**Теория из лекций преподавателя Карпова  
Основы программирования 1 семестра 1 курса**

*Сами лекции:*[*https://drive.google.com/drive/folders/1\_ktSS4AKFrvskmYfAKKxM8vE82vyNYRe?usp=sharing*](https://drive.google.com/drive/folders/1_ktSS4AKFrvskmYfAKKxM8vE82vyNYRe?usp=sharing)

Некоторые пункты вопросов могут дублироваться, так как вся группа составляла документ на основе перечня вопросов на экзамен.

В некоторых задачах дан таймкод на видеолекции преподавателя Карпова Д.А.

**=====================================**

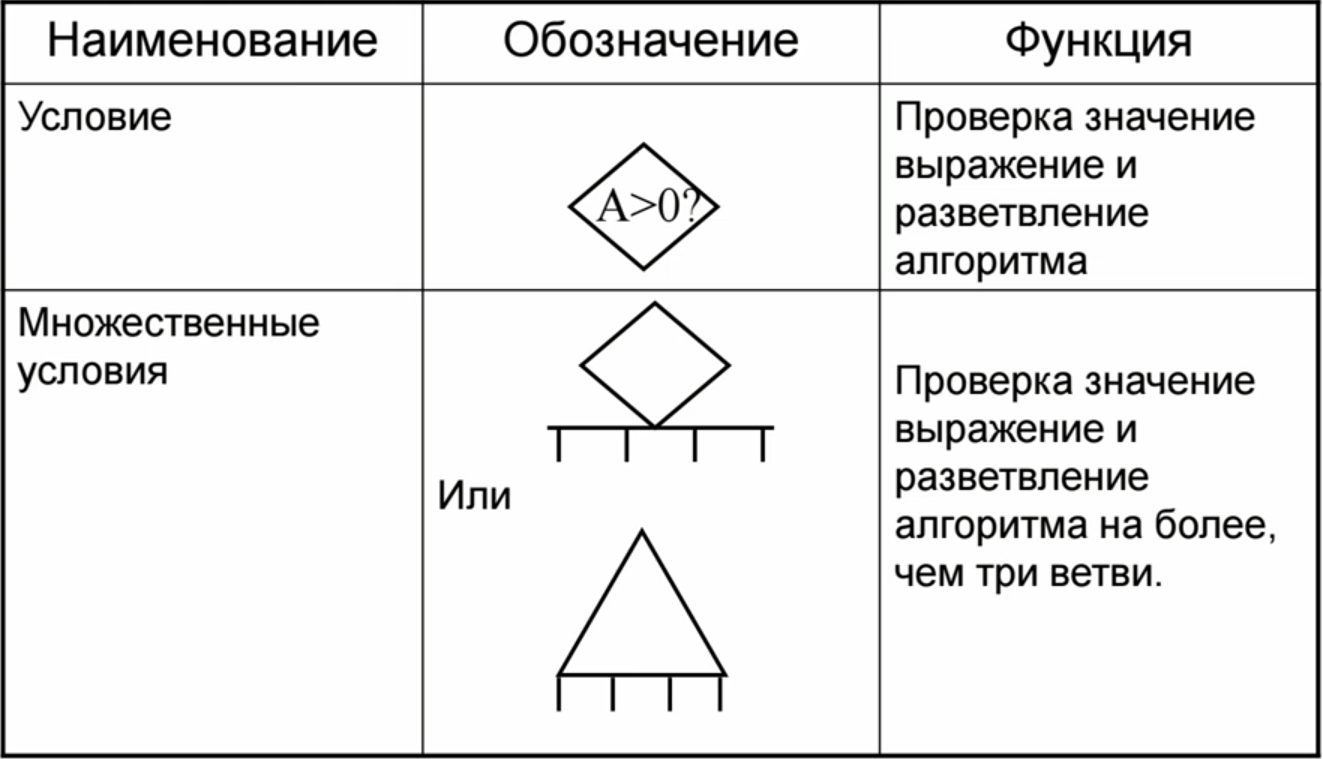
**Задача 1**

Алгоритм. Способы представления алгоритмов. Язык блок-схем.

Лекция №1. time code: 0:00

1. Алгоритм - это последовательность действий, которые необходимо выполнить для решения определенной задачи или получения результата.
2. Алгоритмы могут представляться с помощью блок-схем или языка структурного программирования.
3. Язык блок-схем глазами Карпова







**Задача 2**Алгоритм. Способы представления Алгоритмов. Язык структурного программирования.

**Алгоритм** – это совокупность чётко определенных действий, приводящих к достижению результата за конечное число шагов.

Свойства алгоритмов:

1) Дискретность (от лат. discretus — разделенный, прерывистый) – это разбиение алгоритма на ряд отдельных законченных действий (шагов).

