера
Блок-схема декодера

Характеристики алгоритма RLE:

Коэффициенты компрессии:
первый вариант: 32, 2, 0,5; (лучший, средний, худший коэффициенты);

второй вариант: 64, 3, 128/129; (лучший, средний, худший коэффициенты).

Класс изображений: ориентирован алгоритм на изображения с небольшим количеством цветов: деловую и научную графику.

Симметричность: примерно единица.

Характерные особенности: К положительным сторонам алгоритма, пожалуй, можно отнести только то, что он не требует дополнительной памяти при архивации и разархивации, а также быстро работает. Интересная особенность группового кодирования состоит в том, что степень архивации для некоторых изображений может быть существенно повышена всего лишь за счет изменения порядка цветов в палитре изображения.

Задание 4:

1. Реализовать RLE encoder/decoder по первому алгоритму.
2. Реализовать RLE encoder/decoder по второму алгоритму.

Программа должна предоставлять пользователю возможность: загрузить произвольный файл, сжать его согласно варианту задания, загрузить сжатый файл, декодировать сжатый файл.

При сдаче лабораторной предоставить два файла, один из которых демонстрирует эффективность работы алгоритма, а второй – наоборот, показывает, увеличение объема сжатого файла по сравнению с исходным.

Исходный код
Тестовые данные
Результат выполнения

Контрольные вопросы

1. Операции чтения и записи с двоичными файлами?
2. Первый вариант алгоритма сжатия RLE: енкодер?
3. Первый вариант алгоритма сжатия RLE: декодер?
4. Второй вариант алгоритма сжатия RLE: енкодер?
5. Второй вариант алгоритма сжатия RLE: декодер?
6. Характеристики алгоритма RLE?

СПИСОК ЛИТЕРАТУРЫ

1. Ватолин Д. С. Алгоритмы сжатия изображений. Методическое пособие: М.: Издательский отдел факультета Вычислительной Математики и Кибернетики МГУ им. М.В.Ломоносова, 1999 г. — 76 с.
2. Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д. Структуры данных и алгоритмы. — Издательский дом «Вильямс», 2000. — С. 384.
3. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
4. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
5. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
6. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.

Отчет по лабораторной работе №10 «Основы С++, работа с сокетами. Часть1.»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение принципов функционирования сетевых соединений, ознакомление с программированием сокетов и потоков.

Задачи работы:
-ознакомление с понятием сетевого соединения и сокетов
-разработка простейшего клиента

Краткий конспект теоретической части (ответы на контрольные вопросы)

Понятие сокета_____

Типы сокетов_____

Создание сокетов_____

Понятие IP адреса_____

Связывание сокета_____

Создание канала связи (клиент/сервер)_____

Передача данных _____

UDP-клиент?_____

Основы C++, работа с сокетами. Часть1.

Пример 1: Создание сокета:

Инициализация библиотеки WinSock, содржащей функции работы с сокетами.
int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData);

wVersionRequested версия библиотеки WinSock, необходимая для работы вашего приложения.

lpWSAData - параметр, содержащий указатель на структуру типа WSADATA , в которую будут записаны сведения о конкретной реализации интерфейса Windows Sockets.

В случае успеха функция WSAStartup возвращает нулевое значение. Если происходит ошибка, возвращается одно из следующих значений:

Значение	Описание
WSASYSNOTREADY	Сетевое программное обеспечение не готово для работы
WSAVERNOTSUPPORTED	Функция не поддерживается данной реализацией интерфейса Windows Sockets
WSAEINVAL	Библиотека DLL, обеспечивающая интерфейсе Windows Sockets, не соответствуетверсии, указанной приложением указанной в параметре wVersionRequested

Освобождение ресурсов, выделенных для работы с сокетами:

int WSACleanup (void);

Эта функция может возвратить нулевое значение при успехе или значение SOCKET_ERROR в случае ошибки.

Получение кода последней ошибки: int WSAGetLastError (void);

Создание сокета: SOCKET socket (int af, int type, int protocol);

Af - формат адреса. Для этого параметра вы должны указывать значение AF_INET , что соответствует формату адреса, принятому в Internet.

type - тип сокета

protocol - протокол, который будет использован для данного сокета.

Типы сокетов:

Тип сокета	Описание
SOCK_STREAM	Сокет будет использован для передачи данных через канал связи с использованием протокола TCP
SOCK_DGRAM	Передача данных будет выполняться без создания каналов связи через датаграммный протокол UDP
RAW	возможность пользовательского доступа к низлежащим коммуникационным протоколам, поддерживающим сокет-абстракции. Такие сокеты обычно являются датаграм-ориентированными.

Что же касается параметра protocol, то вы можете указать для него нулевое значение, в этом случае необходимый протокол будет определен по умолчанию, исходя из предыдущих параметров.

В случае успеха функция socket возвращает дескриптор, который нужно использовать для выполнения всех операций над данным сокетом. Если же произошла ошибка, эта функция возвращает значение INVALID_SOCKET

Ниже представлен фрагмент кода, в котором создается сокет для передачи данных с использованием протокола TCP:

```
srv_socket  = socket(AF_INET , SOCK_STREAM, 0);
if(srv_socket  == INVALID_SOCKET)
{
```

MessageBox(NULL, "socket Error", "Error", MB_OK);
return;
}

Удаление сокета: int closesocket (SOCKET sock);

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string.h>
#include <winsock2.h>
#include <windows.h>
using namespace std;

#define PORT 777
#define SERVERADDR "127.0.0.1"

int main(int argc, char* argv[])
{
char buff[1024];
printf("TCP DEMO CLIENT\n");

