МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Языки программирования

Электронный учебно-методический комплекс
по дисциплине в LMS Moodle

САМАРА
2012

УДК 004
Я 411

Автор-составитель: **Привалов Александр Юрьевич**

Языки программирования [Электронный ресурс] : электрон. учеб.-метод. комплекс по дисциплине в LMS Moodle / Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); авт.-сост. А. Ю. Привалов - Электрон. текстовые и граф. дан. - Самара, 2012. – 1 эл. опт. диск (CD-ROM).

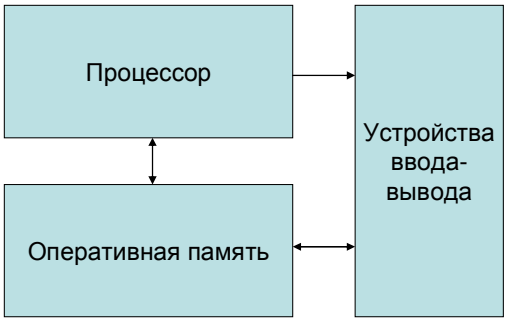
В состав учебно-методического комплекса входят:
1. Раздаточные материалы к лекциям
2. Методические указания к лабораторным занятиям
3. Список тем практических занятий с примерами задач
4. Вопросы для подготовки к экзамену

УМКД «Языки программирования» предназначен для студентов факультета информатики, обучающихся по направлению подготовки бакалавров 010900.62 «Прикладные математика и физика» в 1 и 2 семестрах.

УМКД разработан на кафедре технической кибернетики.

1

Составляющие компьютера



Принципы Фон-Неймана:

- Процессор имеет произвольный доступ к оперативной памяти
- Программа находится в оперативной памяти

2

Элементы языка программирования

- Символы языка (алфавит языка)
- Лексемы (элементарные конструкции)
- Выражения
- Операторы и определения

3

Алфавит языка С

Все видимые символы из набора ASCII, а также пробел, перевод строки и табуляция (пробельные символы).

Коды символов - согласно стандарту ASCII

(American Standard Code for Information Interchange)

4

Лексемы языка С

- Разделители () [] {} . , пробельные символы
- Константы
- Идентификаторы (имена)
- Ключевые (зарезервированные) слова
- Знаки операций

5

Константы

- Целые
- Вещественные
- Символьные
- Строковые

6

Идентификатор

Имя программного объекта, в нём могут использоваться буквы, цифры и _ (подчерк).

Начинаться идентификатор может только с буквы или с подчерка (но не с цифры).

7

Ключевые слова

Это зарезервированные слова, имеющие в языке специальное, заранее установленное значение, и не могущие быть использованными ни в каком другом смысле.

8

Знаки операций в языке С

Это один или более символов, определяющих действие над операндами (операцию) .

По количеству операндов операции бывают

- Унарные
- Бинарные
- Тернарная (одна)

9

Определение переменных

Переменная – именованная область памяти, в которой хранятся данные определённого типа

тип имя;

Типы:

- **int** целый
- **float** вещественный
- **char** символьный
- **char*** строковый

10

Определение именованных констант

#define ИМЯ ЗНАЧЕНИЕ

11

Перечислимый тип

Определение перечислимого типа

enum имя_типа { список_значений };

Определение переменных перечислимого типа

enum имя_типа имя_переменной;

12

Выражения

Выражение – правило вычисления некоторого значения, имеющего определённый тип. Состоит из операндов, знаков операций и скобок

Простейшие операнды:

- константы
- переменные
- вызовы функций

13

Функция

Функция – именованная последовательность определений и операторов, выполняющая какое-либо законченное действие. Для выполнения действия могут быть необходимы **аргументы**. Результатом работы функции является значение определённого типа, называемое **возвращаемым значением**.

14

Вызов функции

имя_функции(значения_аргументов)

15

Арифметические операции

- Присваивание: **=**

операнд1 = операнд2

операнд1 – только переменная

операнд2 – выражение соответствующего типа произвольной сложности

Результат операции – присваиваемое число

Порядок выполнения: **справа налево**.

16

Арифметические операции

+ - * /

Приоритет у * и / выше, чем у + и –

Порядок выполнения – слева направо.

Особенность операции / : если оба операнда целые, то происходит деление нацело. Для целых операндов ещё есть операция % - остаток от деления нацело

17

Арифметические операции

- Специальные операции присваивания

+= -= *= /= %=

Приоритет и порядок выполнения – как у присваивания

- Операции инкремента и декремента

++ --

Приоритет выше, чем у умножения

Есть инфиксная и префиксная формы

18

Оператор вычисления выражения

выражение ;

19

Вывод на экран

printf(аргумент1, аргумент2) ;

аргумент1 – форматная строка

Форматы:

%d – целый

%g – вещественный

%c – символьный

%s – строковый

аргумент2 – выражение соответствующего типа

20

Ввод с клавиатуры

scanf(аргумент1,аргумент2);

аргумент1 – формат ввода

аргумент2 – имя переменной, куда будет введено значение, и перед именем должен стоять **&**

21

Пример программы

```
#include <stdio.h>

void main()
{
int a, b;
printf(“Enter 1-st number:”);  scanf(“%d”,&a);
printf(“Enter 2-nd number:”); scanf(“%d”,&b);
printf(“%d + %d = %d\n”, a, b, a+b);
}
```

22

Операторные скобки

{ }

Любая последовательность операторов и определений, заключённая в операторные скобки, считается одним оператором

23

Комментарии

/* текст комментария */

Любая последовательность символов, заключённая в данные скобки, игнорируется транслятором

// текст комментария

Все символы до конца строки игнорируются транслятором

24

Операции сравнения

> < == != >= <=

Значение операции – **целое** число.

Ноль, если сравнение неверно.

Не ноль, если сравнение верно.

Приоритет ниже, чем у всех арифметических операций.

Порядок выполнения – слева направо.

25

Логические операции

! && ||

Операнды – выражения целого типа

Приоритет – ниже, чем у операций сравнения. Порядок – слева направо.

26

Условный оператор

Полная форма

if (выражение) оператор1

else оператор2

Сокращённая

if (выражение) оператор1

выражение должно быть целого типа

27

Условный оператор

Пример

(ветвление при решении квадратного уравнения):

```
D=b*b-4*a*c;
if (D<0)        { /* выполнится, когда корней нет*/ }
else if (D>0) {  /* выполнится, когда 2 корня */  }
else            {  /* выполнится, когда 1 корень */  }
```

28

Оператор switch

Полная форма:

```
switch(выражение) {
    case константа1 : операторы1
    case константа2 : операторы2
                ...
    case константаN : операторыN
    default : операторы
}
```

выражение целого или символьного типа

константы того же типа, что и выражение

29

Оператор **break**

При использовании внутри оператора switch, прекращает его выполнение и осуществляет переход на следующий оператор.

Пример:

```
        …
   case ‘A’ : /* операторы ветви A */  break;
   case ‘B’ : /* операторы ветви B */
        …
}   /* конец оператора switch */
```

30

Условная операция

(выражение1)? выражение2 : выражение3

выражение1 – целого типа. Если оно не равно нулю, вычисляется значение выражения2, и оно есть значение условного выражения, в противном случае – вычисляется значение выражения3, и оно есть значение условного выражения

31

Оператор цикла с предусловием

while (выражение) оператор

выражение – должно быть целого типа.

Его значение вычисляется, если оно не ноль, выполняется оператор. Потом снова вычисляется выражение и т.д. Когда значение выражения станет равно нулю, выполнение переходит к следующему оператору программы.

32

Оператор цикла с постусловием

do оператор while(выражение);

выражение – должно быть целого типа. Сначала выполняется оператор, потом вычисляется выражение. Если оно не ноль, снова выполняется оператор, и т.д. Когда значение выражения станет равно нулю, выполнение переходит к следующему оператору программы.

33

Оператор цикла for

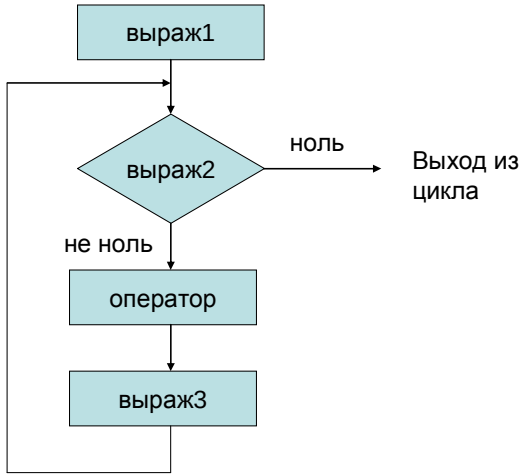
for (выраж1; выраж2; выраж3) оператор

выраж2 - должно быть целого типа.

выраж1 – инициализация цикла,
выраж2 – условие продолжения цикла
выраж3 – итератор цикла

34

Оператор цикла for



35

Вспомогательные операторы в циклах

break; – немедленный выход из цикла (самого внутреннего из вложенных)

continue; – немедленное прекращение текущей итерации цикла и начало следующей

36

Оператор безусловного перехода

goto имя_метки;

Точка перехода:

имя_метки:

37

Основная гипотеза теории алгоритмов

Любой алгоритм может быть построен из более простых с помощью трёх приёмов:

- Последовательное выполнение
- Выполнение одного из вариантов в зависимости от условий
- Многократное повторение до достижения условия

38

Определение функций

```
тип имя(тип_арг1 имя_арг1, ... )
{
тело функции
}
```

В теле функции должен присутствовать оператор

```
return выражение;
```

тип выражения должен совпадать с типом функции

39

Функция main()

```
void main()
{
...
}
```

или

```
int main()
{
...
}
```

Выполнение программы – это вызов функции main(), который выполняет операционная система. В явном виде main() обычно не вызывается.

40

Передача параметров по значению

Пример:

Определение функции

```
void func(int a, float b)
{
int c;
…
}
```

Вызов в программе

```
func(5, 2.5);
```

Переменные, создаваемые внутри функции:

```
int c, a=5;
float b=2.5;
```

41

Объявление функций

тип имя(тип_арг1, ...);

42

Заголовочные файлы стандартной библотеки

Директива включения файла:

#include <имя_файла>

#include “имя_файла”

Заголовочные файлы:

math.h – математические функции

ctype.h – работа с символами

string.h – работа со строками

stdlib.h – функции общего назначения

stdio.h – функции ввода-вывода

43

Области видимости переменных и функций

Классификация переменных по области видимости:

- Локальные
- Локальные статические
- Глобальные
- Внешние глобальные

44

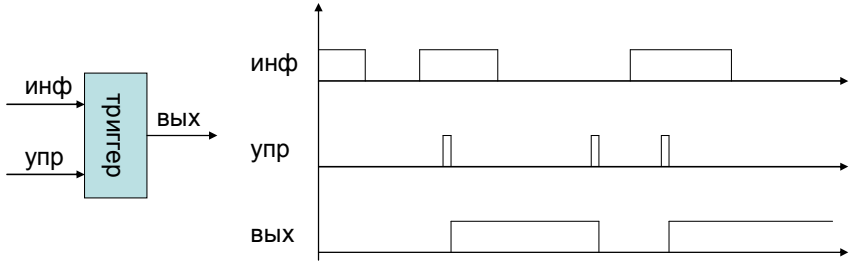
Области видимости переменных и функций

Классификация функций по области видимости:

- Обычные
- Статические

4
5

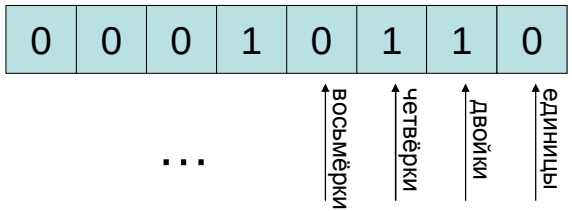
Организация оперативной памяти компьютера



Триггер – однобитовая ячейка памяти. Они группируются в байты, по 8 штук в каждом. Каждый байт имеет свой уникальный номер, называемый адресом.

4
6

Двоичная система счисления



47

Двоичное дополнительное кодирование

Чтобы получить двоичный дополнительный код, надо инвертировать двоичное число и прибавить единицу:

Пример: исходное $00000101_2 = 5_{10}$

инверсия 11111010_2

+1 $11111011_2 = -5_{10}$

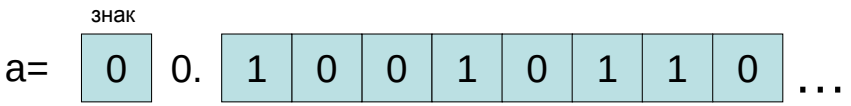
48

Представление вещественных чисел

Нормализованный вид: $X=a*2^b$

a – мантисса ($1/2 \le a < 1$),

b – порядок (показатель степени)



При нормализации сдвиг мантиссы вправо или влево компенсируют увеличением или уменьшением порядка

49

Символьный тип

char

Размер – 1 байт.

Интерпретация – 8-ми битное целое число со знаком (диапазон от -128 до 127) – код символа в таблице ASCII

unsigned char - беззнаковый символьный тип (диапазон от 0 до 255)

50

Целый тип (старый стандарт)

int – Размер 2 байта, диапазон

от $-2^{15}-1=32718$ до $2^{15}-1=32717$

unsigned int – от 0 до $2^{16}-1=65435$

long int – не меньше, чем **int** (как правило, 4 байта)

short int – не больше, чем **int** (как правило, 2 байта)

51

Целый тип (новый стандарт)

long int – 4 байта

short int – 2 байта

int – не меньше **short** и не больше **long**, как правило, 4 байта.