2) Детерминированность (от лат. determinate — определенность, точность) - любое действие алгоритма должно быть строго и недвусмысленно определено в каждом случае. Например, алгоритм проезда к другу, если к остановке подходят автобусы разных маршрутов, то в алгоритме должен быть указан конкретный номер маршрута 5. Кроме того, необходимо указать точное количество остановок, которое надо проехать, скажем, три.

3) Конечность – каждое действие в отдельности и алгоритм в целом должны иметь возможность завершения.

4) Массовость – один и тот же алгоритм можно использовать с разными исходными данными.

5) Результативность – алгоритм должен приводить к достоверному решению.

Различают три основных вида алгоритмов:

1) линейный алгоритм

2) разветвляющийся алгоритм

3) циклический алгоритм

**Линейный алгоритм** – это алгоритм, в котором действия выполняются однократно и строго последовательно.

**Разветвляющийся алгоритм** – это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий.

**Циклический алгоритм** – это алгоритм, команды которого повторяются некое количество раз подряд.

Способы представления алгоритмов:

1) словесный (запись на естественном языке)

2) псевдокод (полуформализованное описание алгоритмов на условном алгоритмическом языке, включающего в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.)

3) графический (изображение из графических символов – блок-схема)

4) программный (тексты на языках программирования – код программы)

Структурное программирование — парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков. Концептуализирована в конце 1960-х — начале 1970-х годов на фундаменте теоремы Бёма — Якопини, математически обосновывающей возможность структурной организации программ, и работы Эдсгера Дейкстры «О вреде оператора goto» (англ. g*oto considered harmful*).

В соответствии с парадигмой, любая программа, которая строится без использования оператора goto, состоит из трёх базовых управляющих конструкций: последовательность, ветвление, цикл; кроме того, используются подпрограммы. При этом разработка программы ведётся пошагово, методом «сверху вниз».

**Задача 3**   
Язык структурного программирования. Принципы структурного программирования. Методы разработки программ.

Язык структурного программирования

Разработан в 1890 Э.В.Дейкстрой

**Принципы структурного программирования**

-Отказ от оператора безусловного перехода (Goto).

-3 базовых конструкции действие, условие, цикл.

-Возможность вкладывания структур в друг друга в произвольной форме в неограниченном количестве.

-Повторяющиеся фрагменты желательно оформить в качестве подпрограммы (если эта подпрограмма используется часто).

-Логически законченные группы желательно оформить в блоки.

-Все перечисленные конструкции должны иметь один вход и один выход.

**Методы СП**

“Сверху-вниз”

При разработке программы создаём управляющий алгоритм используя функции “заглушки” после чего реализуем эти функции “заглушки” в которых можем использовать новые функции “заглушки”. И так мы делаем пока не дойдём до набора базовых конструкций.

“Снизу-вверх”

Предполагает обратный путь: сначала реализуем элементарные функции из которых собираем более сложные конструкции. Используется чаще так как уже существует много библиотек содержащих готовые функции

**Задача 4  
Программа. Программирование. Компиляция. Трансляция. Интерпретация. Определения. Отличия друг от друга.**

**Компьютерная программа** –это алгоритм, реализованный на каком-либо языке программирования или языке машинных команд(инструкций).

Стандарты также дают следующие определения:

ISO/IEC 2382-1:1993

01.05.01 Программа;

Компьютерная программа Синтаксический блок, соответствующий правилам конкретного Языка программирования (01.05.01) и состоящий из Объявлений (15) и Утверждений (15) или Инструкций (07), необходимых для выполнения определённых функций, задач или проблем.

**Программирование** –реализация алгоритма в виде набора команд или инструкций каком-либо языке программирования или языке машинных команд(инструкций).

Стандарт ISO/IEC 2382-1:1993:

01.05.03 Программирование

Проектирование, написание, модификация и тестирование Программ(01.05.01).

**Компиляция** – преобразование программы, написанной на языке программирования, в набор машинных команд или инструкций.

**Трансляция –** преобразование программы, написанной на одном языке программирования, в программу, написанную на другом языке программирования.

**Интерпретация –** построчный анализ, трансляция, компиляция и выполнение исходного кода программы.

**Задача 4**

Программа. Программирование. Компиляция. Трансляция. Интерпретация. Определения. Отличия друг от друга.