// Шаг 1 - инициализация библиотеки Winsock
if (WSAStartup(0x202,(WSADATA *)&buff[0]))
{
printf("WSAStart error %d\n",WSAGetLastError());
return -1;
}
SOCKET my_sock;
my_sock=socket(AF_INET,SOCK_STREAM,0);
if (my_sock<0)
{
printf("Socket() error %d\n",WSAGetLastError());
return -1;
}
closesocket(my_sock);
WSACleanup();
return -1;
}
```

Результат создания проекта

Пример 2:Привязка сокета с адресам:

Для связывания сокета с адресом и номером порта используют системный вызов bind:
int bind (
SOCKET sock, const struct sockaddr FAR * addr, int namelen);

Параметр sock должен содержать дескриптор сокета, созданного функцией socket .

В поле addr следует записать указатель на подготовленную структуру SOCKADDR , а в поле namelen - размер этой структуры.

В случае ошибки функция bind возвращает значение SOCKET_ERROR . Дальнейший анализ причин ошибки следует выполнять при помощи функции WSAGetLastError

Перед использованием необходимо задать параметры сокета.

Для этого нужно подготовить структуру типа sockaddr , определение которой показано ниже:

```
struct sockaddr
{
  u_short sa_family;
  char    sa_data[14];
};
```

Для работы с адресами в формате Internet используется другой вариант этой структуры, в котором детализируется формат поля sa_data:

```
struct sockaddr_in
{
  short   sin_family;
  u_short sin_port;
  struct  in_addr  sin_addr;
  char    sin_zero[8];
};
```

sin_family - определяет тип адреса. Вы должны записать в это поле значение AF_INET , которое соответствует типу адреса, принятому в Internet:

```
srv_address.sin_family = AF_INET ;
```

Поле sin_port определяет номер порта, который будет использоваться для передачи данных.

Особенностью поля sin_port является использование так называемого сетевого формата данных. Этот формат отличается от того, что принят в процессорах с архитектурой Intel, а именно, младшие байты данных хранятся по старшим адресам памяти. Напомним, что архитектура процессоров Intel подразумевает хранение старших байтов данных по младшим адресам.

Для выполнения преобразований из обычного формат в сетевой и обратно в интерфейсе Windows Sockets предусмотрен специальный набор функций. В частности, для заполнения поля sin_port нужно использовать функцию htons, выполняющую преобразование 16-разрядных данных из формата Intel в сетевой формат.

Ниже показано, как инициализируется поле sin_port в приложении SERVER:

```
#define SERV_PORT 5000
srv_address.sin_port = htons(SERV_PORT);
```

Вернемся снова к структуре sockaddr_in .

Поле sin_addr этой структуры представляет собой структуру in_addr:

```
struct in_addr
{
  union
  {
    struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
    struct { u_short s_w1,s_w2; } S_un_w;
    u_long S_addr;
  } S_un;
};
```

При инициализации сокета в этой структуре нужно указать адрес IP, с которым будет работать данный сокет.

Если сокет будет работать с любым адресом (например, вы создаете сервер, который будет доступен из узлов с любым адресом), адрес для сокета можно указать следующим образом:

```
srv_address.sin_addr .s_addr = INADDR_ANY ;
```

В том случае, если сокет будет работать с определенным адресом IP (например, вы создаете приложение-клиент, которое будет обращаться к серверу с конкретным адресом IP), в указанную структуру необходимо записать реальный адрес.

Датаграммный протокол UDP позволяет посылать пакеты данных одновременно всем рабочим станциям в широковещательном режиме. Для этого вы должны указать адрес как INADDR_BROADCAST.

Если вам известен адрес в виде четырех десятичных чисел, разделенных точкой (именно так его вводит пользователь), то вы можете заполнить поле адреса при помощи функции inet_addr :

dest_sin.sin_addr .s_addr = inet_addr ("200.200.200.201");

В случае ошибки функция возвращает значение INADDR_NONE , что можно использовать для проверки.

Обратное преобразование адреса IP в текстовую строку можно при необходимости легко выполнить с помощью функции inet_ntoa , имеющей следующий прототип:

char FAR * inet_ntoa (struct in_addr in);

При ошибке эта функция возвращает значение NULL.

Однако чаще всего пользователь работает с доменными именами, используя сервер DNS или файл HOSTS . В этом случае вначале вы должны воспользоваться функцией gethostbyname , возвращающей адрес IP, а затем записать полученный адрес в структуру sin_addr :

```
PHOSTENT  phe;
phe = gethostbyname ("ftp.microsoft.com");
if(phe == NULL)
{
  closesocket  (srv_socket);
  MessageBox(NULL, "gethostbyname  Error", "Error", MB_OK);
  return;
}
memcpy((char FAR *)&(dest_sin.sin_addr ),
  phe->h_addr , phe->h_length);
```

В случае ошибки функция gethostbyname возвращает NULL. При этом причину ошибки можно выяснить, проверив код возврата функции WSAGetLastError .

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string.h>
#include <winsock2.h>
#include <windows.h>
using namespace std;

#define PORT 777

int main(int argc, char* argv[])
{
char buff[1024];
printf("TCP DEMO CLIENT\n");

// Шаг 1 - инициализация библиотеки Winsock
if (WSAStartup(0x202,(WSADATA *)&buff[0]))
{
printf("WSAStart error %d\n",WSAGetLastError());
return -1;
}

char srv[15];
cout << "Server adress: \n";
 cin >> srv;
 srv[sizeof(srv)]=0;
cout << srv << "\n";
// Шаг 2 - создание сокета
```

```
SOCKET my_sock;
my_sock=socket(AF_INET,SOCK_STREAM,0);
if (my_sock<0)
{
printf("Socket() error %d\n",WSAGetLastError());
return -1;
}

// Шаг 3 - установка соединения

// заполнение структуры sockaddr_in – указание адреса и порта
сервера
sockaddr_in dest_addr;
dest_addr.sin_family=AF_INET;
dest_addr.sin_port=htons(PORT);
HOSTENT *hst;

// преобразование IP адреса из символьного в сетевой формат
//if (inet_addr(SERVERADDR)!=INADDR_NONE)
if (inet_addr(srv)!=INADDR_NONE)
//dest_addr.sin_addr.s_addr=inet_addr(SERVERADDR);
dest_addr.sin_addr.s_addr=inet_addr(srv);
else
// попытка получить IP адрес по доменному имени сервера
//if (hst=gethostbyname(SERVERADDR))
if (hst=gethostbyname(srv))
// hst->h_addr_list содержит не массив адресов, а массив
указателей на адреса
((unsigned long *)&dest_addr.sin_addr)[0]=((unsigned long **)hst-
>h_addr_list)[0][0];
else
{
//printf("Invalid address %s\n",SERVERADDR);
     printf("Invalid address %s\n",srv);
closesocket(my_sock);
WSACleanup();
return -1;
}
}
```