52

Вещественный тип

float – вещественный, размер 4 байта, из них 3 байта мантисса, 1 байт порядок, диапазон примерно от 10^{-38} до 10^{38}

double – вещественный двойной точности, 8 байт, из них 6 - мантисса, 2 – порядок, примерно от 10^{-308} до 10^{308}

long double – 10 байт, из них 7 мантисса, 3 порядок

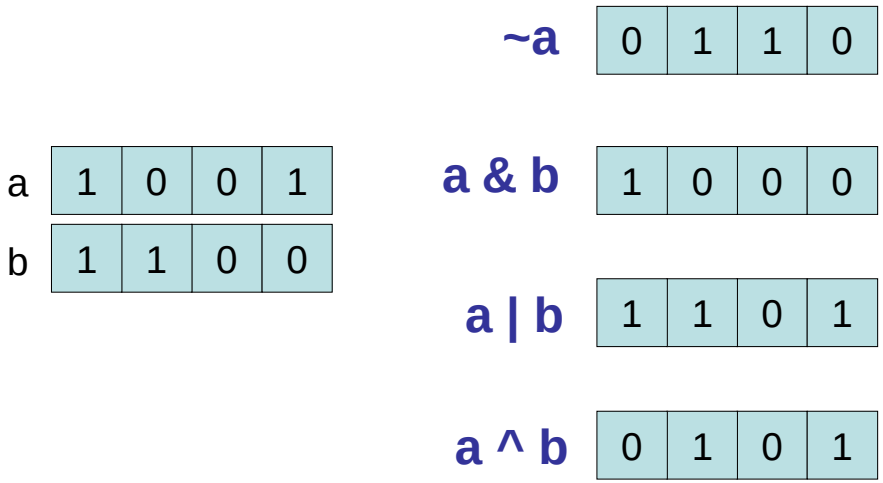
53

Преобразование типов

- Автоматическое – тип с меньшим диапазоном автоматически преобразуется в тип с большим
- Явное – **(тип)имя** – ответственность за последствия возлагается на программиста

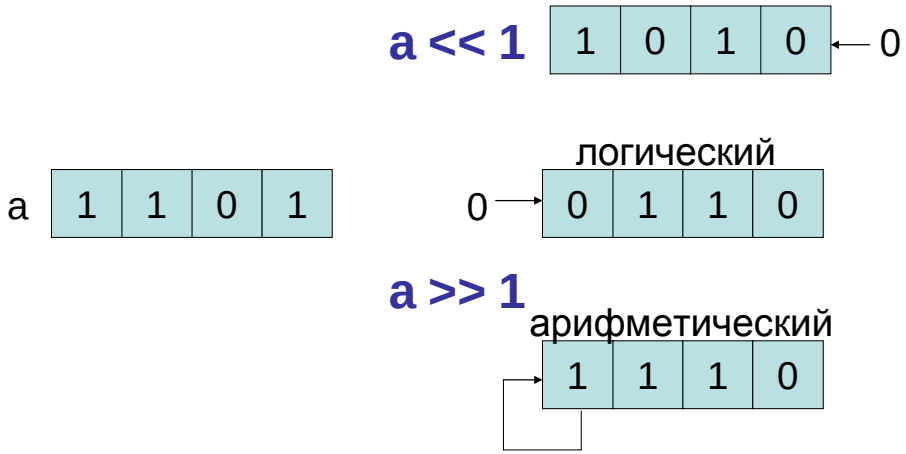
54

Поразрядные логические операции



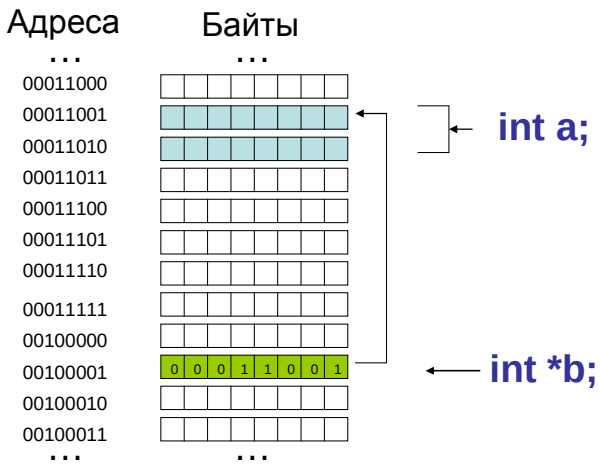
55

Операции сдвига



56

Указатели



57

Операция определения адреса

&имя_переменной

Пример: int a,*b;
b=&a;

58

Операция определения размера

sizeof(объект) – результатом является размер объекта в байтах. В качестве объекта может выступать имя переменной любого типа, а также имя любого типа. Пример: sizeof(int) – размер целого типа int.

59

Массивы

Массив – фиксированное количество элементов данных одного типа, объединённых одним именем, где каждый элемент имеет свой номер. Обращение к элементам массива осуществляется указанием имени массива и номера элемента в массиве

60

Массивы

Определение:

тип имя[размер];

Обращение к элементу:

имя[индекс];

В языке С нумерация элементов массива начинается с 0 !

61

Связь массивов с указателями

Имя массива – это указатель на его первый элемент (элемент с индексом 0).

Пример: **int a[10], *b;**

b=a; // b показывает на a[0]

b+=3; // b показывает на a[3]

62

Многомерные массивы

тип имя[размер1][размер2] …;

В памяти массивы хранятся так, что при движении по памяти быстрее всего меняется самый правый индекс (в частности, двумерные массивы хранятся по строкам).

63

Строки и массивы символов

Строковый тип данных **char*** - это указатель на начало строки.

Конец строки определяется специальным символом **‘\0’**.

64

Структура (структурный тип)

Структура – составной тип данных, содержащий набор элементов разных типов.

Определение структурного типа:

```
struct имя_структуры {
    тип1 имя_поля1;
    тип2 имя_поля2;
    ...
};
```

Определение переменных структурного типа:

```
struct имя_структуры имя_переменной;
```

Обращение к полю:

```
имя_переменной.имя_поля
```

Обращение к полю структуры по указателю

Операция **->**

Пример:

```
struct test {
    int x;
    float y;
} a, *p;

p=&a;
p->x=5;
p->y= 5*p->x;
```


Объединение

Объединение – это переменная, которая может содержать объекты разных типов.

Определение типа объединения:

```
union имя_объединения {
    тип1 имя_поля1;
    тип2 имя_поля2;
    ...
};
```

Определение переменных типа объединения:

```
union имя_объединения имя_переменной;
```

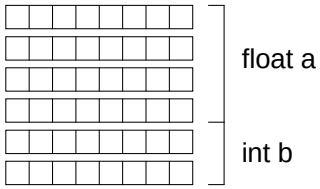
Обращение к полю-варианту:

```
имя_переменной.имя_поля
```

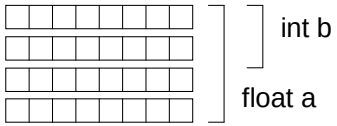
67

Выделение памяти для **struct** и **union**

```
struct {
   float a;
   int b;
};
```




```
union {
   float a;
   int b;
};
```



68

Переименование типов

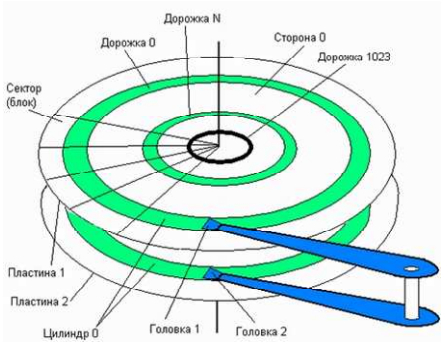
typedef старый_тип новый_тип;

Пример:

typedef char* String;

69

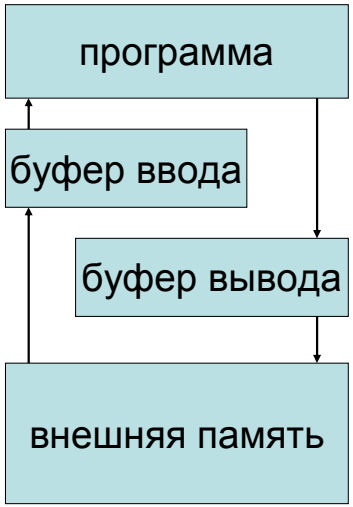
Устройство внешней памяти



- Минимальное количество считанной или записанной информации – 1 блок (несколько килобайт).
- Время доступа в тысячи раз больше, чем у оперативной памяти.

70

Буферизация ввода-вывода



- Программа работает с буферами в оперативной памяти
- Когда буфер заполнится (опустеет), происходит его запись (чтение) на диск (с диска)

71

Файловый ввод-вывод

Файл – это упорядоченная совокупность произвольного числа данных, имеющая имя, называемое **физическим именем файла**

Для работы с файлом в программе ему присваивается **логическое имя.**

Для доступа к данным файла существует **указатель файла**, который указывает на тот элемент данных, с которым будет производится работа при следующем обращении к файлу.

72

Файловый ввод-вывод

Функции **<stdio.h>** для работы с файлами:

FILE* fopen(char* filename, char* type)

int fclose(FILE* file)

void rewind(FILE* file)

73

Виды ввода-вывода

- Бесформатный (блочный)

```
int fread(char* p, int size, int nitems, FILE* f)
int fwrite(char* p, int size, int nitems, FILE* f)
```

- Символьный

```
int fgetc(FILE* f)          int getc(FILE* f)
int fputc(int c, FILE* f)   int putc(int c, FILE* f)
```

- Строковый

```
int fgets(char* str, int n, FILE* f)
int fputs(char* str, FILE* f)
```

- Форматный (см. далее)

74

Форматный вывод

Функции форматного вывода:

`int printf(char* format, …)` - на экран

`int fprintf(FILE* f, char* format, …)` – в файл

`int sprintf(char* buffer, char* format, …)` – в строку

75

Форматы вывода

- Общая спецификация:

  %[выравн][ширина][.точность][доп_пр]символ

выравн – выравнвание – по умолчанию вправо, если стоит – , то влево

ширина – минимальное число выводимых символов. Если слишком маленькая – игнорируется, и выводится сколько надо. Здесь может стоять *

Значения других полей зависят от типа выводимых величин.

76

Форматы вывода

Вывод символа: **%[-][ширина]c**

Вывод строки: **%[-][ширина][.точн]s**

точн – число выводимых символов. Если отсутствует – печатается вся строка. Если есть, а строка длиннее, остаток не печатается.

Форматы вывода

Вывод целого со знаком:

%[-][+][ширина][l]d

После выравнивания стоит поле управления выводом знака положительных чисел. Если оно отсутствует – у положительных чисел знак не выводится, если там + - у них выводится +, а если пробел – то вместо знака у положительных чисел выводится пробел.

l – необходимо указать при выводе данных типа long int.

Форматы вывода

Вывод целого без знака:

%[-][#][ширина][l]u – в десятичном

%[-][#][ширина][l]o – в восьмеричном

%[-][#][ширина][l]x

%[-][#][ширина][l]X – в шестнадцатиричном

x – используются строчные a,b,c,d,f,

X – используются заглавные a,b,c,d,f

- используется только при восми и шестнадцатиричном выводе. Если присутствует, для восьмиричных выводится ведущий 0, а для шестнадчатириных 0x (0X)

79

Форматы вывода

Вывод вещественного числа

%[-][+][#][ширина][.точность][l]f – без экспоненты (с фиксированной точкой)

%[-][+][#][ширина][.точность][l]e

%[-][+][#][ширина][.точность][l]E – в экспоненциальном виде

%[-][+][#][ширина][.точность][l]g

%[-][+][#][ширина][.точность][l]G – наиболее короткий вариант из **f** и **e** (или **E**)

- точка и дробная часть печатаются, даже если дробная часть равна 0.

Точность по умолчанию 6 – для **f,e,E** – это кол-во цифр после точки, для **g,G** – кол-во значащих цифр.

l ставится при выводе **long double**.

80

Форматы вывода

Вывод указателя:

%p

Вывод знака процентов:

%%

Форматный ввод

Функции форматного ввода:

int scanf(char* format, …) - с клавиатуры

int fscanf(FILE* f, char* format, …) – из файла

int sscanf(char* buffer, char* format, …) – из строки

Форматы ввода

- Общая спецификация:

%[*][ширина][доп_пр]символ

*** -** обозначает пропуск данного поля при вводе – ввод по данной спецификации производится, но введённое значение ничему не присваивается;

ширина – максимальное кол-во символов, вводимых по данной спецификации.

83

Форматы ввода

Ввод символа или массива символов:

%[*][ширина]c

ширина – число символов, прочитанных из потока ввода в символьный массив. Вводимы символы могут быть пробельными. При отсутствии поля ширина вводится один символ.

84

Форматы ввода

Ввод строки:

%[*][ширина]s

ширина – максимальная длина вводимой строки. Окончанием строки считается пробельный символ (он не вводится).

85

Форматы ввода

Ввод целого числа:

%[*][ширина][доп]d – в десятичном со знаком

%[*][ширина][доп]u – в десятичном без знака

%[*][ширина][доп]o – в восьмеричном

%[*][ширина][доп]x – в шестнадцатиричном

доп – либо **l** – для long, либо **h** – для short

86

Форматы ввода

Ввод вещественного числа:

%[*][ширина][доп]f

%[*][ширина][доп]e

%[*][ширина][доп]g

доп – либо **l** – для double, либо **L** – для long double

Форматы ввода

Ввод по образцу:

%[*][ширина]образец

образец – множество символов, заключённых в квадратные скобки. В символьный массив из потока ввода вводятся только символы, присутствующие в образце. Если образец начинается с символа **^** - то вводятся не присутствующие в образце. Идущие подряд в коде ASCII символы можно заменять тире. Пример: [abcd] тоже, что [a-d]

[^01234] тоже, что [^0-4]

Препроцессор языка С

Директива включения файла:

#include <файл>

#include “файл”

Директива определения макроподстановки:

#define имя(имя_пар1, …) текст_с_парам

89

Препроцессор языка С

Достоинства макроподстановок:

- Независимость от типов аргументов
- Быстрее, чем вызов функции

Недостатки макроподстановок:

- Увеличивают размер текста программы для компилятора
- **Побочные эффекты !**

90

Препроцессор языка С

Директивы условной компиляции

```
#if константное_выр
…
#else
…
#endif
```

а также

```
#ifdef имя
#ifndef имя
#undef имя
```

91

Алгоритмы и их характеристики

Алгоритм – описание необходимых для решения задачи действий, приведённое в заданной форме.

Временная эффективность (быстродействие) алгоритма – зависимость времени его работы от размера задачи

92

Алгоритмы сортировки

Сортировка – это упорядочивание данных по определённому признаку. Служит для ускорения поиска данных (по этому признаку).

Далее при рассмотрении различных алгоритмов для простоты сортируется массив целых чисел по возрастанию их значений.

93

Пузырьковая сортировка

Описание простейшего варианта алгоритма:

Шаг 1: начиная с первого элемента и до последнего сравниваются между собой пары соседних элементов, и если они стоят не в том порядке, то меняются местами.

Шаг 2: тоже самое, но не до последнего, а до предпоследнего элемента

…

Шаг N-1: просматриваем одну пару – первый и второй элементы.

94

Пузырьковая сортировка

Пример простейшей реализации:

```
for( i=N-1; i>0; i--)
    for( j=0; j<i; j++)
        if( a[ j ]>a[ j+1] ) {
            h=a[ j ]; a[ j ]=a[ j+1]; a[j+1]=h;
            }
```

Быстродействие:

сравнений $N(N-1)/2 = O(N^2)$

95

Сортировка слиянием

Предварительная задача: пусть имеются два уже отсортированных подмассива, надо слить их в один массив, тоже отсортированный.