Лекция №1. time code: unknown

1. Компьютерная программа - алгоритм, реализованный на каком-либо языке программирования или языке машинных команд (инструкций).
2. Программирование - реализация алгоритма в виде набора команд или инструкций какого-либо языка программирования или языка машинных команд (инструкций).
3. Компиляция - преобразование программы, написанной на языке программирования, в набор машинных команд или инструкций.
4. Трансляция - преобразование программы, написанной на одном языке программирования, в программу, написанную на другом языке программирования.
5. Интерпретация - построчный анализ, трансляция, компиляция и выполнение исходного кода программы.
6. Компиляция работает быстрее интерпретации, но не будет воспроизводить код, пока хоть где-то есть ошибка, также не проводит трансляцию. Интерпретация выполняет трансляцию и будет выполнять программу до тех пор, пока не столкнётся с ошибкой.

**Задача 5**

Основные типы данных и их характеристики.

Типы данных  
1**.Целочисленный - int (integer)**Хранит только целые числа.  
Имеет модификаторы:  
short - уменьшенный диапазон чисел и объем занимаемой памяти  
long - увеличенный диапазон чисел и объем занимаемой памяти  
signed (знаковый) - поддерживает отрицательные и положительные числа. Обычно не используется, так как подразумевается по умолчанию.  
unsigned (беззнаковый) - за счет отказа от отрицательных чисел увеличивается диапазон представления положительных чисел. Надо использовать очень осторожно  
2. **Вещественный ( с плавающей точкой) - float,double**  
Несмотря на наличие в стандарте нескольких вариантов представления чисел с плавающей точкой, наибольшее распространение получили числа в формате одинарной( float) и двойной точности (double)  
В результате вычислений могут происходить искажения целых чисел ( напр. 1,399999999999999 вместо 1,4) или потеря точности  
3. **Символьный - char**  
Формально является числом, который при обработке средствами ввода-вывода воспринимается в качестве кодового значения для таблицы символов.  
Несколько символов могут объединяться в один набор, называемый строкой. Строковый тип чаще всего обозначается как string  
4. **Логический (boolean) - bool**  
Также формально является числом, который при обработке воспринимается как имеющий два значения: 0 ("ложь", false) и 1 ("истина", true). Любое другое значение может также восприниматься как "истина".  
Самый "не экономный" тип данных, так как для всего двух значений расходуется объем памяти в 1 байт , достаточный для хранения 256 значений или 8 независимых логических флагов  
5. **Указатель (pointer)**  
Содержит не само значение( число , символ и т.п.), а адрес в памяти, где оно хранится, то есть реализует косвенную адресацию.  
Косвенная адресация может быть вложенной : указатель на указатель  
Может быть типизированным - то есть указывать только на один конкретный тип данных ( только на символы , только на целые числа и т.п.), и не типизированным - не иметь привязки к типу данных ( также называют обобщенным)  
В отличии от других типов данных, указатель требует внимания и осторожности, так как неверное значение адреса легко может привести к ошибке в работе программы. Неиспользуемый указатель должен быть установлен в значение NULL ( в новых версиях языка С++ используется обозначение nullptr), а перед использованием проверяться на это значения , для исключения ошибок.  
6. **Тип void**  
Хотя и относится к типам данных ,не имеет своей памяти и данных собственно не содержит.  
Используется для указания отсутствия данных, если требуется что-то обязательно передавать или для создания нетипизированных указателей   
7. **Структура - struct**  
Составной тип данных, позволяющий объединить в единое целое несколько данных, которые могут быть одного или нескольких типов , в том числе - быть структурами

**Задача 6**

Статистические и динамические массивы. Правила объявления и обращения к элементам. Отличия в организации хранения в памяти.

Ответ:

У статических массивов — **объем выделенной памяти** не меняется в процессе работы программы, у динамических же память выделяется и освобождается программистом самостоятельно с помощью специальных команд в ходе программы.

**Статический массив объявляется следующим образом**:

тип данных имя массива[количество элементов];

Динамический массив объявляется через указатель:

тип данных \* имя массива;

Расположение элементов динамического массива памяти зависит от того, как она была ему выделена! Адресация элементов массива начинается с 0!

**Задача 7**

Префиксные и суффиксные операции. Отличия.

Лекция №2 .time code: 4:40

Различия записи состоит в положении знака операции инкремента (декремента) относительно переменной. Отличаются при совмещении с операцией присваивания; определяют порядок операций: сначала инкремент (декремент), а потом - присвоение, или наоборот.