Результат выполнения

Пример 3: Создание канала связи: клиент

Если вы собираетесь передавать датаграммные сообщения при помощи протокола негарантированной доставки UDP , канал связи не нужен. Сразу после создания сокетов и их инициализации можно приступать к передаче данных. Но для передачи данных с использованием протокола TCP необходимо создать канал связи.
Со стороны клиента связь устанавливается с помощью стандартной функции connect:
*int **connect** (SOCKET s, const struct sockaddr FAR* name, int namelen*

Клиент:
Первый слева аргумент - дескриптор сокета, возращенный функцией socket; второй - указатель на структуру *"sockaddr"*, содержащую в себе адрес и порт удаленного узла, с которым устанавливается соединение. Структура sockaddr используется множеством функций, поэтому ее описание вынесено в отдельный раздел "Адрес раз, адрес два...". Последний аргумент сообщает функции размер структуры sockaddr.
Если по каким-то причинам это сделать не удастся (адрес задан неправильно, узел не существует или "висит", компьютер находится не в сети), функция возвратит ненулевое значение.
Клиент:
после того как соединение установлено, потоковые сокеты могут обмениваться с удаленным узлом данными, вызывая функции *"int **send** (SOCKET s, const char FAR * buf, int len,int flags)"* и *"int **recv** (SOCKET s, char FAR* buf, int len, int flags)"* для посылки и приема данных соответственно.
Функция *send* возвращает управление сразу же после ее выполнения, независимо от того, получила ли принимающая сторона наши данные или нет.Функция же *recv* возвращает управление только после того, как получит хотя бы один байт.
Работой обоих функций можно управлять с помощью *флагов*, передаваемых в одной переменной типа int третьим слева аргументом. Эта переменная может принимать одно из двух значений: *MSG_PEEK* и *MSG_OOB*.
Флаг MSG_PEEK заставляет функцию recv просматривать данные вместо их чтения. Просмотр в отличие от чтения не уничтожает просматриваемые данные. Некоторые источники утверждают, что при взведенном флаге MSG_PEEK функция recv не задерживает управления, если в локальном буфере нет данных, доступных для немедленного получения. Это неверно! Аналогично, иногда приходится встречать откровенно ложное утверждение о том, что якобы функция send со взведенным флагом MSG_PEEK возвращает количество уже переданных байт (вызов send не блокирует управления). На самом деле функция send игнорирует этот флаг!
Флаг MSG_OOB предназначен для передачи и приема срочных (Out Of Band) данных. Срочные данные не имеют преимущества перед другими при пересылке по сети, а всего лишь позволяют оторвать клиента от нормальной обработки потока обычных данных и сообщить ему "срочную" информацию. Если данные передавались функцией send с установленным флагом MSG_OOB, для их чтения флаг MSG_OOB функции recv также должен быть установлен.
Дейтаграммный сокет также может пользоваться функциями send и recv, если предварительно вызовет connect
но у него есть и свои, "персональные", функции: *"int **sendto** (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen)"* и *"int **recvfrom** (SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen)"*.
Они очень похожи на send и recv - разница лишь в том, что sendto и recvfrom требуют явного указания адреса узла, принимаемого или передаваемого данные. Вызов recvfrom не требует предварительного задания адреса передающего узла - функция принимает все пакеты, приходящие на заданный UDP-порт со всех IP-адресов и портов. Напротив, отвечать отправителю следует на тот же самый порт откуда пришло сообщение. Поскольку функция recvfrom заносит IP-адрес и номер порта клиента после получения от него сообщения, программисту фактически ничего не нужно делать - только передать sendto тот же самый указатель на структуру sockaddr, который был ранее передан функции recvfrem, получившей сообщение от клиента.
Во всем остальном обе пары функций полностью идентичны и работают с теми самыми флагами - MSG_PEEK и MSG_OOB.
Все четыре функции при возникновении ошибки возвращают значение SOCKET_ERROR (== -1).

```
// пример простого UDP-клиента
#include <stdio.h>
#include <string.h>
#include <winsock2.h>
#include <windows.h>

#pragma comment(lib, "Ws2_32.lib")

#define PORT 777
#define SERVERADDR "127.0.0.1"

int main(int argc, char* argv[])
{
    char buff[10 * 1014];
    printf("UDP DEMO Client\nType quit to quit\n");

    // Шаг 1 - инициализация библиотеки Winsocks
```

```
    if (WSAStartup(0x202, (WSADATA *)&buff[0]))
    {
        printf("WSAStartup error: %d\n", WSAGetLastError());
        return -1;
    }

    // Шаг 2 - открытие сокета
    SOCKET my_sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (my_sock == INVALID_SOCKET)
    {
        printf("socket() error: %d\n", WSAGetLastError());
        WSACleanup();
        return -1;
    }

    // Шаг 3 - обмен сообщений с сервером
    HOSTENT *hst;
    sockaddr_in dest_addr;

    dest_addr.sin_family = AF_INET;
    dest_addr.sin_port = htons(PORT);

    // определение IP-адреса узла
    if (inet_addr(SERVERADDR) != INADDR_NONE)
        dest_addr.sin_addr.s_addr = inet_addr(SERVERADDR);
    else
    {
        if (hst = gethostbyname(SERVERADDR))
            dest_addr.sin_addr.s_addr = ((unsigned long **) hst-
>h_addr_list)[0][0];
        else
        {
            printf("Unknown host: %d\n", WSAGetLastError());
            closesocket(my_sock);
            WSACleanup();
            return -1;
        }
    }
    while (1)
    {
        // чтение сообщения с клавиатуры
        printf("S<=C:"); fgets(&buff[0], sizeof(buff) - 1, stdin);
        if (!strcmp(&buff[0], "quit\n")) break;

        // Передача сообщений на сервер
        sendto(my_sock, &buff[0], strlen(&buff[0]), 0, \
        (sockaddr *)&dest_addr, sizeof(dest_addr));

        // Прием сообщения с сервера
        sockaddr_in server_addr;
        int server_addr_size = sizeof(server_addr);

        int n = recvfrom(my_sock, &buff[0], sizeof(buff) - 1, 0, \
```

```
        (sockaddr *)&server_addr, &server_addr_size);

        if (n == SOCKET_ERROR)
        {
            printf("recvfrom() error: %d\n", WSAGetLastError());
            closesocket(my_sock);
            WSACleanup();
            return -1;
        }

        buff[n] = 0;

        // Вывод принятого с сервера сообщения на экран
        printf("S=>C:%s", &buff[0]);
    }

    // Шаг последний - выход
    closesocket(my_sock);
    WSACleanup();
    return 0;
}
```