Алгоритм решения: сравниваем первые элементы подмассивов. Меньший делаем первым элементом результата, а следующий за ним элемент берём для следующего сравнения. Продолжаем так, пока один из подмассивов не закончится. Тогда остаток другого приписываем сзади к результату.

96

Сортировка слиянием

```
void join( int a[], int b, int m, int e)
{
int h[ N ], i, j, k;
i=b; j=m+1; k=0;
while( i<=m && j<=e )
     if( a[ i ]< a[ j ] ) h[ k++ ] = a[ i++ ];
     else h[ k++ ] = a[ j++ ];
while ( i<=m ) h[ k++] = a[ i++ ];
while ( j<=e  ) h[ k++] = a[ j++ ];
for( i=0; i<k; i++ ) a[ b+i ] = h[ i ];
}
```

97

Сортировка слиянием

Алгоритм: исходный массив делим пополам, потом каждую часть ещё пополам, и так до тех пор, пока размер кусочка не станет 2 или 1. После этого сливаем отсортированные кусочки (на обратном ходу рекурсии), пока не получим исходный отсортированный массив.

98

Сортировка слиянием

```
void sortjoin( int a[], int b, int e)
{
int h;
if( b<e )                          // в массиве больше одного эл-та
   if( e – b == 1 )  {             // в массиве ровно 2 элемента
      if( a[b]>a[e] ) { h=a[b]; a[b]=a[e]; a[e]=h; }
      }
   else {
      sortjoin(a, b, (b+e)/2 );
      sortjoin(a, (b+e)/2 +1, e );
      join(a, b, (b+e)/2, e );
      }
}
```

Быстродействие: N сравнений на каждом из $\log_2 N$ уровней разбиения = $O(N \log_2 N)$

99

Быстрая (барьерная) сортировка

Выберем какой-либо элемент массива, и переставим остальные элементы массива так, чтобы слева от выбранного все были меньше него, а справа больше. Потом в обеих частях (где собраны меньшие, и большие) проделаем ту же процедуру, и т.д. Пока не дойдём до кусочков размером 1.

100

Быстрая (барьерная) сортировка

```
void quicksort(int a[], int b, int e)
{
int i, j, x, h;
i=b; j=e; x=a[(b+e)/2];
do {
    while( a[i]<x && i<e ) i++;
    while( a[j]>x && j>b ) j--;
    if( i<=j ) { h=a[i]; a[i]=a[j]; a[j]=h; i++; j--; }
} while( i<=j );
if( b<j ) quicksort(a, b, j);
if( i<e ) quicksort(a, i, e);
}
```

Быстродействие:N сравнений на каждом из $\log_2 N$ уровней разбиения (если барьер – примерно среднее значение в сортируемом куске) = $O(N \log_2 N)$

Специальные случаи сортировки

Задача: отсортировать массив целых чисел при условии, что каждое число может встретиться в массиве не более одного раза.

Алгоритм (без единого сравнения): пусть целое число занимает 2 байта, и его максимальное значение есть 65435. Создадим символьный массив размером 65435, заполненный нулями (назовём его таблицей). Просмотрим сортируемый массив, и встречая там какое-либо число, соответствующий элемент таблицы устанавливаем в 1. Потом, просматривая таблицу от младших индексов к старшим, формируем отсортированный массив.

1

Структуры данных

Структура данных – это совокупность определённым образом организованных данных, относящихся к одной задаче и отражающих её специфику. Структура данных подразумевает соответствующие ей методы добавления, удаления и доступа к данным.

2

Таблица

Таблица – это массив данных, упорядоченный по значению элемента данных, называемого ключом. Каждому значению ключа соответствует своё место в таблице.

3

Таблица

Пример: перекодировка русского текста из DOS (кодовая таблица 866) в Windows (кодовая таблица 1251).

Таблица 866

А	128	е	197
В	129	ж	198
…	…	…	…
Е	133	п	175
Ж	134	р	224
…	…	…	…
Я	159	я	239
а	160	Ё	240
…	…	ё	241

Таблица 1251

Ё	168
ё	184
А	192
…	…
Я	223
а	224
…	…
я	255

4

Таблица (пример)

Символьный массив DosToWin (таблица перекодировки)

Индекс	128	…	159	160	…	175		224	…	239	240	241
Значение	192	…	223	224	…	239		240	…	255	168	184
Коммен-тарий	А	…	Я	а	…	п		р	…	я	Ё	ё

Реализация перекодировки:

```
unsigned char c, DosToWin[256]={0,1,…};
FILE *infile, *outfile;
…
while( (c=fgetc(infile)) != EOF ) fputc(DosToWin[c], outfile);
…
```

5

Стек

Стек – упорядоченный набор данных, добавление в который и удаление из которого производятся с одного конца, называемого вершиной стека.

Стек реализует дисциплину LIFO – Last In First Out (последний пришёл, первый ушёл).

6

Стек (пример реализации)

```
char stack[ST_SIZE];
int tos=0;

int push(char c)       //  поместить данные в стек
{
if( tos == ST_SIZE-1 ) return 0;
else { stack[tos++]=c;  return 1; }
}

int pop(char *c)      //  забрать данные из стека
{
if( tos == 0 ) return 0;
else { *c=stack[--tos]; return 1; }
}

int is_stack_empty() { return !tos; }     // проверить стек на пустоту
```

7

Очередь

Очередь – упорядоченный набор данных, добавление элементов в который производится с одного конца (называемого хвостом), а удаление – с другого (называемого головой).

Очередь реализует дисциплину FIFO – First In First Out (первый пришёл, первый ушёл).

8

Очередь (пример реализации)

```
char queue[Q_SIZE];
int head=0, tail=0, qfull=0;

int enter(char c)
{
if( qfull=1 ) return 0;                    // очередь полна
else {
     queue[ tail++]=c;
     if( (tail%=Q_SIZE) == head ) qfull=1;
     }
return 1;
}

int leave(char* c)
{
if( head == tail && qfull=0 ) return 0;       // очередь пуста
else {  *c=queue[ head++ ];  head%=Q_SIZE; }
qfull=0;
return 1;
}
```

9

Динамическая память

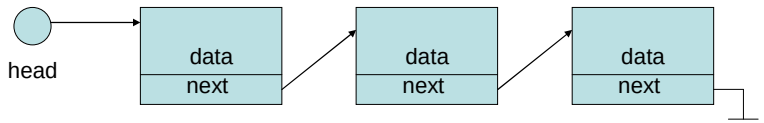
Заголовочный файл stdlib.h

Функции:

```
void* malloc( size_t size );
void* calloc( size_t num, size_t size );
void* realloc(void *block, size_t size);
void free(void *block);
```

10

Однонаправленный список




```
struct node {
   int data;
   struct node* next;
   };
struct node* head=NULL;
```

11

Однонаправленный список

Вывод списка:

```
void view1(struct node* head)        // нерекурсивная версия
{
while( head != NULL ) {
   printf("%d\n", head->data);
   head=head->next;
   }
}

void view2(struct node* head)       // рекурсивная версия
{
if( head != NULL ) {
     printf("%d\n", head->data);
     view2(head->next);
    }
}
```

12

Однонаправленный список

Вставка:

```
struct node** insert( int data, struct node** place)
{
struct node* tmp;
if( place != NULL ) {
     tmp=(struct node*)malloc(sizeof(node));
     tmp->data=data; tmp->next=*place;
     *place=tmp;
     return place;
    }
return NULL;
}
```

13

Однонаправленный список

Удаление:

```
int delete( struct node** delnode)
{
struct node* tmp;
if( delnode != NULL ) {
    tmp=*delnode;
    *delnode=(*delnode)->next;
    free(tmp);
    return 1;
    }
return 0;
}
```

14

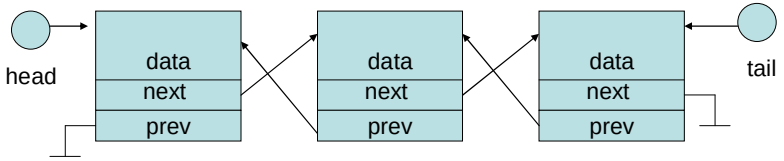
Однонаправленный список

Поиск:

```
struct node** find(int data, struct node** head)
{
struct node** tmp;
tmp=head;
while( *tmp != NULL && data != (*tmp)->data )
    tmp=&( (*tmp)->next );
if( data == (*tmp)->data ) return tmp;
else return NULL;
}
```

15

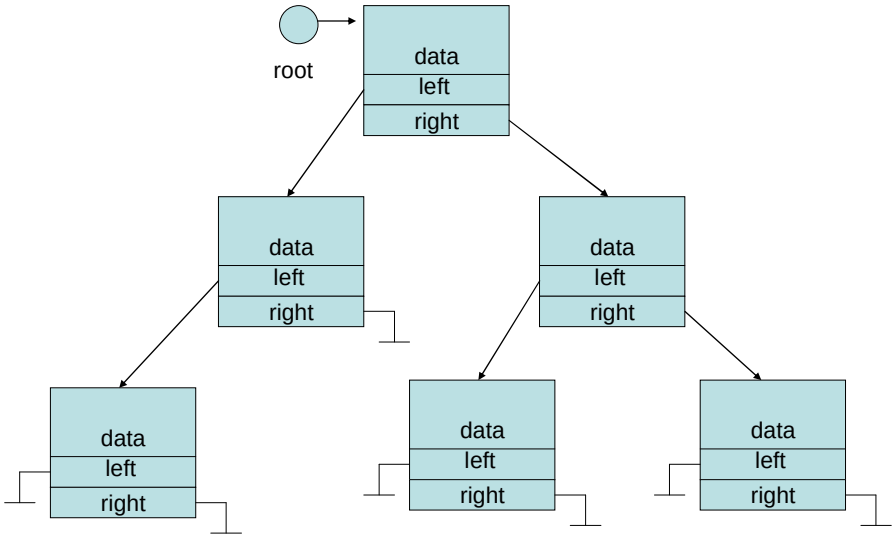
Двунаправленный список




```
struct node {
   int data;
   struct node *next, *prev;
  };
struct node *head=NULL, *tail=NULL;
```

16

Двоичное дерево



17

Двоичное дерево

```
struct node {
    int data;
    struct node *left, *right;
    };
struct node *root=NULL;
void view_tree(struct node *root)
{
if( root != NULL ) {
    printf("%d\n", root->data);
    view_tree(root->left);
    view_tree(root->right);
    }
}
```

18

Двоичное дерево поиска

Двоичное дерево поиска – это двоичное дерево, каждый узел которого обладает следующим свойством: его информационное поле больше, чем информационное поле любого его потомка из левого поддерева и меньше или равно, чем информационное поле любого его потомка из правого поддерева.

19

Двоичное дерево поиска

Поиск:

```
struct node** find(int data, struct node**root)
{
if ( *root != NULL)
   if ( (*root)->data = = data ) return root;
   else if( data < (*root)->data )
          return find(data, &( (*root)->left) ) );
   else return find(data, &( (*root)->right) ) );
else return NULL;
}
```

20

Двоичное дерево поиска

Вставка:

```
struct node** insert(int data, struct node**root)
{
struct node *tmp;
if ( *root == NULL) {
    tmp=(struct node*)malloc(sizeof(node));
    tmp->data=data; tmp->left=tmp->right=NULL;
    *root=tmp;
    return root;
    }
else if( data < (*root)->data )
       return insert( data, &( (*root)->left  ) ) );
else return insert( data, &( (*root)->right) ) );
}
```

21

Двоичное дерево поиска

Удаление:

```
void delete(struct node** delnode)
{
struct node *tmp, **rtst;
if ( delnode != NULL )
    if ( (*delnode)->right == NULL ) {
        tmp=*delnode; *delnode=(*delnode)->left; free(tmp);
        }
    else if( (*delnode)->left == NULL ) {
        tmp=*delnode; *delnode=(*delnode)->right; free(tmp);
        }
    else {
        rtst=rightest( &((*delnode)->left) );  tmp=*rtst;
        (*delnode)->data=(*rtst)->data; *rtst=(*rtst)->left; free(tmp);
        }
}
```

22

Двоичное дерево поиска

Поиск самого правого узла в поддереве (для функции удаления):

```
struct node** rightest(struct node** subtree)
{
struct node** tmp;
if( subtree != NULL) {
    tmp=subtree;
    while( (*tmp)->right != NULL ) tmp=&( (*tmp)->right)
    }
return tmp;
}
```

23

Куча-пирамида (heap)

Это двоичное дерево, удовлетворяющее двум условиям:

1. Упорядоченность – значение любого узла не превышает значение любого его потомка.
2. Форма – каждый уровень дерева заполняется всегда слева направо.

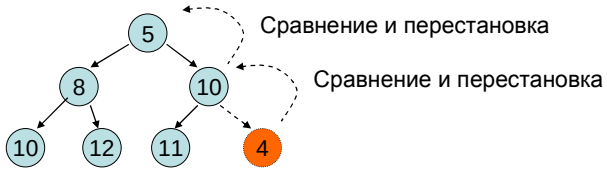
Количество элементов в куче однозначно определяет её форму!

Вспомогательные операции работы с кучей:

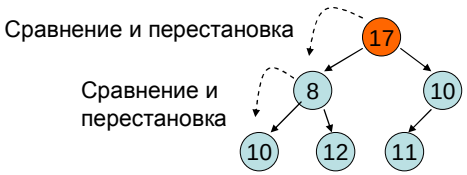
1. Добавление снизу (всплытие)
2. Добавление сверху (погружение).