**Задача 8**  
Арифметические, логические и побитовые операции. Обозначения. Принципы работы.

Над объектами в языке Си могут выполняться различные операции:

1) операции присваивания

2) операции отношения

3) арифметические

4) логические

5) сдвиговые операции

Результатом выполнения операции является число.

## **Операция присваивания:**

Операция присваивания обозначается символом = и выполняется в 2 этапа:

1) вычисляется выражение в правой части;

2) результат присваивается операнду, стоящему в левой части

В случае если объекты в левой и правой части операции присваивания имеют разные типы используется операция явного приведения типа: *объект=(тип)выражение;*

## **Операции отношения**

**Основные операции отношения:**

== эквивалентно – проверка на равенство;

!= не равно – проверка на неравенство;

< меньше;

> больше;

<= меньше или равно;

>= больше или равно.

Операции отношения используются при организации условий и ветвлений. Результатом этих операций является 1 бит, значение которого равно 1, если результат выполнения операции – истина, и равно 0, если результат выполнения операции – ложь.

## **Арифметические операции:**

Основные бинарные операции, расположенные в порядке уменьшения приоритета:

\* умножение;

/ деление;

+ сложение;

- вычитание;

% остаток от целочисленного деления.

**Основные унарные операции:** (не забудьте что ++i и i++ это различные вещи)

++ инкрементирование(увеличение на 1);

- - декрементирование(уменьшение на 1);

- изменение знака.

## **Логические операции:**

Логические операции делятся на две группы:

Условные;

Побитовые.

*Условные* логические операции чаще всего используются в операциях проверки условия “if” и могут выполняться над любыми объектами. Результат условной логической операции:

1 если выражение истинно;

0 если выражение ложно.

Все значения, отличные от нуля, интерпретируются условными логическими операциями как истинные.

**Основные условные логические операции:**

&& – И (бинарная) - требуется одновременное выполнение всех операций отношения

|| – ИЛИ (бинарная) - требуется выполнение хотя бы одной операции отношения

! – НЕ (унарная) - требуется невыполнение операции отношения.

*Побитовые* логические операции оперируют с битами, каждый из которых может принимать только два значения: 0 или 1.

**Основные побитовые логические операции в языке Си:**

& – конъюнкция (логическое И) – бинарная операция, результат которой равен 1 только когда оба операнда единичны (в общем случае – когда все операнды единичны);

| – дизъюнкция (логическое ИЛИ) – бинарная операция, результат которой равен 1 когда хотя бы один из операндов равен 1;

~ – инверсия (логическое НЕ) – унарная операция, результат которой равен 0 если операнд единичный, и равен 1, если операнд нулевой;

^ – исключающее ИЛИ – бинарная операция, результат которой равен 1, если только один из двух операндов равен 1 (в общем случае если во входном наборе операндов нечетное число единиц).

## **Сдвиговые операции:**

Операции арифметического сдвига применяются в целочисленной арифметике и обозначаются как:

>> – сдвиг вправо;

Арифметический сдвиг целого числа вправо >> на 1 разряд соответствует делению числа на 2.

<< – сдвиг влево.

Арифметический сдвиг целого числа влево << на 1 разряд соответствует умножению числа на 2

**Задача 9**

Операции сравнения и условные операции.

Операции сравнения:

Равно ли ==

Не равно !=

Больше >

Меньше <

Больше или равно >=

Меньше или равно <=

Не желательно сравнивать на равенство и не равенство числа с плавающей точкой. Лучше использовать сравнение модуля разности сравниваемых чисел с коэффициентом погрешности.

Условные операции:

if (условие)

операциЯ если условие выполняется

else

операциЯ если условие НЕ выполняется

if (условие)

{

операциИ если условие выполняется

}

else

{

операциИ если условие НЕ выполняется

}

**Задача 10**

Способы организации множественных условий

1.Вложенные конструкции:

If (условие 1)

…

else if (условие 2)

…

else if (условие 3)

…

else

…

2.Оператор множественного выбора:

switch(переменная)

{

Case значение1:

Действия если значение1

Case значение2:

Действия если значение2

Case значение3:

Действия если значение3

default:

Действия если ничего не подошло

}

Если же мы хотим, чтобы программа не проходила по всем “case” мы можем вставить в программу break, которая позволит прервать нам цикл, после выполнения одного из условий пример:

switch(переменная)

{

Case значение1:

Действия если значение1

break

Case значение2:

Действия если значение2

break

Case значение3:

Действия если значение3

break

default:

Действия если ничего не подошло

}

**Задача 11**

Циклы. Счетный цикл. Цикл с условием.