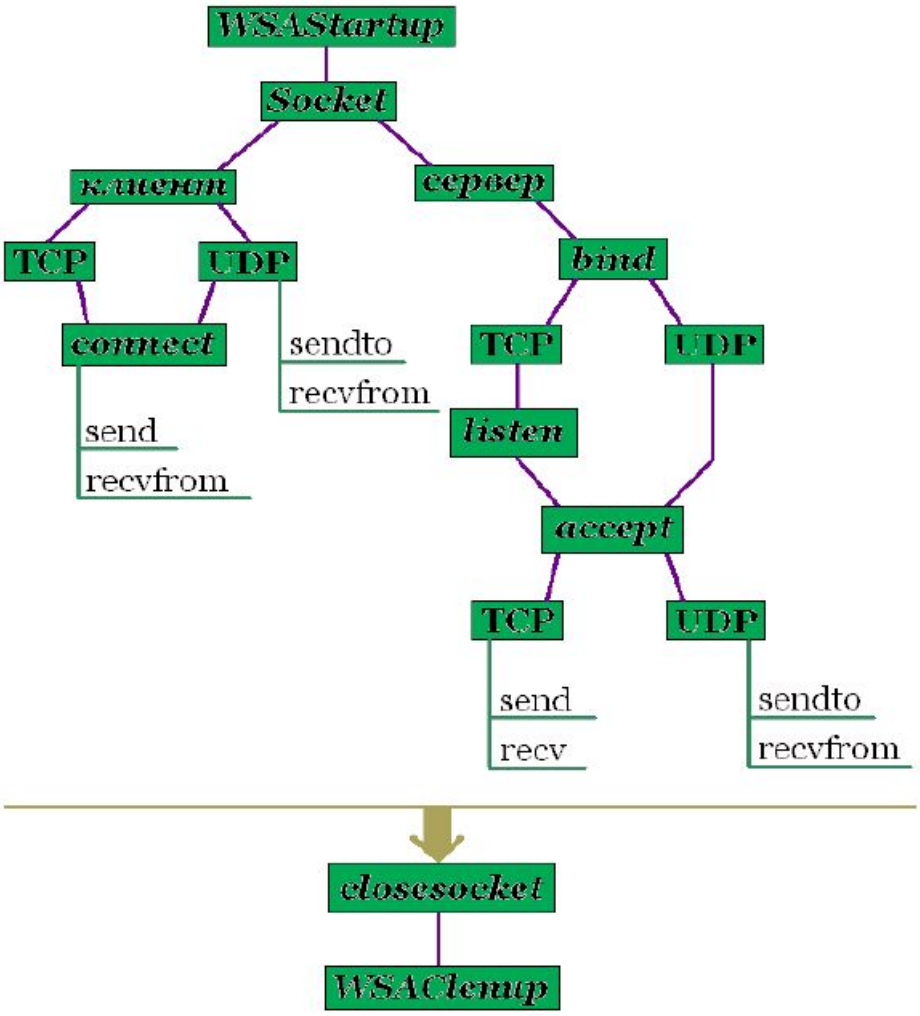
Скопмилировать udp-клиент и запустить. Отправить собранному в примере 3 эхо-серверу сообщение и получить ответ.

Результат выполнения

Задание 1:

Структура приложения-клиента TCP

WSAStartup
socket
connect
send
recv
closesocket
WSACleanup



Изменив исходные коды UDP клиента и примеров 3 и 4, написать TCP клиент для последующей работы с эхо-сервером. Клиент должен считывать из консоли адрес и порт по которыс сервер принимает соединения (сервер принимает соединения на все свои ip-адреса по порту 777). Отладку клиента производить с приложением sr1.exe.

Исходный код TCP клиента
Результат выполнения

Контрольные вопросы

1. Понятие сокета?
2. Типы сокетов?
3. Создание сокета?
4. Понятие IP адреса?
5. Связывание сокета?
6. Создание канала связи?
7. Передача данных?
8. UDP-клиент?

СПИСОК ЛИТЕРАТУРЫ

1. Ватолин Д. С. Алгоритмы сжатия изображений. Методическое пособие: М.: Издательский отдел факультета Вычислительной Математики и Кибернетики МГУ им. М.В.Ломоносова, 1999 г. — 76 с.
2. Стивенс У., UNIX: Разработка сетевых приложений. - СПб.: Питер, 2004
3. Шмидт Д., Хьюстон С. Программирование сетевых приложений на C++. Том1. — Бином-Пресс, 2003. — С. 304.
4. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
5. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
6. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
7. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.

Отчет по лабораторной работе №11 «Основы С++, работа с сокетами. Часть2.»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение принципов функционирования сетевых соединений, ознакомление с программированием сокетов и потоков.