24

Добавление снизу (всплытие)



Добавление сверху (погружение)



Трудоёмкость обоих операций: $O(\log_2 N)$

25

Пример реализации кучи-пирамиды

int heap[N];

int heapsize=0;

Потомки узла, хранящегося в ячейке i, располагаются в ячейках 2*i+1 (левый) и 2*i+2 (правый)

26

Пример реализации кучи-пирамиды

```
void moveup()
{
int i, p, h;
i=heapsize; p=(i-1)/2;
while( heap[ i ]<heap[ p ] && i >0) {
      h=heap[ i ]; heap[ i ]=heap[ p ]; heap[ p]=h;
      i=p; p=(i-1)/2;
      }
}

void movedown()
{
int i, p, h;
i=0; p=2*i+1;
while( p<=heapsize) {
      if (p+1<=heapsize && heap[p+1] < heap[p] ) p++;
      if( heap[ p ] < heap[ i ] ) {
            h=heap[ i ]; heap[ i ]=heap[ p ]; heap[ p]=h;
            i=p; p=2*i+1;
            }
      else break;
      }
}
```

27

Пример реализации кучи-пирамиды

```
int insert(int data)
{
if( heapsize >= N-1 ) return 0;
heap[heapsize++]=data;
moveup();
return 1;
}

int extractmin(int* data)
{
if( heapsize<1) return 0;
*data=heap[0];
heap[0]=heap[--heapsize];
movedown();
return 1;
}
```

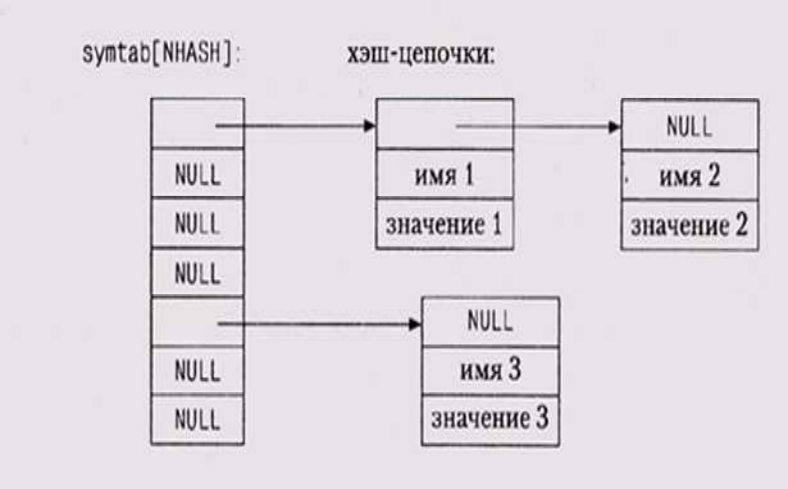
28

Хэш-таблица

Хэш-таблица – это набор данных, доступ к которым осуществляется по сокращённому ключу, получаемому из полного ключа применением специальной функции, называемой хэш-функцией.

29

Хэш-таблица



30

Пример образцовой вычислительной практики (для физиков)

Решение задачи многих тел.

Ссылки:

- Andrew Appel “An Efficient Program for Many-Body Simulation” // SIAM Journal on Scientific and Statistical Computing, January 1985, v.6, № 1, pp. 85-103
- Andrew Appel “An Investigation of Galaxy Clustering Using an Asymptotically Fast N-body Algorithm” // Ph.D. Thesises, Princeton University, April 24, 1981

31

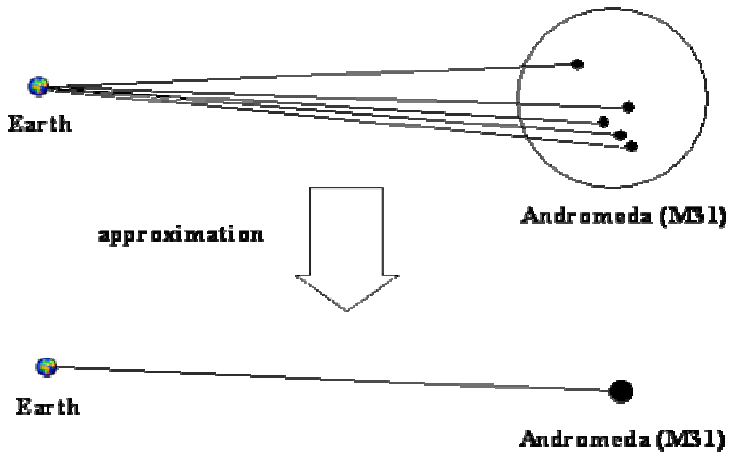
Задача многих тел

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i,$$

$$\frac{d\mathbf{v}_i}{dt} = \sum_{j \neq i}^{N} G\, m_j \, \frac{\mathbf{r}_j - \mathbf{r}_i}{\left|\mathbf{r}_j - \mathbf{r}_i\right|^3}.$$

32

Задача многих тел



33

Повышение эффективности реализации для задачи многих тел

Улучшение	Ускорение	Изменения
Алгоритм и структура данных	12 раз	Двоичное дерево для хранения кластеров
Оптимизация алгоритма	2 раза	Увеличение шага по времени
Реорганизация структуры данных	2 раза	Перестройка кластеров под реальную ситуацию
Системно-независимая оптимизация кода	2 раза	Замена двойной точности на одинарную
Системно-зависимая оптимизация кода	2,5 раза	Кодирование критического участка на ассемблере
Улучшение аппаратуры	2 раза	Использование сопроцессора

≈400 раз

34

Проблемы больших программ

- Ошибки из-за некорректной работы с данными
- Ошибки при повторном использовании кода

35

C++ как расширение C

- Новые типы данных: bool wchar_t
- Новая реализация констант: const
- Второй способ инициализации переменных
  Пример: int a=10; - старый
  int a(10); - новый
- Встроенные функции: inline
- Ссылочный тип: тип& имя;
- Перегрузка функций – функции распознаются по **сигнатурам**
- Аргументы по умолчанию
- Определение переменных сложных типов
- Операции для работы с динамической памятью: new delete
- Явное указание области видимости переменной
- Пространства имён: namespace

36

Класс и объект в C++

Определение класса:

```
class имя_класса {
  закрытая_часть_класса
public:
  открытая_часть_класса
};
```

Определение объектов класса:

```
имя_класса имя_переменной;
```

37

Класс и объект в С++

Пример:

```
class test {
    int a;
public:
    int b;
    void set_a(int num);
    int get_a();
} ob1, ob2;
```

Обращение в программе:

```
ob1.b=10;
ob2.set_a(10);
x=5*ob2.get_a();
```

38

Описание методов вне класса

```
тип имя_класса::имя_метода(параметры)
{
тело_метода
}
```

Пример:

```
void test::set_a(int num)
{
if( num<0 ) a=0;
else a=num;
}
```

39

Пример реализации класса стек

```
class stack {
    char stk[SIZE];
    int tos;
public:
    void init();
    int push(char c);
    int pop(char& c);
};

void stack::init()
{
tos=0;
}
```

```
int stack::push(char c)
{
if ( tos==SIZE ) return 0;
else stk[tos++]=c;
return 1;
}

int stack::pop(char& c)
{
if( tos==0 ) return 0;
else c=stk[--tos];
return 1;
}
```

40

Конструкторы и деструкторы

Конструктор класса – это метод, который автоматически вызывается всякий раз, когда создаётся объект данного класса. В C++ конструктор имеет имя класса и не имеет возвращаемого значения.

Деструктор класса – это метод, который вызывается всякий раз, когда исчезает объект данного класса. В C++ деструктор не имеет возвращаемого значения и параметров, а его имя – это имя класса, перед которым стоит знак ~ (тильда).

41

Конструкторы и деструкторы

Пример - конструктор и деструктор класса stack:

```
class stack{
    char stk[SIZE];
    int tos;
public:
    stack();
   ~stack();
    int push(char);
    int pop(char&);
};
```

```
stack::stack()
{
tos=0;
}

stack::~stack()
{
printf("%d elements left in
    stack", tos);
}
```

42

Конструктор с параметрами

```
class stack{
    char* stk;
    int size;
    int tos;
public:
    stack(int);
   ~stack();
    int push(char);
    int pop(char&);
};
```

```
stack::stack(int s=SIZE)
{
stk=new char[ size=(s>0)?s:1 ];
if( stk == NULL ) size=0;
tos=0;
}
stack::~stack()
{
delete[] stk;
}
```

Передача параметров в конструктор:

```
stack s(50), s2(1000),s3;
```

43

Перегрузка методов класса

При перегрузке методов (в том числе конструкторов), объявление (или определение) каждого варианта должно быть включено в определение класса.

Перегрузка операций в классе

тип& operator<знак>(параметры)

Некоторые операции перегружать нельзя, у других есть ограничения на тип возвращаемого значения и параметры.

44

Пример перегрузки операции =

```
class stack{
    char* stk;
    int size;
    int tos;
public:
    stack(int);
   ~stack();
    int push(char);
    int pop(char&);
    stack& operator=(stack&);
};
```

```
stack& stack::operator=(stack& arg)
{
if( &arg == this ) return *this;
if( stk != NULL ) delete[] stk;
if( arg.stk != NULL ) {
    stk=new char[ size=arg.size ];
    tos=arg.tos;
    for( int i=0; i<tos; i++ )
            stk[ i ] = arg.stk[ i ];
    }
else { stk=NULL; size=0; }
return *this;
}
```

45

Консольный ввод-вывод в C++

```
#include <iostream>
using namespace std;
```

```
cout<<выражение;
cin>>переменная;
```

Перегруженные операции >> и << возвращают cout и cin соответственно, и выполняются слева направо.

46

Файловый ввод-вывод в C++

```
#include <fstream>
using namespace std;
```

ifstream – класс входных файловых потоков

ofstream – класс выходных файловых потоков

fstream – класс двунаправленных файловых потоков

47

Файловый ввод-вывод в C++

Конструкторы с параметрами:
ifstream(const char* name, int mode=ios::in);
ofstream(const char* name, int mode=ios::out|ios::trunc);
fstream(const char* name, int mode=ios::in|ios::out);
Есть конструкторы без параметров.

Методы:
open(…), close(), eof()

Перегруженные операции: >> и << соответственно.

48

Класс string

```
#include <string>
using namespace std;
```

Перегруженные операции:

= + += == != <= >=

Некоторые методы:

find insert remove substr empty

Шаблон vector

```
#include <vector>
using namespace std;
```

Определение класса по шаблону:

```
vector<class> имя(размер);
```

Примеры:

```
vector<int> a;    // пустой вектор целых
vector<stack> s(10);  // начальный размер 10
```


Шаблон vector

Некоторые методы:

size – текущий размер

capacity – текущая ёмкость

resize – изменение размера

reserve – изменение ёмкости

push_back – добавить элемент в конец

pop_back – удалить элемент с конца

51

Инициализация массивов объектов

класс имя[размер]={класс(параметры), класс(параметры), …};

Пример:

stack st[3]={stack(10), stack(20), stack(30)};

Для конструктора с одним параметром есть сокращённая форма:

stack st[3]={10, 20, 30};

52

Конструктор копирования

имя_класса(имя_класса&)

Вызывается в трёх случаях:

- При передаче объекта в функцию в качестве параметра
- При возврате объекта из функции, как возвращаемого значения
- При определении нового объекта, как копии существующего

53

Пример конструктора копирования

```
stack::stack(stack& arg)
{
if(stk != NULL) delete[] stk;
if( arg.stk != NULL ) {
   stk=new char[size=arg.size];
   tos=arg.tos;
   for( int i=0; i<tos; i++ ) stk[ i ] = arg.stk[ i ];
else { stk=NULL; size=0; }
}
```

54

Статические элементы класса

- Определяются с помощью ключевого слова static
- Они общие для всех объектов данного класса
- Создаются при определении класса, и существуют, даже если нет объектов данного класса
- Не учитываются операцией sizeof()
- Обращаться к ним можно и через имя объекта, и через имя класса
- В открытой части класса инициализация статических полей должна быть вне функций (как глобальных переменных)
- В закрытой части класса обращаться к статическим полям могут только статические методы

55

Дружественные функции и дружественные классы

Определяются в классе с помощью ключевого слова friend. Дружественные классу функции, не являющиеся членами класса, тем не менее имеют доступ к закрытой части класса. Все методы дружественного класса имеют доступ к закрытой части класса. **Дружественность не взаимна и не наследуется**.

56

Объекты внутри объектов

Для передачи параметров конструкторам объектов, являющих полями главного объекта, конструктор главного объекта должен быть определён так:

```
имя_класса(параметры_1): имя_поля(параметры_2)
{
…
}
```

где параметры_2 должны быть подмножеством параметры_1.

57

Наследование классов

Объявление одного класса наследником другого:

```
class имя_потомка : спец_наслед имя_предка {
…
};
```

Спецификаторы наследования:

public – открытая часть класса-предка переходит в открытую часть класса-потомка

private – открытая часть класса-предка переходит в закрытую часть класса-потомка.

Закрытая часть класса-предка присутствует в классе-потомке, но определённым в потомке методам не доступна при любом наследовании (она доступна только через унаследованные методы)!

58

Работа конструкторов и деструкторов при наследовании

- Конструкторы срабатывают согласно цепочке наследования – сначала предок, потом потомок, деструкторы – в обратном порядке.
- Передача параметров в конструктор базового класса осуществляется следующим образом:

```
имя_потомка(параметры_1): имя_предка(параметры_2)
{
…
}
```

где параметры_2 должны быть подмножеством параметры_1.

59

Переопределение методов при наследовании

- Если в классе определён метод с той же сигнатурой, что и метод класса-предка, он перекрывает метод класса-предка.
- Из класса- потомка перекрытые методы класса-предка доступны через явное указание имени класса при вызове метода:

имя_класса_предка::имя_метода(парам)

60

Указатели на базовый класс

Указателю на объект базового класса (класса-предка) может быть присвоен адрес объекта любого производного класса (класса-потомка), но обращаться через такой указатель можно только к полям и методам, унаследованным от этого базового класса.

61

Виртуальные методы

Определяются с помощью ключевого слова virtual в базовом классе.

При обращении к виртуальному методу через указатель на базовый класс, происходит определение класса объекта, на который показывает указатель, и вызывается метод этого класса, а не класса-предка, как у обычных (невиртуальных) методов.

62

Пример классов с виртуальными методами

```
class area {
    double dim1, dim2;
public:
    void setarea(double d1, double d2) {dim1=d1; dim2=d2;}
    void getdim(double& d1, double& d2) {d1=dim1; d2=dim2;}
    virtual double getarea() { cout<<“To be defined!”; return 0.;}
};

class rectangle : public area {
public:
    double getarea() { double d1,d2; getdim(d1,d2); return d1*d2; }
};

class triangle : public area {
public:
    double getarea() { double d1,d2; getdim(d1,d2); return d1*d2/2; }
};
```

63

Чистые виртуальные методы и абстрактные классы

Определение чистого виртуального метода в классе:

virtual тип имя_метода(параметры)=0;

Класс, в котором есть хотя бы один чистый виртуальный метод, является абстрактным классом. Нельзя определить объект такого класса, он служит только для наследования, и в производном классе все чистые виртуальные методы должны быть переопределены.

64

Пример абстрактного класса

```
class area {
double dim1, dim2;
public:
    void setarea(double d1, double d2) {dim1=d1; dim2=d2;}
    void getdim(double& d1, double& d2) {d1=dim1; d2=dim2;}
    virtual double getarea()=0;
};
```

Теперь, если мы не переопределим в производных классах метод getarea, и попытаемся создать объекты таких классов, будет ошибка компиляции.

65

Пример иерархии классов

Задача: создать иерархию классов, реализующих следующую ситуацию: есть склад (у склада есть адрес), на котором хранится бытовая техника – телевизоры и холодильники.

У всех предметов бытовой техники есть размеры и марка. Телевизоры могут быть цветными и чёрно-белыми. Холодильники могут быть однокамерными и двухкамерными.