Лекция №2. time code: 20:00

Цикл - код, позволяющий выполнять некоторые операции неограниченное количество раз (написано точь в точь, как диктовал Карпов)

1.1) Счётный цикл применяется при заранее известном количестве операций. Чаще всего при обработке массивов данных.

1.2) Структура счетного цикла:

for (начальное значение; условие выполнения цикла; действие, выполняемое в каждом цикле)

операция в цикле;

P.S. По умолчанию выполняется только 1 операция, следующая за командой цикла. Для нескольких операций используются фигурные скобки {}

2.1) Цикл с условием предназначен для выполнения действий в теле цикла до тех пор, пока условие истинно (возможно прерывание выполнения цикла операторами break и continue). Бывают с предусловием (while, for), а также с постусловием (do-while)

2.2) Структура циклов с условиями.

2.2.1) Цикл while:

while (условие выполнения цикла)

операция в цикле;

2.2.2) Цикл for:

for (начальное значение; условие выполнения цикла; действие, выполняемое в каждом цикле)

операция в цикле;

2.2.3) Цикл do-while:

do

операция в цикле;

while (условие выполнения цикла)

P.S. По умолчанию выполняется только 1 операция, следующая за командой цикла. Для нескольких операций используются фигурные скобки {}

**Задача 12**

Циклы. Циклы с предусловием и постусловием.

1) Условный цикл с предусловием while.

Универсальный. Применяется если не захотели или не

смогли использовать цикл for.

while(условие выполнения цикла)

операция в цикле;

По умолчанию выполняется только 1 операция,

следующая за командой цикла. Для нескольких операций

используются фигурные скобки { }.

Операция в теле цикла или в его условии должна

ОБЯЗАТЕЛЬНО влиять на его условие, иначе программа

зависнет.

ИСКЛЮЧЕНИЕ — бесконечный цикл while(true), который

применяется СОЗНАТЕЛЬНО!

2) Условный цикл с постусловием do while.

Применяется если надо сперва выполнить действие, а

потом убедиться, что оно было выполнено правильно.

do

операция в цикле;

while(условие выполнения цикла);

По умолчанию выполняется только 1 операция,

следующая за командой цикла. Для нескольких операций

используются фигурные скобки { }.

Операция в теле цикла или в его условии должна

ОБЯЗАТЕЛЬНО влиять на его условие, иначе программа

зависнет.

ИСКЛЮЧЕНИЕ — НЕТ. Надо использовать только

бесконечный цикл while(true).

При использовании условных циклов надо помнить о том,

что факт выполнения условия может быть не всегда

реализуем и/или долго ожидаем. Поэтому имеет смысл

заводить контрольную переменную-счетчик для аварийного

завершения работы цикла.

int warden=0;

while ((условие)&&(warden<1000))

{

...

warden++.

}

if (warden>=1000)

{

// цикл прервался ДО выполнения условия

}

**Задача 13**

Проверка вводимых данных. Основные проблемы и способы их решения.

Лекция №3. time code: вся лекция

1. Операции с данными.

Проблема: ограничения операндов на ОДЗ, выход за которую может возникнуть ошибка

Решение: все операнды функций, имеющих ограничения на область допустимых значений (ОДЗ), должны в обязательном порядке проверяться на соответствие ОДЗ.

1. Операции с массивами и памятью

Проблема: память данных и команд в современных ЭВМ в большинстве случаев не разделена и слабо защищена от попадания в чужую область памяти.

Решение: необходимо контролировать выход индексных переменных за границы. При копировании блоков памяти необходимо проверять как размер копируемого блока, так и размер области получателя, выбирая минимальное из них или отказываясь от копирования при превышении допустимого объема.

1. Знаковые и беззнаковые переменные.

Проблема: различная интерпретация одних и тех же значений разрядной сетки числа в одном выражении.

Решение: перед вычислениями знаковые и беззнаковые числа должны быть принудительно приведены к одной форме.

**Задача 14**

Исключения. Определение. Виды. Назначение. Способы использования

Лекция №4. time code: 0:00

1. Исключение - это событие при выполнении программы, которое приводит к ее ненормальному или неправильному поведению.
2. Существует два вида исключений:

2.1. Аппаратные (структурные, SE-Structured Exception) и программные.

1. Назначение: альтернативная проверка вводимых данных.
2. [не очень понял суть вопроса] Для обработки исключительных ситуаций используются конструкции:

4.1. try - секция, содержащая код с потенциальной ошибкой.

4.2. throw - генерация программного исключения до события с передачей в блок обработки идентификационного параметра.