Задачи работы:
-ознакомление с понятием сетевого соединения и сокетов
-разработка серверного приложения

Краткий конспект теоретической части (ответы на контрольные вопросы)

Понятие сокета_____

Типы сокетов_____

Создание сокетов_____

Понятие IP адреса_____

Связывание сокета_____

Создание канала связи (клиент/сервер)_____

Передача данных _____

Понятие callback-фукнции _____

Понятие процесса и потока

Клиент-сервер UDP/TCP

Текстовый чат

Основы C++, работа с сокетами. Часть2.

Пример 1: Создание канала связи: сервер.

Если вы собираетесь передавать датаграммные сообщения при помощи протокола негарантированной доставки UDP , канал связи не нужен. Сразу после создания сокетов и их инициализации можно приступать к передаче данных. Но для передачи данных с использованием протокола TCP необходимо создать канал связи.
Со стороны клиента связь устанавливается с помощью стандартной функции connect:
*int **connect** (SOCKET s, const struct sockaddr FAR* name, int namelen*

Сервер:
сервер переходит в режим ожидания подключений, вызывая функцию *"int **listen** (SOCKET s, int backlog)"*, где s - дескриптор сокета, а backlog - максимально допустимый размер очереди сообщений.
извлечение запросов на соединение из очереди осуществляется функцией *"SOCKET **accept** (SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen)"*, которая автоматически создает новый сокет, выполняет связывание и возвращает его дескриптор, а в структуру *sockaddr* заносит сведения о подключившемся клиенте (IP-адрес и порт). Если в момент вызова accept очередь пуста, функция не возвращает управление до тех пор, пока с сервером не будет установлено хотя бы одно соединение. В случае возникновения ошибки функция возвращает отрицательное значение.
Для параллельной работы с несколькими клиентами следует сразу же после извлечения запроса из очереди порождать новый поток (процесс), передавая ему дескриптор созданного функцией accept сокета, затем вновь извлекать из очереди очередной запрос и т.д. В противном случае, пока не завершит работу один клиент, север не сможет обслуживать всех остальных.
Сервер:
после того как соединение установлено, потоковые сокеты могут обмениваться с удаленным узлом данными, вызывая функции *"int **send** (SOCKET s, const char FAR * buf, int len,int flags)"* и *"int **recv** (SOCKET s, char FAR* buf, int len, int flags)"* для посылки и приема данных соответственно.
Функция *send* возвращает управление сразу же после ее выполнения, независимо от того, получила ли принимающая сторона наши данные или нет.Функция же *recv* возвращает управление только после того, как получит хотя бы один байт.
Работой обоих функций можно управлять с помощью *флагов*, передаваемых в одной переменной типа int третьим слева аргументом. Эта переменная может принимать одно из двух значений: *MSG_PEEK* и *MSG_OOB*.
Флаг MSG_PEEK заставляет функцию recv просматривать данные вместо их чтения. Просмотр в отличие от чтения не уничтожает просматриваемые данные. Некоторые источники утверждают, что при взведенном флаге MSG_PEEK функция recv не задерживает управления, если в локальном буфере нет данных, доступных для немедленного получения. Это неверно! Аналогично, иногда приходится встречать откровенно ложное утверждение о том, что якобы функция send со взведенным флагом MSG_PEEK возвращает количество уже переданных байт (вызов send не блокирует управления). На самом деле функция send игнорирует этот флаг!
Флаг MSG_OOB предназначен для передачи и приема срочных (Out Of Band) данных. Срочные данные не имеют преимущества перед другими при пересылке по сети, а всего лишь позволяют оторвать клиента от нормальной обработки потока обычных данных и сообщить ему "срочную" информацию. Если данные передавались функцией send с установленным флагом MSG_OOB, для их чтения флаг MSG_OOB функции recv также должен быть установлен.
Дейтаграммный сокет также может пользоваться функциями send и recv, если предварительно вызовет connect
но у него есть и свои, "персональные", функции: *"int **sendto** (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen)"* и *"int **recvfrom** (SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen)"*.
Они очень похожи на send и recv - разница лишь в том, что sendto и recvfrom требуют явного указания адреса узла, принимаемого или передаваемого данные. Вызов recvfrom не требует предварительного задания адреса передающего узла - функция принимает все пакеты, приходящие на заданный UDP-порт со всех IP-адресов и портов. Напротив, отвечать отправителю следует на тот же самый порт откуда пришло сообщение. Поскольку функция recvfrom заносит IP-адрес и номер порта клиента после получения от него сообщения, программисту фактически ничего не нужно делать - только передать sendto тот же самый указатель на структуру sockaddr, который был ранее передан функции recvfrem, получившей сообщение от клиента.
Во всем остальном обе пары функций полностью идентичны и работают с теми самыми флагами - MSG_PEEK и MSG_OOB.
Все четыре функции при возникновении ошибки возвращают значение SOCKET_ERROR (== -1).

```
// Пример простого UDP-эхо сервера
#include <stdio.h>
#include <winsock2.h>
```

```
#include "windows.h"
#include <iostream>
#pragma comment(lib, "Ws2_32.lib")

#define PORT 777 // порт сервера
#define sHELLO "Hello, %s [%s] Sailor\n"

using namespace std;
int main()
{
    char buff[1024];
    printf("UDP DEMO echo-Server\n");
    // Шаг 1 - подключение библиотеки
if (WSAStartup(0x0202,(WSADATA *) &buff[0]))
    {
        printf("WSAStartup error: %d\n", WSAGetLastError());
        return -1;
    }

    // Шаг 2 - создание сокета
    SOCKET my_sock;
    my_sock = socket(AF_INET,SOCK_DGRAM,0);
    if (my_sock == INVALID_SOCKET)
    {
        printf("Socket() error: %d\n", WSAGetLastError());
        WSACleanup();
        return -1;
    }

    // Шаг 3 - связывание сокета с локальным адресом
    sockaddr_in local_addr;
    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = INADDR_ANY;
    local_addr.sin_port = htons(PORT);

    if (bind(my_sock, (sockaddr *)&local_addr,
sizeof(local_addr)))
    {
        printf("bind error: %d\n", WSAGetLastError());
        closesocket(my_sock);
        WSACleanup();
        return -1;
    }

    // Шаг 4 обработка пакетов, присланных клиентами
      while (1)
    {
        sockaddr_in client_addr;
        int client_addr_size = sizeof(client_addr);
        int bsize = recvfrom(my_sock,&buff[0],sizeof(buff)-1,0,
        (sockaddr *)&client_addr, &client_addr_size);
        if (bsize == SOCKET_ERROR)
            printf("recvfrom() error: %d\n", WSAGetLastError());
```

```
        // Определяем IP-адрес клиента и прочие атрибуты
        HOSTENT *hst;
        hst = gethostbyaddr((char *)&client_addr.sin_addr, 4,
AF_INET);
        printf("+%s [%s:%d] new DATAGRAM!\n",
        (hst) ? hst->h_name : "Unknown host",
        inet_ntoa(client_addr.sin_addr),
        ntohs(client_addr.sin_port));

        // добавление завершающего нуля
        buff[bsize] = 0;

        // Вывод на экран
        printf("C=>S:%s\n", &buff[0]);

        // посылка датаграммы клиенту
        sendto(my_sock, &buff[0], bsize, 0,
            (sockaddr *)&client_addr, sizeof(client_addr));
    }

    return 0;
}
```