66

Пример иерархии классов

```
class home_device {
      float length, width, height;
      char* type;
public:
      home_device(float l, float w, float h, char* t);
      virtual void save_to_file(ofstream& file);
      virtual void load_from_file(ifstream& file);
};

class TV_set : public home_device {
      enum TV_type { BW, COLOR } feature;
public:
      TV_set(float l, float w, float h, char* t, TW_type f);
      virtual void save_to_file(ofstream& file);
      virtual void load_from_file(ifstream& file);
};

class Fridge: public home_device {
      enum Fridge_type { ONECHAMBER, TWOCHAMBER } feature;
public:
       Fridge(float l, float w, float h, char* t, Fridge_type f);
      virtual void save_to_file(ofstream& file);
      virtual void load_from_file(ifstream& file);
};
```

67

Пример иерархии классов

```
const storage_capacity=100;

class storage {
    char* address;
    home_device* dev[ storage_capacity ];
    int size;
public:
    storage(char* addr);
    add_TV(TV_set&);
    add_Fridge(Fridge&);
    remove_device(int i);
    int find_device(char* type);
    void save_to_file(ofstream& file);
    void load_from_file(ifstream& file);
};
```

68

Интерфейс командной строки

int main(int argc, char* argv[], char** envp)
{…}

argc – количество аргументов (слов) в командной строке, включая название программы

argv – массив строк размером argc+1, где каждая – соответствующий аргумент (слово), начиная с названия программы (argv[0]). Последний элемент массива – нулевой указатель.

envp – массив строк переменных среды операционной системы вида имя=значение. Последний элемент массива – нулевой указатель. Зависит от операционной системы.

возвращаемое целое значение – после завершения программы передаётся в операционную систему. Принято значение 0 считать признаком успешного завершения, а не нулевое число – номером некоторой аварийной ситуации. Зависит от операционной системы

69

Интерфейс командной строки

Пример:

```
int main(int argc, char* argv[], char** envp)
{
cout<<“Аргументы командной строки:”<<endl;
for( int i=0; i<argc; i++) cout<<argv[i]<<endl;
cout<<“Переменные среды:”<<endl;
char** p;
for( p=envp; *p!=NULL; p++ ) cout<<*p<<endl;
return 0;
}
```

70

Прикладной интерфейс программирования

Прикладной интерфейс программирования (API – Application Programming Interface) – это набор классов и функций, предназначенных для организации взаимодействия прикладной программы с какой-либо программной системой (операционной системой, базой данных и т.д.)

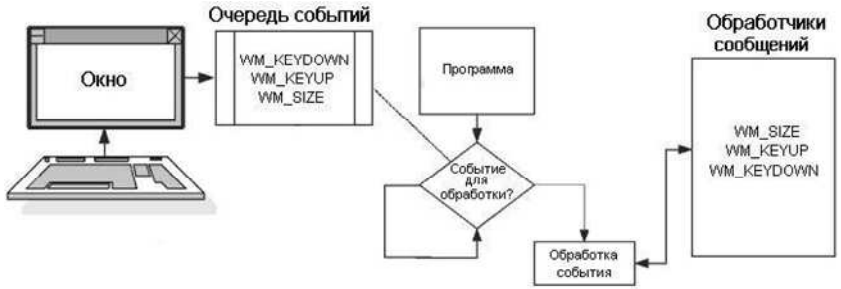
71

События и сообщения

Событие представляет собой сообщение, посылаемое объектом, чтобы сигнализировать о совершении какого-либо действия. Это действие может быть вызвано в результате взаимодействия с пользователем, например при нажатии кнопки мыши, или может быть обусловлено логикой работы программы. Объект, вызывающий событие, называется отправителем события. Объект, который захватывает событие и реагирует на него, называется получателем события.

72

Обработка событий



73

Пример программирования для Windows Forms .NET

```
#using <mscorlib.dll>
#using <System.Windows.Forms.dll>
using namespace System::Windows::Forms;
int main()
{
MessageBox::Show(“Hello, World!”);
return 0;
}
```

74

Пример программирования для Windows Forms .NET

```
#using <mscorlib.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

__gc class Hello: public Form {
    Button *testButton;
public:
    Hello()
    {
    Text=“Hello,World!”;
    testButton=new Button;
    testButton->Text=“Click me”;
    testButton->Top=120;  testButton->Left=100;
    testButton->Click += new EventHandler(this, Button_click);
    this->Controls->Add(testButton);
    }
```

Пример программирования для Windows Forms .NET

```
    void Button_click(Object* sender, EventArgs* e)
    {
    MessageBox::Show("Hello, Wold!");
    }
};      // конец определения класса Hello

int main()
{
Application::Run(new Hello);
return 0;
}
```

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
имени академика С.П. Королёва (национальный исследовательский университет)

ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ “ЯЗЫКИ ПРОГРАММИРОВАНИЯ” (ЯЗЫКИ ПРОГРАММИРОВАНИЯ C/C++)

САМАРА 2012

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
имени академика С.П. Королёва (национальный исследовательский университет)

ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ “ЯЗЫКИ ПРОГРАММИРОВАНИЯ” (ЯЗЫКИ ПРОГРАММИРОВАНИЯ C/C++)

Методические указания

САМАРА 2012

УДК 4480.24/29
Составитель Привалов А.Ю.

Лабораторные работы по курсу “Языки программирования” (языки программирования C/C++): Методические указания / Самар. гос. аэрокосм. ун-т; Сост. А.Ю. Привалов; Самара, 2012. 18 с.

Предназначены для проведения лабораторных работ по курсу “Языки программирования” для студентов бакалавриата, обучающихся по направлению “Прикладные математика и физика”. Также могут быть использованы при обучении программированию на языках C и C++ родственных направлений и специальностей.

Табл. 2, Ил. 6.

Печатаются по решению редакционно-издательского совета Самарского государственного аэрокосмического университета имени академика С.П.Королёва

Рецензент: В.В. Пшеничников

Лабораторная работа №1.
Начало работы со средой программирования Microsoft Visual Studio 2005.
Простейшие программы

1.1 Цели и постановка задачи

Цель: Познакомиться со средой разработки Microsoft Visual Studio 2005. Научиться программировать вычисления с помощью основных операторов языка С и определять собственные функции.

Задание:

1. Написать на языке С программу вычисления значения выражения, зависящего от одной переменной для задаваемого пользователем значения этой переменной. Выражение взять из таблицы 1 согласно номеру варианта. Программа должна запрашивать у пользователя значение аргумента x, при котором надо вычислить значение выражения, и после получения этого значения выводить на экран результат.

2. Переделать программу из предыдущего пункта так, чтобы она вычисляла значения выражения для заданного количества значений аргумента, равномерно заполняющих заданный интервал. Программа должна запрашивать пользователя ввести левый и правый границы интервала, и количество точек на нем. Соседние точки должны отстоять друг от друга на одинаковый шаг, первая точка должна совпадать с левой границей интервала, последняя – с правой границей. Результаты расчёта должны быть выведены в виде таблицы из двух столбцов: первый столбец – значение аргумента, второй – соответствующее значение выражения.

3. Написать на языке С функцию приближённого вычисления суммы бесконечного степенного ряда из таблицы 2 согласно номеру варианта. При работе функция должна суммировать не менее 10 членов ряда, а далее продолжать суммирование до тех пор, пока очередной член ряда не станет достаточно мал по сравнению с суммой предыдущих членов ряда. Окончание суммирования должно быть произведено, когда модуль отношения очередного члена ряда к сумме всех предыдущих членов не станет меньше некоторого заданного пользователем числа, которое будем называть “точностью”. Функция должна принимать в качестве аргументов два вещественных значения: первый аргумент – значение x, при котором надо вычислить значение суммы ряда, второй аргумент – “точность”. Возвращаемое функцией значение – это значение вычисленной суммы ряда.

4. С использованием функции из предыдущего пункта переделать программу, из пункта 2 так, чтобы в выводимой на экран таблице было четыре колонки: кроме значения аргумента и значения выражения из таблицы 1, выводилось бы также значение суммы ряда из таблицы 2, а также модуль разности значений выражения и суммы ряда. Программа должна запрашивать у пользователя границы интервала, число заполняющих этот интервал с постоянным шагом точек, в которых необходимо вычислять значения, и “точность” вычисления суммы ряда. При вводе границ интервала учитывать область допустимых значений x для данного ряда (указаны в таблице 2).

1.2 Начальные сведения о работе с Microsoft Visual Studio 2005.

Одними из основных единиц логической организации программного кода в среде Microsoft Visual Studio 2005 являются Solution и Project.

Project (проект) – включает в себя исходный код создаваемой программы.

Solution (решение) – может включать в себя один или несколько проектов, которые, как правило, логически связаны. Проекты в одном solution могут быть разного типа. Например, можно в один solution объединить проект web-сервиса на языке C# и проект программы на языке C++, которая осуществляет обращение к этому сервису.

Для создания проекта в среде Microsoft Visual Studio 2005 следует в главном меню выбрать File → New → Project. В появившемся диалоговом окне New Project в левой части (Project types) следует выбрать Visual C++ → Win32, а в правой (Templates) – Win32 Console Application (либо Other Languages → Visual C++ → Win32, а затем Win32 Console Application). Далее в поле Name этого же диалога следует указать имя проекта (например, Lab1), в поле Location следует указать имя директории, где будут храниться файлы проекта, в поле Solution Name следует указать имя создаваемого solution (например, Labs). В дальнейшем в этот же solution можно будет добавлять проекты последующих лабораторных работ. Далее следует нажать кнопку OK, после чего появится мастер создания проекта (Win32 Application Wizard), в котором следует выбрать пункт Application Settings, либо нажать кнопку Next. На странице Application Settings мастера следует убедиться, что в качестве Application Type выбрано Console application, а в Additional options выбрать опцию Empty Project, нажать кнопку Finish и завершить тем самым создание проекта.

После создания проекта программы в него можно добавлять файлы исходного кода. Для этого следует найти инструментальное окно Solution Explorer, в котором отображается структура solution и проекта. Если Solution Explorer не открыт по умолчанию, то его можно вызвать, выбрав View → Solution Explorer в главном меню Visual Studio. Для добавления файла исходного кода следует в Solution Explorer правой кнопкой мыши нажать на пиктограмму проекта (Lab1 в нашем случае), и в появившемся меню выбрать Add → New Item. В появившемся диалоговом окне Add New Item в левой части (Categories) следует выбрать Code, а в правой части (Templates) выбрать C++ file (.cpp), а затем в поле Name задать имя (например lab1), после чего в проект добавится файл lab1.cpp. Вместо этого можно в диалоговом окне Add New Item в поле Name явно указать имя файла исходного кода с расширением (.c или .cpp). Затем следует нажать кнопку Add, после чего добавленный файл появится в окне Solution Explorer и одновременно откроется для редактирования. В появившемся редакторе исходного кода следует написать код программы. При копировании solution в другое место (например, на usb-flash) следует полностью копировать содержащую его директорию со всеми вложенными в него файлами и поддиректориями.

Ниже представлен пример простейшего варианта программы, вычисляющей логарифм введённого пользователем вещественного числа:

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

void main()
{
double x;

printf("Calculation of function ln(x)\n");
printf("(x > 0) x = ");
scanf("%lg", &x);
printf("f(x) = %f\n", log(x));
}
```

1.3 Требования к коду программы

При программировании вычисления значения степенного ряда в заданной точке следует избегать использования функции pow (возведение в степень) там, где без нее можно обойтись. Также, как правило, не следует писать отдельную функцию для вычисления факториала или двойного факториала. Часто при программировании

вычисления значений степенных рядов последующий член ряда вычисляют через предыдущий, что уменьшает число операций при вычислении.

Пусть, например, дан ряд $f(x)=\sum_{n=0}^{\infty}(-1)^n\frac{x^n}{n!}=1-x+\frac{x^2}{2}-\frac{x^3}{6}+\ldots$

Нетрудно заметить, что каждый последующий член ряда можно получить из предыдущего, умножив его на (–*x*), и поделив на номер члена в ряду.

Ниже представлен код функции на языке С, которая производит вычисление N первых членов этого ряда. Переменная Si служит для вычисления члена ряда, переменная S – для накопления суммы ряда, а переменная i (индекс цикла) – номер члена ряда, обрабатываемого на данной итерации.

```
double func(double x, int N)
{
int i;
double Si = 1, S = 1;