4.3. catch - блок обработки исключительных ситуаций. Получает или стандартный системный код ошибки, или идентификационный параметр, переданный с помощью throw.

**Задача 15**

Сигналы. Назначение. Сложности использования

**Сигнал** - это механизм который позволяет сообщить программе об ошибке (программа пытается выполнить несуществующие команды), после получения сигнала операционная система закроет программы и вы можете только минимизировать ущерб.

Кроме перехвата исключений программа может настраивать свои обработчики сигналов критических ситуаций для ОС.

Сигналы позволяют обмениваться информацией между различными программными продуктами.

Однако надо помнить, что для ряда сигналов невозможно продолжение работы программы в любом случае, можно лишь <<аккуратно>> её завершить, до того, как ОС <<рассердится окончательно>>.

То есть, например, сигнал критического нарушения обращения к памяти, сигнал принудительного завершения(такие сигналы есть разной <<силы>>: от <<вежливой просьбы завершиться>>, до <<принудительного уничтожения без предупреждения>>), перехватить и обработать можно, но отменить не получиться.

**Задача 16**

Алгоритмы обработки. Поиск максимума и минимума.

Поиск максимума и минимума в массиве может осуществляться следующим образом…

Чтобы найти **минимум**,нужно создать переменную с значением, которое точно будет превосходить самое максимальное значение, которое может встретиться. Например, если мы знаем, что в массиве не может встретиться число > 127, то можно присвоить переменной значение 127. Далее будем сравнивать каждый элемент массива с текущим минимумом, и если окажется, что элемент меньше, присвоим текущему минимуму значение элемента. Тогда алгоритм обработки на языке C++ для **статитического** массива будет следующий:

int findMinimum(int n, int a[])

{

int minimum = 127;

*for* (int i = 0; i < n; i++)

{

*if* (a[i] < minimum) minimum = a[i];

}

*return* minimum;

}

int main()

{

int a[] = {89, -55, 34, -21, 13, -5, 127};

int minimum = findMinimum(7, a);

cout << minimum << "\n";

}

Чтобы найти **максимум**,нужно создать переменную с значением, которое точно будет меньше самого минимального значения, которое может встретиться. Например, если мы знаем, что в массиве не может встретиться число < -127, то можно присвоить переменной значение -127. Далее будем сравнивать каждый элемент массива с текущим максимум, и если окажется, что элемент больше, присвоим текущему максимуму значение элемента. Тогда алгоритм обработки на языке C++ для **статического** массива будет следующий:

int findMaximum(int n, int a[])

{

int maximum = -127;

*for* (int i = 0; i < n; i++)

{

*if* (a[i] > maximum) maximum = a[i];

}

*return* maximum;

}

int main()

{

int a[] = {89, -55, 34, -21, 13, -5, -127};

int maximum = findMaximum(7, a);

cout << maximum << "\n";

}

**Замечание.** Можно было бы присвоить -INF и INF, но для экономии ресурсов и безопасности выполнения, в случае когда нам известно пограничное значение, лучше использовать данный способ присваивания.

**Задача 17**

Алгоритмы обработки. Сортировка "пузырек".

Сразу хочется заметить, что сортировка “пузырьком” (bubble sort) далеко не самый эффективный способ сортировки, но достаточно простой в реализации. Суть алгоритма заключается в том, что мы проходим по массиву столько раз, сколько у него элементов - 1, и каждый раз сравниваем пары соседних чисел. Если текущее число больше следующего, то меняем их местами. Для этого в C++ можно завести переменную temp. Для оптимизации алгоритма нужно на каждом проходе уменьшать его длину на 1, т.к. в самый конец у нас будут сдвигаться наибольшие элементы массива. В итоге сложность O(n^2). На C++ данный алгоритм будет выглядеть следующим образом:

int a[] = {6, 1, 5, 2, 4, 3};

int n = 6;

void bubbleSort()

{

*for* (int i = 0; i < n - 1; i++)

{

*for* (int j = 0; j < n - i - 1; j++)

{

*if* (a[j] > a[j + 1])

{

int temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

}

}

int main()

{

bubbleSort();

*for* (int i = 0; i < n; i++) cout << a[i] << " ";

cout << "\n";

}

**Задача 18**

Алгоритмы обработки. Гномья сортировка.

Лекция №4. time code: 28:00

Идея алгоритма - расстановка элементов массива по возрастанию. Элемент сравнивается с соседним. Если они стоят правильно - переход к следующей паре. Если нет - меняем их местами и возвращаемся назад, сравнивая с предыдущими.