Скомпилировать udp эхо-сервер и запустить.

Результат выполнения

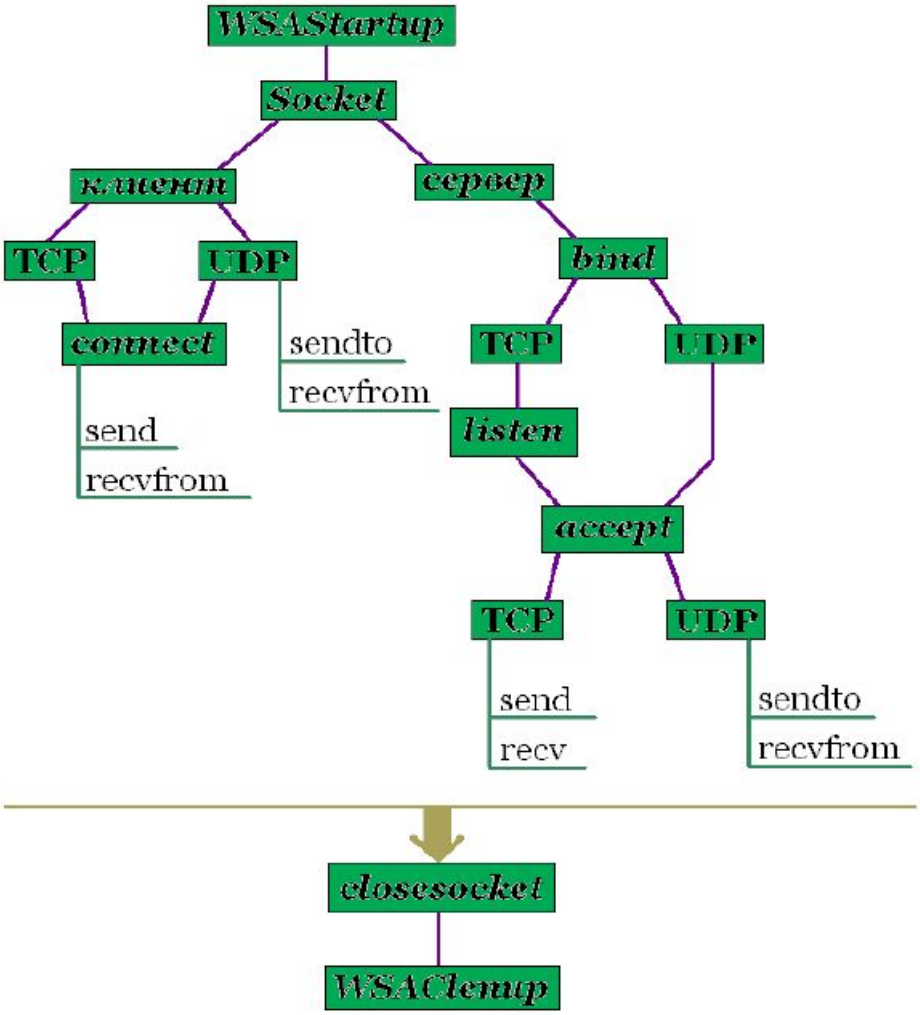
Задание 1:

Структура приложения-сервера TCP

WSAStartup
socket
bind
listen
accept
send
recv
closesocket
closesocket
WSACleanup

Структура приложения-клиента TCP

WSAStartup
socket
connect
send
recv
closesocket
WSACleanup



Изменив исходные коды UDP сервера и примеров 3 и 4, написать TCP эхо сервер. В приложении сервера вынести работу с клиентом в отдельную функцию. Продемонстрировать совместную работу клиента с лабораторной работы №10 и написанного сервера.

Исходный код TCP сервера

Результат выполнения

Контрольные вопросы

1. Понятие сокета?
2. Типы сокетов?
3. Создание сокета?
4. Понятие IP адреса?
5. Связывание сокета?
6. Создание канала связи?
7. Передача данных?
8. UDP эхо-сервер?

СПИСОК ЛИТЕРАТУРЫ

1. Ватолин Д. С. Алгоритмы сжатия изображений. Методическое пособие: М.: Издательский отдел факультета Вычислительной Математики и Кибернетики МГУ им. М.В.Ломоносова, 1999 г. — 76 с.
2. Стивенс У., UNIX: Разработка сетевых приложений. - СПб.: Питер, 2004
3. Шмидт Д., Хьюстон С. Программирование сетевых приложений на C++. Том1. — Бином-Пресс, 2003. — С. 304.
4. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
5. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
6. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
7. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.

Отчет по лабораторной работе №11 «Основы C++, понятие callback функции и потока.»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:
Изучение принципов функционирования сетевых соединений, ознакомление с программированием сокетов и потоков.