for (i = 1; i <= N; i++){
    Si = -Si * x / i;
    S = S + Si;
    }
return S;
}
```

1.4 Варианты задания

Таблица 1

№	Выражение	№	Выражение
1	$f(x)=x\cos x-\sin x$	2	$f(x)=\ln\left(\sqrt{\frac{1+x}{1-x}}\right)$
3	$f(x)=\frac{1}{4}\ln\left(\frac{1+x}{1-x}\right)+\frac{1}{2}\operatorname{arctg}x$	4	$f(x)=\operatorname{arctg}\frac{2-2x}{1+4x}$
5	$f(x)=x\operatorname{arctg}x-\ln\sqrt{1+x^2}$	6	$f\left(x\right)=\left(1+x^2\right)\operatorname{arctg}x$
7	$f\left(x\right)=\left(1+x\right)\ln\left(1+x\right)$	8	$f\left(x\right)=\left(1+x\right)\exp\left(-x\right)$
9	$f\left(x\right)=\left(1+x^2\right)\operatorname{arctg}x$	10	$f\left(x\right)=\exp x\cos x$
11	$f\left(x\right)=\exp x\sin x$	12	$f\left(x\right)=\ln^2\left(1-x\right)$
13	$f\left(x\right)=\frac{\ln\left(1+x\right)}{1+x}$	14	$f(x)=\frac{1}{(1-x^2)\sqrt{1-x^2}}$

Таблица 2

№	Ряд
1	$f(x)=\sum_{n=1}^{\infty}(-1)^n\frac{2n\cdot x^{2n+1}}{(2n+1)!}$

№	Ряд
2	$f(x)=\sum_{n=0}^{\infty}\frac{x^{2n+1}}{2n+1}$ (-1<x<1)
3	$f(x)=\sum_{n=0}^{\infty}\frac{x^{4n+1}}{4n+1}$ (-1<x<1)
4	$f(x)=arctg\,2+\sum_{n=1}^{\infty}\frac{(-1)^n 2^{2n-1}}{2n-1}x^{2n-1}$ (-1/4<x≤1/2)
5	$f(x)=\sum_{n=1}^{\infty}(-1)^{n+1}\frac{x^{2n}}{2n(2n-1)}$ (-1≤x≤1)
6	$f(x)=\sum_{n=0}^{\infty}(-1)^{n}\frac{x^{2n+1}(1+x^2)}{2n+1}$ (-1≤x≤1)
7	$f(x)=\sum_{n=1}^{\infty}(-1)^{n-1}\frac{x^{n}(1+x)}{n}$ (-1≤x≤1)
8	$f(x)=1+\sum_{n=2}^{\infty}\frac{(-1)^{n+1}(n-1)}{n!}x^{n}$
9	$f(x)=x+2\sum_{n=1}^{\infty}\frac{(-1)^{n+1}}{4n^2-1}x^{2n+1}$ (-1≤x≤1)
10	$f(x)=\sum_{n=0}^{\infty}\frac{2^{\frac{n}{2}}\cos\left(\frac{n\pi}{4}\right)}{n!}x^{n}$
11	$f(x)=\sum_{n=0}^{\infty}\frac{2^{\frac{n}{2}}\sin\left(\frac{n\pi}{4}\right)}{n!}x^{n}$
12	$f(x)=2\sum_{n=1}^{\infty}\left(1+\frac{1}{2}+\ldots+\frac{1}{n}\right)\frac{x^{n+1}}{n+1}$ (-1≤x<1)
13	$f(x)=\sum_{n=1}^{\infty}\left\{(-1)^{n-1}\left(1+\frac{1}{2}+\ldots+\frac{1}{n}\right)x^{n}\right\}$ (-1<x<1)
14	$f(x)=\sum_{n=0}^{\infty}\frac{(2n+1)!!}{(2n)!!}x^{2n}$ (-1<x<1)

Лабораторная работа №2.
Битовые операции в языке С.

2.1 Цели и постановка задачи

Цель: Научится использовать битовые операции над целыми типами данных и работу с битовым представлением целых чисел в памяти компьютера.

Задание:

1. Написать функцию вывода на экран битового представления целой переменной без знака (unsigned int) с разбиением на удобно-читаемые части (байты или тетрады). Для простоты реализации разрешается выводить биты так, чтобы младший бит был самым левым, а старший – самым правым. Функция должна корректно работать для возможных разных размеров целых переменных

2. Написать программу, реализующую заданный вариант преобразования данных с использованием битовых операций (арифметические операции не использовать). Программа перед началом вычислений должна запрашивать у пользователя исходное значение, выводить на экран его битовое представление, а после вычислений – битовое представление результата. Реализацию битовых вычислений в программе оформить как функцию.

2.2 Требования к коду программы

При выполнении данного задания, а так же последующих необходимо выполнять ряд требований, которые помогают программисту допускать меньшее число ошибок:

1) Имена, которыми мы можем обозначать программные объекты, (константы, переменные, функции, типы данных) называются идентификаторами. Для облегчения понимания программы идентификаторы в них должны быть осмысленными именами, отражающими назначение именуемых программных объектов, т.е., например, если переменная предназначена для хранения количества символов, ее можно назвать, скажем, **charCount** (от английского сокращения словосочетания «счётчик символов»). Замете, что название переменной состоит из двух слов, чтобы разделить которые в данном примере используется т.н. “верблюжья” нотация, т.е. начало слов внутри имени отмечается заглавной буквой. Есть и другой способ - отделять слова символом подчеркивания, в нашем случае это выглядит так: **char_count**. В качестве исключения из правила осмысленных имён могут служить индексные переменные, используемые, например, в циклах.

2) Для легкого чтения кода и упрощения поиска ошибок (например, при отладке программы) оформлять код следует в соответствии с «лесенкой»: т.е. если тело оператора цикла или условного оператора является составным оператором (последовательностью операторов в фигурных скобках), то содержание этого составного оператора записывается с дополнительным отступом. Пример структурирования кода:

```
if (a>0) {
   x++;
   for(y=1; y<x; y++) {  // цикл for внутри оператора if
      z*=sin(y)/y;
      s+=z;
      }                  // конец тела цикла for
   printf("z=%g, s=%g\n", z, s);
   }                  // конец тела первой ветви оператора if
else {
   s=0;
   x=1;
   }                  // конец тела второй ветви оператора if
```

3) Если нужное действие может быть оформлено и в виде оператора, и в виде выражения, (например, условного оператора и условного выражения) лучше использовать выражение, чем оператор. Это связанно с тем, что выражения оптимизируются компилятором лучше, чем операторы. Поэтому при излишнем использовании операторов теряется быстродействие программы. Так же следует помнить, что заводить лишние переменные, особенно массивы, там, где этого можно избегать, не целесообразно. Оперативная память является важнейшим ресурсом программ, написанных на C/C++. Кроме того, чем больше кода (разных переменных) в программах – тем больше ошибок в них может быть.

3.3 Варианты задания

1. Перевернуть битовое представление числа – первый бит сделать последним, второй – предпоследним и т.д.
2. Поменять местами старший и младший байт в целом числе.
3. Поменять местами тетрады в младшем байте числа – старшую с младшей.
4. Поменять тетрады в первых двух байтах числа – 1-ю с 4-ой, а 2-ю и 3-ю не трогать.
5. Поменять тетрады в первых двух байтах числа – 2-ю с 3-ей, а 1-ю и 4-ю не трогать.
6. Поменять тетрады в первых двух байтах числа – 1-ю с 3-ей, а 2-ю и 4-ю не трогать.
7. Поменять тетрады в первых двух байтах числа – 2-ю с 4-ой, а 1-ю и 3-ю не трогать.
8. В первых двух байтах числа заменить младший байт на перевёрнутый старший.
9. В первых двух байтах числа заменить старший байт на перевёрнутый младший.
10. Посчитать количество единиц в двоичной записи числа.
11. Посчитать количество нулей в двоичной записи числа.
12. Из двух первых байт целого числа получить другое число следующим образом: в младшем байте собрать все нечётные биты из исходных двух байт, а во втором байте – чётные. Старшие байты результата (если они есть) должны содержать нули.
13. Из двух целых чисел a и b собрать одно, состоящее из чётных битов a и нечётных битов b.
14. Из двух целых чисел a и b собрать одно, где 1-ая и 3-я тетрады из a, а 2-ая и 4-ая – из b. Остальные биты результата (если они есть) должны быть равны нулю.
15. Дано два целых числа a и b. В a занулить те нечётные биты, которые в b равны 1. Чётные биты a не трогать.
16. Дано два целых числа a и b. В a установить в 1 те чётные биты, которые в b равны 0. Нечётные биты a не трогать.
17. Дано два целых числа a и b. Получить из них два целых числа следующим образом – первое состоит из старших байтов a и b, второе – из младших (в данном пункте надо написать две функции).

Лабораторная работа №3.
Работа с массивами и файловый ввод-вывод.

3.1 Цели и постановка задачи

Цель: Научится работать с файлами для чтения и записи в них числовых данных в текстовом формате, а так же изучить обращение с таким типом данных как массивы.

Задание:

1. Написать функцию, реализующую указанный вариант обработки данных, аргументами которой являются необходимые матрицы (двумерные массивы) и вектора (одномерные массивы). Также аргументами функции должны быть размерности используемых массивов.

2. Написать программу, которая использует функцию из пункта 1 для обработки матриц и векторов, значения которых загружаются из файлов. После загрузки из файлов начальных данных программа выводит их на экран, потом производит обработку, записывает получившиеся результаты в файлы, и выводит результаты на экран.

3.2 Требования к коду программы

При выполнении данного задания, а так же последующих необходимо выполнять ряд требований, которые помогают программисту допускать меньшее число ошибок:

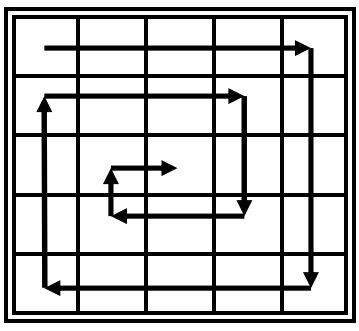
1. Избегайте использования глобальных переменных. Это приводит к частым ошибкам, связанных с использованием одного и того же ресурса для разных целей, “утечкам памяти” и непредсказуемому поведению программы при выходе за границы выделенной памяти. Самой распространенной ошибкой является использование одной и той же глобальной переменной в разных функциях. Одна функция изменила для себя ее значение, а вторая, к которой вернулась программа, нужное ей значение этой переменной утратило, так как его затерла предыдущая функция. Другим часто встречающимся подводным камнем являются массивы. Если их объявлять глобальными, то обращаться можно к любому их элементу из любого места программы, притом, что выход за границы массива не приводит к ошибке исполнения, ошибка в вычислениях будет неявной и ее будет трудно найти.

2. При описании переменных или массивов, инициализируйте их (придавайте начальное значение, особенно если оно известно).

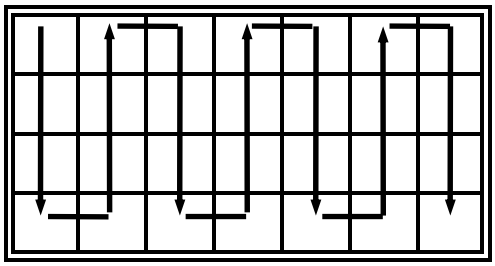
3. Если вы решили использовать один и тот же файл и для чтения, и для записи, следует помнить, что функция fopen при этом требует использования специальных форматов, например "r+", "w+". В программе необходимо учитывать возможность того, что функция открытия файла завершит работу неуспешно, и при необходимости надо отображать сообщение об этой ошибке и прекращать выполнение программы.

3.3 Варианты задания

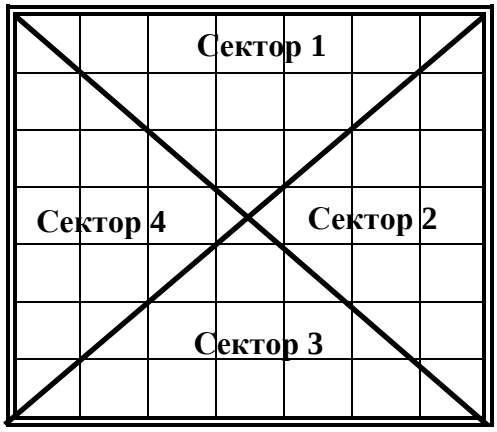
1. Дана матрица $A(n \times n)$ целых чисел. Переписать в вектор $D(n^2)$ элементы матрицы в следующем порядке:



2. Дана матрица *A(n×m)* целых чисел. Переписать в вектор *D(n×m)* элементы матрицы в следующем порядке:



3. Дана матрица *A(n×n)* целых чисел. Поменять местами элементы главной и побочной диагонали матрицы. Элементы, находящиеся в секторах 1 и 3, обнулить, а элементы, находящиеся в секторах 2 и 4, удвоить. При работе с секторами элементы, принадлежащие диагоналям матрицы, не изменять.



4. Дана матрица *A(n×n)* целых чисел. Получить вектор, элементы которого равны сумме минимального и максимального элементов соответствующих строк 1 и 3 секторов. (См. рис варианта 3).

5. Дана матрица *A(n×n)* целых чисел. Получить вектор, элементы которого равны сумме минимального и максимального элементов соответствующих столбцов 2 и 4 секторов. (См. рис варианта 3).

6. Дана матрица *A(n×n)* целых чисел. Поменять местами четверти матрицы по следующему принципу: элементы первой четверти должны стать элементами третьей, элементы четвертой - второй и наоборот.

IV	IV	I	I
IV	IV	I	I
III	III	II	II
III	III	II	II

7. Дана матрица *A(n×m)* целых чисел. Определить ее «седловую точку», т. е. значение, являющееся одновременно максимальным в своей строке и минимальным в своем столбце. На экран, кроме исходной матрицы, вывести номер строки, номер столбца и значение «седловой точки». Если «седловой точки» у матрицы нетнет, вывести соответствующее сообщение

8. Дана матрица *A(n×n)* целых чисел, составленная из чисел **1 , 2 , 4 , . . . n^2**. Определить, является ли она «магическим квадратом» (т.е. суммы по каждому столбцу, каждой строке и каждой из двух диагоналей равны между собой).

9. В матрице символов *A(n×m)* подсчитать количество фрагментов вида (**0** обозначает символ 0, ***** - любой другой символ):

O	*****	**O**
*****	**O**	*****
O	*****	**O**

10. Дана матрица *A(n×n)* целых чисел. Найти среднее арифметическое наибольшего и наименьшего значения ее элементов. Если полученный результат больше нуля, то поменять местами элементы главной и побочной диагоналей. Если результат отрицательный, то изменить знаки элементов, расположенных выше главной диагонали на противоположный.

11. Дана матрица *A(n×n)* целых чисел. Получить вектор *C(n)*, элементы которого равны произведениям элементов, стоящих на главной и побочной диагоналях матрицы. В полученном векторе найти минимальный и максимальный элементы. На место минимального элемента, записать 0, а на место максимального элемента, записать максимально возможное целое число.

12. Дана матрица *A(n×n)* целых чисел. Найти минимальный элемент в главной диагонали и максимальный элемент в побочной диагонали. Все элементы матрицы, находящиеся ниже побочной диагонали, увеличить на максимальный элемент, а элементы, находящиеся выше побочной диагонали, уменьшить на минимальный элемент.

13. Дана матрица *A(n×n)* целых чисел. Найти минимальный и максимальный элементы матрицы. Если минимальный элемент - четный, то обнулить часть матрицы, находящуюся над главной диагональю, а если нечетный и кратный заданному значению, то сменить знак на противоположный у элементов, находящихся над побочной диагональю.

14. Даны две матрицы *A(n×k)* и *B(k×m)* целых чисел. Получить матрицу - произведение заданных матриц *C(n×m)*.

15. Дана матрица *A(n×m)* целых чисел. Получить вектор, элементы которого равны произведениям элементов соответствующих столбцов матрицы. Если элемент вектора - величина отрицательная, то минимальный и максимальный элементы соответствующего столбца матрицы обнулить.

16. Дана матрица *A(n×m)* целых чисел. В столбцах с номерами *p* и *q* найти элементы равные между собой в текущей строке. Элементы строк, в которых находятся найденные значения, обнулить. Если равные элементы не будут найдены, то обнулить заданные столбцы.

17. Даны матрица *A(n×m)* и вектор *B(n)*. Получить два новых вектора *C(n)* и *D(n)*. В вектор ***C*** поместить индекс первого вхождения элемента вектора ***B*** в соответствующую строку исходной матрицы. В вектор ***D*** поместить индекс последнего вхождения элемента вектора ***B*** в соответствующую строку исходной матрицы. Если в строке матрицы элемент из вектора ***B*** отсутствует, то в соответствующие элементы векторов ***C*** и ***D*** записать нули.

18. Даны матрица *A(n×m)* и вектор *B(n)*. Получить матрицу *C(n×m)* такую, что:

$$c_{ij} = b_i, \quad \text{при } a_{ij} > 0$$
$$c_{ij} = -b_i, \quad \text{при } a_{ij} < 0$$
$$c_{ij} = 0, \quad \text{при } a_{ij} = 0$$

19. Даны две матрицы *A(n×m)* и *D(n×m)* целых чисел. Получить матрицу *B(n×m)* по следующему правилу:

$$b_{ij} = 1 \quad \text{если } a_{ij} = d_{ij}, \text{ и } a_{ij} > 0$$
$$b_{ij} = -1 \quad \text{если } a_{ij} = d_{ij}, \text{ и } a_{ij} \le 0$$
$$b_{ij} = 0 \quad \text{если } a_{ij} \ne d_{ij}$$

20. Дана матрица *A(n×m)* целых чисел. Получить вектор, элементы которого равны суммам элементов соответствующих строк матрицы. Если сумма ≥ заданной величины, элементы матрицы в данной строке обнулить, в противном случае сменить их знак на противоположный.

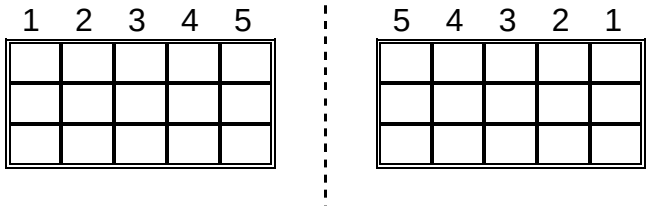
21. Дана матрица *A(n×m)* целых чисел. Получить два новых вектора логических значений *B(n)* и *C(m)*. Положить b_i равным истина, если в i-ой строке матрицы есть положительные элементы, и ложь, если нет. Аналогично, элемент вектора c_i должен показывать наличие в соответствующем столбце отрицательных элементов.

22. Дана матрица *A(n×m)* целых чисел. Получить вектор *C(m)*, каждый элемент которого равен количеству элементов, стоящих до нулевого элемента в соответствующих столбцах матрицы. Получить вектор *B(n)*, каждый элемент которого равен сумме элементов, стоящих до нулевого элемента в соответствующих строках матрицы.

23. Дана матрица *A(n×m)* действительных чисел. Заменить нулями все элементы, отличающиеся от среднего значения более чем на заданную величину.

24. Дана матрица *A(n×m)* целых чисел. Получить вектор *C(m)*, элементы которого равны максимальным элементам соответствующих столбцов матрицы. Найти сумму элементов матрицы и минимальный элемент вектора увеличить на полученное значение, а максимальный элемент заменить на максимальное целое число.

25. Дана матрица *A(n×m)* целых чисел. Получить новую матрицу, симметричную исходной относительно вертикальной оси. Вывести обе матрицы рядом. Пронумеровать строки и столбцы, так, чтобы нумерация столбцов новой матрицы шла в обратном порядке.



26. Дана матрица *A(n×m)* целых чисел. Определить максимальный элемент и количество максимальных элементов, минимальный элемент и количество минимальных элементов за один просмотр матрицы.

27. Дана матрица *A(n×m)* целых чисел. Получить вектор *B(n)*, где b_k - сумма наибольшего и наименьшего элементов *k* -ой строки.

28. Дана матрица *A(n×m)* целых чисел. Получить вектор *X(n)*, элементы которого равны номерам максимальных элементов соответствующих строк матрицы.

29. Даны два вектора *X(n)* и *Y(n)* целых чисел. Получить "таблицу умножения" этих векторов: каждый элемент вектора ***X*** умножается на каждый элемент вектора ***Y***.

30. Даны два вектора *A(n)* и *B(n)* целых чисел. Если $a_i < b_i$, то поменять значения местами, так чтобы максимальные значения были в векторе ***A***.

Лабораторная работа №4.
Алгоритмы сортировки.

4.1 Цели и постановка задачи

Цель: Изучить несколько алгоритмов сортировки и сравнить их свойства.

Задача: Написать программу на языке С, реализующую два заданных алгоритма сортировки (согласно варианту). Сравнить скорость работы алгоритмов на массивах различной длины.

4.2 Общие сведения.

Если над некоторым множеством элементов определена операция сравнения, то такое множество может быть упорядочено. Для упорядочивания наборов данных, представленных в виде массивов, существуют различные алгоритмы. Алгоритмы классифицируются по различным характеристикам. В частности, по зависимости количества необходимых операций от размера входных данных, называемой вычислительной эффективностью (или просто быстродействием). В данной лабораторной работе рассматриваются примеры алгоритмов, обладающих квадратичной и субквадратичной вычислительной эффективностью (такие алгоритмы принято называть быстрыми).

4.3 Задание

1. Написать две функции сортировки массива целых чисел, реализующих заданные алгоритмы сортировки – один из класса квадратичных алгоритмов, другой из класса быстрых алгоритмов.

2. Исследовать вычислительную эффективность этих алгоритмов. Для этого необходимо использовать функции стандартной библиотеки, которые позволяют измерять время работы участка программы. Программа для измерения вычислительной эффективности должна создавать массив для сортировки, заполняя его случайными числами, потом вызывать функцию сортировки, замеряя время её работы. Если время одной сортировки слишком мало, надо проводить генерацию случайного массива и его сортировку в цикле большое число раз и измерять суммарное (либо среднее) время сортировки. Результатом данного пункта должны быть времена выполнения сортировки обоими методами для размеров массивов примерно от 1000 до 100000 (6-7 значений размера).

4.4 Варианты задания

1. Шейкер-сортировка и пирамидальная сортировка (heapsort).
2. Шейкер-сортировка и сортировка слиянием.
3. Шейкер-сортировка и быстрая сортировка (quicksort).
4. Пузырьковая сортировка и сортировка слиянием.
5. Пузырьковая сортировка и быстрая сортировка (quicksort).

Примечания:

1). Шейкер – сортировкой называется алгоритм, при котором итерации, подобные итерациям пузырьковой сортировки проводятся по массиву поочерёдно – то с младших индексов к старшим, то в противоположном направлении.

2). Для измерения времени работы программы можно воспользоваться функцией **clock( )** (заголовочный файл **time.h**), которая при вызове возвращает значение, равное количеству “тиков” системных часов, прошедших от начала работы программы до момента вызова. В **time.h** также определена константа для перевода “тиков” в секунды.

Лабораторная работа №5.
Сложные структуры данных.

5.1 Цели и постановка задачи

Цель: Познакомиться со сложными и динамическими структурами данных.
Задание: Написать программу на языке С, реализующую работу с заданной структурой данных (согласно варианту).

5.2 Общие сведения.

Массив как линейная структура данных обладает как достоинствами, так и недостатками. Например, доступ к элементу по индексу есть операция со сложностью порядка *O(1)*, тогда как нахождение элемента в массиве по значению более трудоемкая операция, и ее сложность порядка *O(N)*, где *N* – число элементов в массиве (если массив не отсортирован). Если массив отсортирован, то сложность этой операции *O(logN)*, но при добавлении данных в отсортированный массив приходится восстанавливать упорядоченность, что требует *O(N)* действий. Для быстрого поиска, добавления и удаления элементов по значению придуманы специальные структуры данных, такие как, например, двоичные деревья поиска, кучи-пирамиды (heap) и хэш-таблицы.

5.3 Варианты заданий

1. Хэш-таблица.
2. Двоичное дерево поиска.
3. Двунаправленный список.
4. Однонаправленный список.
5. Куча-пирамида (heap).

Для списков написать программу, которая по указанию пользователя помещает в список заданное значение, удаляет из списка заданное значение и распечатывает текущее содержимое списка. На основе однонаправленного списка реализовать стек, а на основе двунаправленного списка – очередь.

Для двоичного дерева поиска написать программу, которая помещает в дерево заданное пользователем количество случайных данных, распечатывает получившееся дерево, потом ищет и удаляет в получившемся дереве заданное пользователем число и т.д. Также содержимое дерева должно выводиться на экран в удобном для просмотра виде.

Для кучи и хэш-таблицы написать программу, которая помещает в структуру данных заданное число случайных данных, а потом забирает данные из структуры данных и распечатывает их, пока структура данных не опустеет. При реализации хэш-таблицы предусмотреть возможность вывода на экран сообщения при возникновении коллизии, а также текущего состояния хэш-таблицы.

Лабораторная работа №6.
Основы объектно-ориентированного программирования.

6.1 Цели и постановка задачи

Цель: Познакомиться с основами объектно-ориентированного программирования в C++ на примере создания простого класса, реализующего заданную структуру данных.

Задание: Написать класс на языке C++, реализующий заданную структуру данных (согласно варианту). Написать программу, демонстрирующую работу данного класса.

6.2 Варианты задания

1. Хэш-таблица.
2. Двоичное дерево поиска.
3. Двунаправленный список.
4. Однонаправленный список.
5. Куча-пирамида (heap).

Примечания: данные, хранящиеся в структуре данных должны быть полностью защищены от некорректного доступа. Доступ к ним должен быть возможен только с использованием методов класса. Кроме методов класса и обычного конструктора (а также деструктора, если это необходимо), должен быть реализован конструктор копирования.

Лабораторная работа №7
Основы механизма наследования классов.

7.1 Цели и постановка задачи

Цель: Познакомиться с основами объектно-ориентированного программирования в C++ на примере простой структуры классов прикладной задачи.

Задание:

1. Используя наследование, полиморфизм, инкапсуляцию и композицию, написать структуру классов заданного варианта. Интерфейс класса главного объекта должен включать методы добавления, удаления и поиска элементов (объектов внутренних классов), хранимых внутри главного объекта, а также методы вывода на экран содержимого главного объекта в удобном для пользователя виде..
2. Реализовать для класса главного объекта и для классов объектов, хранимых внутри главного, виртуальные методы сохранения и загрузки из файла.
3. Написать тест, в котором будет создан главный объект и заполнены его поля (не через конструктор, а через соответствующие методы). Затем этот объект должен сохраняться в файл и из этого же файла данные должны загружаться в новый объект. Содержимое обоих объектов далее должно быть выведено на экран для сравнения.

7.2 Варианты задания

1. Логистическая компания (главный объект) имеет название, владеет N фурами (грузовыми автомобилями), каждая из которых должна сопровождаться одним водителем. У водителя есть имя, ежемесячная заработная плата, и стаж вождения. Водители фур могут выполнять еще работу механиков. Механик должен характеризоваться конкретным уровнем познаний авто-техники, и у всех он может быть разный.
2. Библиотека (главный объект) имеет номер общегородской, адрес по которому она находится и статус (детская, городская, областная…). В библиотеке есть N единиц хранения (книг или журналов). Для журнала должно быть известно название журнала, год и месяц издания. Для книги помимо этого существует еще и автор.
3. Школа (главный объект) имеет общегородской номер, статус (средняя, лицей, специальная…). В школе работают M учителей и учатся N школьников. Школьники имеют имя, класс в котором учатся. Учителя – имя, возраст, предметы, которые преподают. Один из учителей является директором школы, ещё несколько – завучами. У них у всех должен быть стаж работы в должности директора или завуча.
4. Кафедра института (главный объект) включает в себя N сотрудников (преподавателей). У кафедры есть название, факультет к которому она относится. Каждый сотрудник характеризуется именем, стажем преподавания, званием (доцент, профессор) и должностью (ассистент, преподаватель, доцент, профессор). Руководит кафедрой заведующий, являющийся также сотрудником кафедры, у заведующего есть заместитель, выбранный так же из сотрудников. У них обоих есть стаж нахождения на руководящей должности.
5. В игре WarCraft есть игровая карта (главный объект) которая имеет прямоугольный размер (например 128 на 128), название (Летний Лордерон, Нортренд, Калимдор, Кель-Талас и т.д.) и тип ландшафта. На карте располагается N юнитов двух типов: обыкновенные и герои. Первые характеризуются силой, ловкостью и интеллектом и у них есть имена, а у вторых добавляется еще определенный навык и возможность повышать свои характеристики.
6. В компьютерном кластере (главный объект) используются обычные стационарные компьютеры и сервера (N штук). Стационарные характеризуются частотой процессора, объемом оперативной памяти, размером HDD, названием операционной системы. Сервер помимо этой информации имеют количество процессоров, а так же количество HDD в RAID массиве.

7. В автосалоне (главный объект) под известным названием продается N машин. Машины характеризуются маркой, годом выпуска и стоимостью, типом топлива (бензин, дизель, газ). Машины могут быть двух типов: легковые и грузовые, каждый тип имеет дополнительные характеристики. Легковые машины характеризуются вместимостью (колвом пассажиров), скоростью передвижения и классом комфортности. Грузовые – проходимостью и грузоподъёмностью (количеством тон, которые они способны провозить в качестве груза).

8. Звёздная система (звезда, вокруг которой вращаются N планет) является главным объектом. Планеты и звезда являются небесными телами, которые характеризуются массой, радиусом, температурой поверхности и названием. Звезда ещё характеризуется звездной величиной (яркостью светимости), а планеты радиусом орбиты, типом атмосферы (газовая, метеоритная, и т.п.).

9. Поезд (главный объект), включает в себя локомотив и N вагонов. Для всего поезда известен его номер (состоящий из цифр и русских букв) рейс – станция отправления и назначения, время отправления и назначения и имя начальника вагона. Помимо этой информации каждый вагон имеет свой номер и имя проводника. Локомотив имеет тип и содержит имена машинистов.

Лабораторная работа №8.
Интерфейс командной строки

8.1 Цели и постановка задачи

Цель: знакомство со стандартными средствами передачи аргументов командной строки в программу.

Задание: написать программу обработки текстового файла, получающую через интерфейс командной строки при запуске программы имя обрабатываемого файла, а также при необходимости, имя выходного файла для записи результата. В случае отсутствия имени выходного файла программа должна выводить результат на экран. В случае запуска программы без параметров, она должна выводить на экран краткое описание правильного её использования.

Пример: пусть программа, которая называется **wordcount** подсчитывает количество слов в текстовом файле (словом считается любая последовательность непробельных символов между двумя пробельными символами), и выводит в качестве результата сообщение: **“A text in file** <имя файла> **consists of** <кол-во слов> **words.**”

Тогда при запуске программы с двумя параметрами, например, таким:

wordcount test.txt result.txt

в файл **result.txt** будет записано (предполагая, что в исходном файле 5 слов):

A text in file test.txt consists of 5 words.

При запуске с одним параметром, например:

wordcount test.txt

надпись будет выведена на экран, а при запуске без параметров, то есть

wordcount

на экран будет выведен текст пояснения о том, как пользоваться программой, например:

This program counts number of words in a text file. Correct usage is
wordcount input [output]
where input is name of input file, output is optional name of output file. If output filename is not given, output is made to the screen.

8.2. Варианты задания

1. Подсчёт, сколько раз в файле повторяется каждое из содержащихся в нём слов, и вывод списка слов с количеством повторений.
2. Поиск слова, которое повторяется наибольшее количество раз, и вывод этого слова с числом его повторений.
3. Подсчёт статистики длин слов в файле и её вывод, то есть слов длины 1 столько-то, слов длины 2 столько-то и т.д.
4. Поиск самого длинного слова в файле. Если таких слов несколько, вывести их все (повторения допускаются).
5. Подсчёт, сколько раз в файле встречается та или иная буква (цифры и знаки препинания не учитывать), и вывод результата по всем буквам.
6. Подсчёт, сколько раз в файле встречается та или иная цифра (буквы и знаки препинания не учитывать), и вывод результата по всем буквам.
7. Подсчёт, сколько и каких в файле знаков препинания.
8. Поиск буквы, которая встречается в файле наибольшее число раз.
9. Подсчёт числа слов в файле.

Учебное издание

ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ “ЯЗЫКИ ПРОГРАММИРОВАНИЯ” (ЯЗЫКИ ПРОГРАММИРОВАНИЯ C/C++)

Методические указания

Составитель: Привалов Александр Юрьевич

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва
443086, Самара, Московское шоссе, 34

Список тем практических занятий с примерами возможных задач

Тема 1: Основы языка, выражения, операторы.
Возможные задачи:

- **Найти значение выражения, в котором использованы специальные операторы присваивания**
- **Найти значение выражения, в котором использованы операции инкремента и декремента**
- **Написать условный оператор, решающий квадратное уравнение**
- **Написать цикл, складывающий все целые числа в указанном диапазоне**

Тема 2: Основы языка, операторы, функции

- **Написать цикл, вычисляющий целую положительную степень заданной переменной**
- **Написать цикл, вычисляющий конечную сумму членов заданного ряда**
- **Написать функцию, вычисляющую конечную сумму членов заданного ряда, у которой количество суммируемых членов является аргументом**
- **Написать функцию, вычисляющую сумму ряда с заданной относительной точностью, где точность задаётся в качестве аргумента**

Тема 3: Битовые операции

- **Написать код, формирующий заданную битовую маску, например, 010101…**
- **Написать код, проверяющий значение заданного бита**
- **Написать код, устанавливающий заданное значение заданного бита**
- **Написать функцию, меняющую местами чётные и нечётные биты целого числа**
- **Написать функцию, выводящую на экран битовое представление числа с использованием только битовых операций**

Тема 4: Основы работы с массивами

- **Написать код, вычисляющий сумму элементов массива**
- **Написать код, находящий сумму элементов в каждой строке двумерного массива, и записывающий её в одномерный массив.**
- **Написать код, транспонирующий квадратный массив относительно главной диагонали**
- **То же самое относительно побочной диагонали**
- **Написать код, перебирающий элементы двумерного массива в необычном порядке, например, змейкой, спиралью и т.д.**

Тема 5: Рекурсия

- **Написать нерекурсивную и рекурсивную версию возведения числа в целую степень**
- **Написать нерекурсивную и рекурсивную версию вычисления факториала**
- **Написать функцию, подсказывающую перестановки колец для Ханойских башен**

Тема 6: Алгоритмы перебора

- **Написать функцию генерации следующей перестановки**
- **Написать функцию, решающую задачу коммивояжёра**

Тема 7: Простейшие алгоритмы работы с массивами

- **Написать код, определяющий наибольший элемент в массиве**
- **Написать код, определяющий второй по величине элемент в массиве**
- **Написать код, который ищет заданный элемент в неупорядоченном массиве**
- **Написать дихотомический поиск в упорядоченном массиве**
- **Написать простейший вариант пузырьковой сортировки**
- **Улучшить простейший вариант пузырьковой сортировки контролем условия досрочного прекращения работы**
- **Улучшить пузырьковую сортировку контролем диапазона итерации**
- **Написать простейший вариант шейкер-сортировки**
- **Улучшить его аналогично улучшения пузырьковой сортировки**

Тема 8: Структуры данных

- **Написать простейшую реализацию стека на основе массива**
- **С помощью стека написать функцию, определяющую правильность расстановки скобок в строке текста**
- **Написать простейшую реализацию очереди с помощью кольцевого буфера**
- **Написать простейшую реализацию двунаправленного списка**
- **Написать простейшую реализацию двоичного дерева поиска**

Тема 9: Основы объектно-ориентированного подхода

- **Разработать интерфейс класса стек и его простейшую реализацию**
- **Разработать интерфейс класса очередь и его простейшую реализацию**
- **Разработать интерфейс класса двоичное дерево поиска и его простейшую реализацию**

Тема 10: Наследование классов

- **Разработать интерфейс базового и производных классов для программы складского учёта**
- **Разработать реализацию данного интерфейса**

Тема 11: Интерфейс командной строки

- **Написать код, выводящий на экран аргументы командной строки в обратном порядке**
- **Написать код, определяющий, есть ли среди аргументов командной строки заданное слово**

Примечание: каждая тема может рассматриваться на 1-2 занятиях, задачи, подобные решаемым в классе, задаются на дом.

Языки программирования

Вопросы для текущего контроля

---

1. В каком заголовочном файле объявляются функции ввода-вывода?

---

2. Каков диапазон типа unsigned long int?

---

3. Каков диапазон типа unsigned short int?

---

4. Каков диапазон типа unsigned char, если его значение интерпретировать как целое число?

---

5. Каков диапазон типа long int?

---

6. Каков диапазон типа short int?

---

7. Каков диапазон типа char, если его значение интерпретировать как целое число?

---

8. В каком заголовочном файле объявляются математические функции?

---

9. При выполнении какого из операторов for переменная i пробегает по порядку все целые значения от 1 до 10?

---

10. Какой из операторов языка C не содержит ошибки, если при равенстве значений **a** и **b** значение **x** должно быть **1**, а в противном случае **-1**?

---

11. В каком заголовочном файле объявляются функции работы с символами?

---

12. В каком заголовочном файле объявляются функции работы со строками?

---

13. В каком заголовочном файле объявляются функции общего назначения, в том числе функции работы с динамической памятью?

---

14. Каков размер типа данных float ?

---

15. Каков размер типа данных double ?

---

16. Каков размер типа данных long double ?

---

17. Каков размер типа данных long int ?

---

18. Каков размер типа данных short int ?

---

19. Каков размер типа данных char ?

---

20. Каков размер типа данных int ?

---

21. Пусть массив **a** определён в программе как
int a[10][10];
Какой тип имеет выражение **a[5]** ?

---

22. Пусть массив **a** определён в программе как
int a[10][10];
Какой тип имеет выражение **a** ?

---

23. Пусть массив **a** определён в программе как
int a[10][10];
Какой тип имеет выражение **a[5][5]** ?

---

24. Выберите правильный вариант описания указателя на функцию, возвращающую значение целого типа и не имеющую аргументов.

---

25. Выберите правильный вариант описания указателя на функцию, не имеющую возвращаемого значения и имеющую аргумент типа указатель на целое.

---

26. Пусть значением символьной переменной является **'A'**.
Что будет выведено на экран, если при выводе использован формат
"%-3c" ?
(В вариантах ответа символ _ обозначает пробел)

---

27. Пусть значением символьной переменной является **'A'**.
Что будет выведено на экран, если при выводе использован формат
"%3c" ?
(В вариантах ответа символ _ обозначает пробел)

---

28. Пусть значением целой переменной без знака является **20** (десятичное).
Что будет выведено на экран, если при выводе использован формат
"%-#5X" ?
(В вариантах ответа символ _ обозначает пробел)

---

29. Пусть значением целой переменной без знака является **20** (десятичное).
Что будет выведено на экран, если при выводе использован формат
"%5X" ?
(В вариантах ответа символ _ обозначает пробел)

---

30. Пусть значением целой переменной является **5**.
Что будет выведено на экран, если при выводе использован формат
"%- 3d" ?
(В вариантах ответа символ _ обозначает пробел)

---

31. Пусть значением целой переменной является **5**.
Что будет выведено на экран, если при выводе использован формат
"%-+3d" ?
(В вариантах ответа символ _ обозначает пробел)

---

32. Какое значение получит переменная **a** после выполнения следующего фрагмента кода?