Алгоритм:

i=0

ПОКА (i<n)

ЕСЛИ (i==0) ИЛИ (A[i-1]<=A[i]) ТО

i=i+1

ИНАЧЕ

j=i-1

врем=A[i]

A[i]=A[j]

A[j]=врем

i=j

**Задача 19**Алгоритмы обработки.Сортировка расчёской.

Сортировка расчёской (*Comb sort*) – алгоритм сортировки массива, является улучшенным вариантом [сортировки пузырьком](https://programm.top/c-sharp/algorithm/array-sort/bubble-sort/)

## Принцип работы алгоритма сортировки расческой:

Основная идея сортировки расческой в использовании большего расстояния между сравниваемыми элементами. При этом первоначально необходимо брать большое расстояние, и постепенно уменьшать его, по мере упорядочивания данных вплоть до единицы. Изначально сравнивается первый и последний элемент массива, и на каждой итерации уменьшается разрыв между элементами на *фактор уменьшения*. Итерации продолжаются до тех пор, пока разность индексов больше единицы, а замет массив сортируется пузырьковой сортировкой.

Оптимальное значение фактора уменьшения равно *1/(1-e^-φ) ≈ 1.247*, где *е* – основание натурального логарифма, а *φ* – золотое сечение.

**Задача 20**Алгоритмы обработки. Быстрая сортировка.

Быстрая сортировка, она же сортировка Хоара, quicksort, qSort — алгоритм, разработанный английским информатиком, Чарльзом Хоартом во время его работы в МГУ в 1960 году.

Является одним из самых быстрых универсальных алгоритмов, хотя является улучшением сортировки пузырьком. Основан на стратегии «разделяй и властвуй».

Общий принцип:

В массиве выбирается опорное значение. Стратегия выбора не важна в общем случае, хотя может сократить время сортировки. Массив переупорядочивается так, чтобы слева от опорного элемента были элементы массива со значением меньше или равным опорному, а справа больше. Для каждой части массива операция разделения повторяется снова и снова.

Обобщенный алгоритм быстрой сортировки:

1) Выбираем опорный элемент.

2) Разделяем массив.

2.1) Выбираем два индекс l(left) и r(right), в которые заносятся значения правой и левой границ массива.

2.2) Индекс l увеличивается пока l-ый элемент не будет больше или равен опорному

2.4) Если l равен r, то операция разделения закончена иначе возвращаемся к шагу 2.2

3) Рекурсивно упорядочиваются полученные в результате разделения подмассивы. База рекурсии — пустой массив или массив из одного элемента.

void qsort (int \* arr, int left, int right)

{

int l = left, r = right;

int m = arr[(l + r) / 2];

while (l <= r)

{

while (arr[l] < m)

l++;

while (arr[r] > m)

r--;

if (l <= r)

{

int tmp = arr[l++];

arr[l++] = arr[r--];

arr[r--] = tmp;

}

}

if (left < r)

qsort (arr, left, r);

if (right > l)

qsort (arr, l, right);

}

**Задача 21**

Рекурсия. Определение. Ограничения применения.

Лекция №4. time code: 33:50

1. Рекурсией называется вызов функцией самой себя с некоторым изменением входных параметров.
2. Во-первых, для использовании рекурсии необходимо наличие базы рекурсии - то есть значения должны зависеть только от входного параметра и позволять выйти из рекурсии. Во-вторых, ограничивающим фактором применения рекурсии является объем специальной области памяти, где хранится контекст и адреса возврата при вызове функции. При большой глубине рекурсии эта память может быть переполнена, что приведёт к ошибке выполнения программы.

**Задача 22**

Подпрограммы. Принцип работы подпрограмм. Отличия процедур и функций друг от друга.

Подпрограммы:

Для упрощения разработки, построения и использования программ и программных библиотек готовые конструкции, решающие какую либо конкретную задачу принято объединять в подпрограммы.

Желательно формировать подпрограммы так, чтобы их можно было изменять с минимальным влиянием на другие подпрограммы.

Основные варианты реализации подпрограмм:

- процедуры

- функции

Принцип работы подпрограмм:

1. В момент обращения к подпрограмме сохраняются текущие значения служебных переменных и регистров процессора в специальной области памяти (организованной как стек)

2. В эту же область сохраняется адрес следующей за вызовом подпрограммы команды (адрес возврата)

3. Происходит передача управления первой команде подпрограммы

4. Работа подпрограммы

5. После завершения работы подпрограммы происходит восстановление сохраненной служебной информации и передача управления команде, чей адрес был сохранен на шаге 2

Шаги 1 и 2 называются сохранением контекста

Шаг 5 - восстановление контекста

Функции отличаются от процедур фактом возвращения какого-либо результата по своему имени. То есть, процедура выполняет набор внутренних команд и передает управление следующей за своим вызовом команде. А у функции следующей за ней командой является команда присваивания, причем присваиваемое значение формируется в теле функции.