Задачи работы:
-ознакомление с понятием сетевого соединения и сокетов
- понятие процесса и потока
-разработка клиент серверного приложение «чат»

Краткий конспект теоретической части (ответы на контрольные вопросы)

Понятие callback-фукнции?__
Создание и вызов callback-функции?
Понятие процесса и потока?
Создание потока?
Клиент-сервер UDP/TCP?
Текстовый чат?

Основы C++, понятие callback функции и потока.

Пример 1: Использование callback-функции

Callback (обратный вызов) — передача исполняемого кода в качестве одного из параметров другого кода. Обратный вызов позволяет в функции исполнять код, который задается в аргументах при её вызове.

```
#include <stdio.h>
#include <iostream>

typedef unsigned char BYTE;

using namespace std;

typedef int (__stdcall *CompareFunction)(const BYTE*, const
BYTE*);

 void  __stdcall  Bubblesort(BYTE* array,
                             int size,
                             int elem_size,
                             CompareFunction cmpFunc);

void __stdcall  Bubblesort(BYTE* array,
                             int size,
                             int elem_size,
                             CompareFunction cmpFunc)
{
   for(int i=0; i < size; i++)
   {
      for(int j=0; j < size-1; j++)
      {
         // make the callback to the comparison function
         if(1 == (*cmpFunc)(array+j*elem_size,
                  array+(j+1)*elem_size))
         {
            // the two compared elements must be interchanged
            int* temp = new int[elem_size];
            memcpy(temp, array+j*elem_size, elem_size);
            memcpy(array+j*elem_size,
                   array+(j+1)*elem_size,
                   elem_size);
            memcpy(array+(j+1)*elem_size, temp, elem_size);
            delete [] temp;
         }
      }
   }
}


int __stdcall   CompareInts(const BYTE* velem1, const BYTE*
velem2)
{
   int elem1 = *(int*)velem1;
```

```
   int elem2 = *(int*)velem2;

   if(elem1 < elem2)
      return -1;
   if(elem1 > elem2)
      return 1;

   return 0;
}

int __stdcall  CompareStrings(const BYTE* velem1, const BYTE*
velem2)
{
   const char* elem1 = (char*)velem1;
   const char* elem2 = (char*)velem2;

   return strcmp(elem1, elem2);
}


int main(int argc, char* argv[])
{
   int i;
   int array[] = {5432, 4321, 3210, 2109, 1098};

   cout << "Before sorting ints with Bubblesort\n";
   for(i=0; i < 5; i++)
      cout << array[i] << '\n';

   Bubblesort((BYTE*)array, 5, sizeof(array[0]), &CompareInts);

   cout << "After the sorting\n";
   for(i=0; i < 5; i++)
      cout << array[i] << '\n';

   const char str[5][10] = {"estella",
                            "danielle",
                            "crissy",
                            "bo",
                            "angie"};

   cout << "Before sorting strings with Bubblesort\n";
   for(i=0; i < 5; i++)
      cout << str[i] << '\n';

   Bubblesort((BYTE*)str, 5, 10, &CompareStrings);

   cout << "After the sorting\n";
   for(i=0; i < 5; i++)
      cout << str[i] << '\n';

   return 0;
}
```

Результат выполнения

Пример 2: Понятие потока:

При запуске программы ОС всегда создает один поток – главный. Именно в нем программа начинает свое выполнение. В зависимости от типа программы и настроек компилятора в главном потоке может выполняться функция main, WinMain, _tmain или функция, заданная пользователем. Главный поток живет до тех пор, пока самая первая функция в стеке не завершила свое выполнение. В то ж время, главная функция может создавать дополнительные потоки, которые будут выполняться одновременно в основным потоком. Для нашей задачи нужно, чтобы главная функция сервера main создавала отдельный поток для работы с каждым подключившимся клиентом.

CreateThread.

HANDLE CreateThread(
LPSECURITY_ATTRIBUTES lpThreadAttributes,
DWORD dwStackSize,
LPTHREAD_START_ROUTINE lpStartAddress,
LPVOID lpParameter,
DWORD dwCreationFlags,
LPDWORD lpThreadId);

Параметры функции CreateThread.
LpThreadAttributes - является указателем на структуру LPSECURITY_ATTRIBUTES. Для присвоения атрибутов защиты по умолчанию, передавайте в этом параметре NULL.
DwStackSize - параметр определяет размер стека, выделяемый для потока из общего адресного пространства процесса. При передаче 0 - размер устанавливается в значение по умолчанию.
LpStartAddress - указатель на адрес входной функции потока.
LpParameter - параметр, который будет передан внутрь функции потока.
DwCreationFlags - принимает одно из двух значений: 0 - исполнение начинается немедленно, или CREATE_SUSPENDED - исполнение приостанавливается до последующих указаний.
LpThreadId - Адрес переменной типа DWORD в который функция возвращает идентификатор, приписанный системой новому потоку.