```
#define SQR(x) x*x
y=5;
a=SQR(y+5);
```

---

33. Какое значение получит переменная **a** после выполнения следующего фрагмента кода?

```
#define MOD(x) (x<0)?-x:x
y=5;
a=MOD(y-10);
```

---

34. Какое значение получит переменная **a** после выполнения следующего фрагмента кода?

```
#define MOD(x) (x<0)?-x:x
y=1;
a=MOD(y-2);
```

---

35. Какое значение получит переменная **a** после выполнения следующего фрагмента кода?

```
#define SQR(x) x*x
y=1;
a=SQR(y+5);
```

---

36. Какое значение получит переменная **a** после выполнения следующего фрагмента кода?

```
#define SQR(x) x*x
y=0;
a=SQR(y+5);
```

---

37. Если в начале **a=1**, **b=2**, то чему станет равно значение **b** после выполнения оператора:

```
swich(a) {
case 1: b+=1;
case 2: b+=2;
case 3: b+=3;
```

default: b*=2;
}

---

38. Если в начале **a=1**, **b=2**, то чему станет равно значение **b** после выполнения оператора:
swich(a) {
case 1: b+=1; break;
case 2: b+=2; break;
case 3: b+=3; break;
default: b*=2; break;
}

---

39. Если в начале **a=4**, **b=2**, то чему станет равно значение **b** после выполнения оператора:
swich(a) {
case 1: b+=1;
case 2: b+=2;
case 3: b+=3;
default: b*=2;
}

---

40. Каково двоичное представление результата операции **a & b**, если
двоичное представление значения **a** есть 10011101,
а двоичное представление значения **b** есть 10010010

---

41. Каково двоичное представление результата операции **a | b**, если
двоичное представление значения **a** есть 10011101,
а двоичное представление значения **b** есть 10010010

---

42. Каково двоичное представление результата операции **a ^ b**, если
двоичное представление значения **a** есть 01011101,
а двоичное представление значения **b** есть 10010000

---

43. Предполагая, что целое число занимает один байт (8 бит), получите битовое представление числа **-5** в двоичном дополнительном коде.

---

44. Предполагая, что целое число занимает один байт (8 бит), получите битовое представление числа **-7** в двоичном дополнительном коде.

---

45. Предполагая, что целое число занимает один байт (8 бит), получите битовое представление числа **-6** в двоичном дополнительном коде.

---

46. Предполагая, что целое число занимает один байт (8 бит), получите битовое представление числа **-8** в двоичном дополнительном коде.

---

47. Предполагая, что целое число занимает один байт (8 бит), получите битовое представление числа **-9** в двоичном дополнительном коде.

---

48. Каково значение выражения **a << b**,
если двоичное представление **a** есть **00100111**,
а двоичное представление **b** есть **00000010**.

---

49. Каково значение выражения **a >> b**,
если двоичное представление **a** есть **10100111**,
а двоичное представление **b** есть **00000010**.

---

50. Пусть в начальный момент значения переменных таковы: **a=1, b=2, c=3**.
Какое значение получит переменная **a**, после того, как будет выполнен оператор: **a*=b-=c%=10;**

---

51. Пусть в начальный момент значения переменных таковы: **a=1, b=2, c=3**.
Какое значение получит переменная **a**, после того, как будет выполнен оператор:
a+=b*=c-=10;

---

52. Пусть в начальный момент значения переменных таковы: **a=1, b=2, c=31**.
Какое значение получит переменная **a**, после того, как будет выполнен оператор:
a+=b*=c/=10;

---

53. Пусть в начальный момент значения переменных таковы: **a=10, b=10, c=-3**.
Какое значение получит переменная **a**, после того, как будет выполнен оператор:
a/=b+=c*=2;

---

54. Пусть в начальный момент значения переменных таковы: **a=10, b=1, c=3**.
Какое значение получит переменная **a**, после того, как будет выполнен оператор:
a%=b*=c+=2;

---

55. Какой тип битовых сдвигов гарантируется стандартом С для беззнаковых целых?

Экзаменационные вопросы по курсу “Языки программирования”

1. Общее устройство ЭВМ. Роль программного обеспечения. Назначение языков программирования, их краткая история. Языки программирования высокого уровня и трансляторы.

2. Язык С. Алфавит языка, лексемы языка. Типы констант, определение именованных констант в программе.

3. Идентификаторы, правила их определения. Элементарные типы данных языка С, определение переменных в программе, инициализация переменных.

4. Целые типы данных в языке С. Их размеры, диапазон значений, формат хранения в памяти компьютера.

5. Вещественные типы данных в языке С. Их размеры, диапазон значений, формат хранения в памяти компьютера.

6. Символьный тип данных, его свойства. Строковый тип, его свойства и особенности. Стандартные функции для работы со строковыми данными.

7. Выражения в языке С. Операции (арифметические, сравнения, логические, побитовые). Условная операция. Оператор присваивания и операции присваивания в языке С.

8. Условные операторы языка С. Оператор безусловного перехода.

9. Операторы цикла языка С. Вспомогательные операторы.

10. Определение функций в языке С. Способы передачи параметров в функцию, использующиеся в языке С. Функция main(), её особенности.

11. Стандартная библиотека С. Функции ввода-вывода, форматный ввод-вывод.

12. Работа с файлами в языке С. Функции работы с файлами. Буферизованный ввод-вывод.

13. Области видимости переменных в языке С. Локальные, глобальные, статические переменные. Области видимости функций.

14. Указатели в языке С. Их определение в программе, приёмы их использования. Указатели на функции.

15. Массивы в языке С, одномерные и многомерные. Инициализация массивов. Связь массивов с указателями. Передача массивов в качестве параметров в функцию. Строки как массивы символов.

16. Структуры и объединения в языке С. Их определение в программе и работа с ними. Указатели на структуры и объединения. Перечислимый тип данных.

17. Препроцессор языка С. Основные директивы.

18. Динамическая память. Её отличие от статической. Работа с динамической памятью в языке С.

19. Структуры данных. Определение. Простые структуры данных: таблица, стек, очередь. Их назначение и использование в программе.

20. Динамические структуры данных. Списки, деревья.

21. Куча (пирамида). Хэш-таблица. Использование этих типов данных.

22. Проблемы при разработке больших программ. Назначение и основные черты объектно-ориентированного программирования на примере C++.

23. Отличия C и C++, не относящиеся к объектно-ориентированному программированию.

24. Классы в C++, инкапсуляция данных, поля класса, методы класса. Конструкторы и деструкторы. Конструкторы копирования.

25. Полиморфизм. Сигнатура функции (метода). Перегрузка функций в классах. Виртуальные функции.

26. Наследование. Виды наследования, его назначение и использование.

27. Работа с указателями на объекты. Ссылки. Передача объектов в качестве параметров в функцию и в качестве возвращаемого значения.

28. Взаимодействие программы с операционной системой. Интерфейс командной строки. Графический интерфейс и событийно-управляемое программирование.