В случае серверного приложения из задания 1 в callback функцию можно вынести работа сервера с клиентом, после чего вызывать эту функцию в отдельном **потоке** с помощью функции CreateThread.

```
// Пример простого – эхо сервера

#include <stdio.h>
#include <winsock2.h> // Wincosk2.h должен быть раньше windows!
#include <windows.h>

#define MY_PORT 777 // Порт, который слушает сервер

// макрос для печати количества активных пользователей
#define PRINTNUSERS if (nclients) printf("%d user on-
line\n",nclients);else printf("No User on line\n");
```

```
// прототип функции, обслуживающий подключившихся пользователей
DWORD WINAPI WorkWithClient(LPVOID client_socket);

// глобальная переменная – количество активных пользователей
int nclients = 0;
int index=-1; // индекс для массива сокетов
SOCKET sockAr[128];

void sockAr_init()
{
     for(int i=0;i<127;i++)
          sockAr[i]=0;
}

int main(int argc, char* argv[])
{
char buff[1024]; // Буфер для различных нужд
printf("TCP SERVER DEMO\n");

// Шаг 1 - Инициализация Библиотеки Сокетов
// Т.к. возвращенная функцией информация не используется
// ей передается указатель на рабочий буфер, преобразуемый к
указателю
// на структуру WSADATA.
// Такой прием позволяет сэкономить одну переменную, однако,
буфер
// должен быть не менее полкилобайта размером (структура WSADATA
// занимает 400 байт)
if (WSAStartup(0x0202,(WSADATA *) &buff[0]))
{
// Ошибка!
printf("Error WSAStartup %d\n",WSAGetLastError());
return -1;
}

// Шаг 2 - создание сокета
SOCKET mysocket;
// AF_INET - сокет Интернета
// SOCK_STREAM - потоковый сокет (с установкой соединения)
// 0 - по умолчанию выбирается TCP протокол
if ((mysocket=socket(AF_INET,SOCK_STREAM,0))<0)
{
// Ошибка!
printf("Error socket %d\n",WSAGetLastError());
WSACleanup(); // Деинициализация библиотеки Winsock
return -1;
}

// Шаг 3 связывание сокета с локальным адресом
sockaddr_in local_addr;
local_addr.sin_family=AF_INET;
```

```
local_addr.sin_port=htons(MY_PORT); // не забываем о сетевом
порядке!!!
local_addr.sin_addr.s_addr=0; // сервер принимает подключения
// на все свои IP-адреса

// вызываем bind для связывания
if (bind(mysocket,(sockaddr *) &local_addr, sizeof(local_addr)))
{
// Ошибка
printf("Error bind %d\n",WSAGetLastError());
closesocket(mysocket); // закрываем сокет!
WSACleanup();
return -1;
}

// Шаг 4 ожидание подключений
// размер очереди – 0x100
if (listen(mysocket, 0x100))
{
// Ошибка
printf("Error listen %d\n",WSAGetLastError());
closesocket(mysocket);
WSACleanup();
return -1;
}

printf("Waiting for connections... \n");

// Шаг 5 извлекаем сообщение из очереди
SOCKET client_socket; // сокет для клиента
sockaddr_in client_addr; // адрес клиента (заполняется системой)

// функции accept необходимо передать размер структуры
int client_addr_size=sizeof(client_addr);

sockAr_init();

// цикл извлечения запросов на подключение из очереди
while((client_socket=accept(mysocket, (sockaddr *) &client_addr,
&client_addr_size)))
{
nclients++; // увеличиваем счетчик подключившихся клиентов
index++; //увеличиваем индекс в массиве сокетов
sockAr[index] = client_socket;
// пытаемся получить имя хоста
HOSTENT *hst;
hst=gethostbyaddr((char *)
&client_addr.sin_addr.s_addr,4,AF_INET);

// вывод сведений о клиенте
printf("+%s [%s] new connect!\n",
(hst)?hst->h_name:"",inet_ntoa(client_addr.sin_addr));
PRINTNUSERS
```

```
// Вызов нового потока для обслуживания клиента
// Да, для этого рекомендуется использовать _beginthreadex
// но, поскольку никаких вызов функций стандартной Си библиотеки
// поток не делает, можно обойтись и CreateThread
DWORD thID;
CreateThread(NULL,NULL,&WorkWithClient,&client_socket,NULL,&thID
);
}
return 0;
}

// Эта функция создается в отдельном потоке
// и обсуживает очередного подключившегося клиента независимо от
остальных
DWORD WINAPI WorkWithClient(LPVOID client_socket)
{
SOCKET my_sock;
my_sock=((SOCKET *) client_socket)[0];
char buff[20*1024];
#define sHELLO "Hello, Sailor\r\n"
// отправляем клиенту приветствие
send(my_sock,sHELLO,sizeof(sHELLO),0);

int bytes_recv=0;
// цикл эхо-сервера: прием строки от клиента и возвращение ее
клиенту
while( (bytes_recv=recv(my_sock,&buff[0],sizeof(buff),0)) &&
(bytes_recv !=SOCKET_ERROR))
{
send(my_sock,&buff[0],bytes_recv,0);
system("ipconfig >> log.txt");

}
// если мы здесь, то произошел выход из цикла по причине
// возращения функцией recv ошибки - соединение с клиентом
разорвано
nclients--; // уменьшаем счетчик активных клиентов
printf("-disconnect\n"); PRINTNUSERS

// закрываем сокет
closesocket(my_sock);
return 0;
}
```

Собрать модифицированный эхо-сервер и присоединить к нему несколько клиентов.

Результат выполнения

Задание 1:

Реализовать простой консольный чат: один участник пишет – все читают.
(Сервер запоминает соединившихся к нему клиентов и рассылает принятое от одного из клиентов сообщение всем остальным. Для организации работы с клиентами использовать callback функцию).

Исходный код
Результат выполнения

Контрольные вопросы

1. Понятие callback-фукнции?
2. Создание и вызов callback-функции?
3. Понятие процесса и потока?
4. Создание потока?
5. Клиент-сервер UDP/TCP?
6. Текстовый чат?

СПИСОК ЛИТЕРАТУРЫ

1. Ватолин Д. С. Алгоритмы сжатия изображений. Методическое пособие: М.: Издательский отдел факультета Вычислительной Математики и Кибернетики МГУ им. М.В.Ломоносова, 1999 г. — 76 с.
2. Стивенс У., UNIX: Разработка сетевых приложений. - СПб.: Питер, 2004
3. Шмидт Д., Хьюстон С. Программирование сетевых приложений на C++. Том1. — Бином-Пресс, 2003. — С. 304.
4. Дейтел Х. М. Как программировать на C++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.: ил.
5. Страуструп Б. Язык программирования C++. Специальное издание: ++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2008 г. – 1104 с. :ил.
6. Шилдт Г. Полный справочник по C++: ++: Пер. с англ. – М.: Изд-во Вильямс, 2007 г. – 800 с.: ил.
7. Шилдт Г. C++: Базовый курс: ++: Пер. с англ. – М.: Изд-во Вильямс, 2008 г. – 624 с.: ил.