**Задача 23**

**Правила объявления функций и возвращения результата её работы.**

Формальное определение функции выглядит так:

тип имя\_функции(параметры)

{

инструкции

}

Первая строка представляет заголовок функции. Вначале указывается возвращаемый тип функции.

Затем идет имя функции, которое представляет произвольный идентификатор. К именованию функции применяются те же правила, что и к именованию переменных.

После имени функции в скобках идет перечисление параметров. Функция может не иметь параметров, в этом случае указываются пустые скобки.

После заголовка функции в фигурных скобках идет тело функции, которое содержит выполняемые инструкции.

Для возвращения результата функция применяет оператор **return**. Если функция имеет в качестве возвращаемого типа любой тип, кроме void, то она должна обязательно с помощью оператора return возвращать какое-либо значение.

Пример функции:

float MyFunction(float number)

{

float result = number\*number;

return result;

}

**Задача 24**

Виды параметров функций. Правила объявления. Назначение, возможности и *ограничения* каждого вида параметров.

Лекция №4. time code: 22:10

1. Передача значений (то есть данные обратно не передаются, можно указывать константы при вызове функции):

тип параметра имя параметра

1. Передача адреса (параметр-ссылка) (то есть функция имеет доступ к данным вызываемой программы и может их менять, указывать константы при вызове функции нельзя):

тип параметра &имя параметра

1. Передача указателя (параметр-указатель) (то есть передается адрес переменной, по которому надо читать и писать данные.Если этот адрес изменить в функции, основная программа будет работать с прежним адресом):

тип параметра\* имя параметра

1. Передача адреса указателя (то есть передается адрес ячейки, содержащий указатель переменной, по которому надо читать и писать данные. Если этот адрес изменить в функции, основная программа получит измененный адрес):

тип параметра\* &имя параметра

Количество параметров функции и аргументов вызова должно совпадать. Но иногда часть параметров может задаваться по умолчанию. Параметры по умолчанию должны находиться в конце перечня параметров. Значения указываются один раз либо в заголовке функции, либо в ее объявлении.

**Задача 25**

Параметры функции «по умолчанию». Правила объявления. Назначение, возможности и ограничения.

Параметр по умолчанию (или «необязательный параметр») – это параметр функции, который имеет определенное (по умолчанию) значение. Если пользователь не передает в функцию значение для параметра, то используется значение по умолчанию. Если же пользователь передает значение, то это значение используется вместо значения по умолчанию.

**Правила объявления:**

1) Все параметры по умолчанию в прототипе или в определении функции должны находиться справа.

2) Значения указываются один раз либо в заголовке функции, либо в её объявлении.

3) Если параметров несколько, то должны указываться все параметры, стоящие в объявлении до нужного параметра.

+ Если имеется более одного параметра по умолчанию, то самым левым параметром по умолчанию должен быть тот, который с наибольшей вероятностью (среди всех остальных параметров) будет явно переопределен пользователем.

**Задача 26**

Алгоритмы поиска. Основные виды. Принципы построения. Достоинства и недостатки.

**Алгоритмы поиска**  
1.**Последовательный поиск**  
Может проводиться как в упорядоченном наборе данных(массиве), так и в неупорядоченном  
При поиске в неупорядоченном массиве происходит сравнение искомого элемента с содержанием элементов массива  
Если нужен только один элемент, то поиск идет до первого совпадения или до конца массива. Если все элементы - то до конца массива  
При поиске в упорядоченном массиве , поиск осуществляется до первого элемента, превышающего искомое значение.  
Но тогда массив данных надо постоянно поддерживать в упорядоченном состоянии

2.**Поиск по ключам**  
заменяет поиск содержимого ячейки поиском цифрового ключа, который может быть как порядковым номером элемента в массиве , так и цифровой комбинацией, сформированной на основе блока данных.  
В частном случае, когда ключ это номер элемента в массиве поиск проводить не нужно , но возникает проблема удаления элементов : "Что делать с пустыми ячейками?"  
Занимать? - Могут быть ошибки, если кто-то обратится к ячейке, думая, что там старые значения.  
Оставлять пустыми? - Потеря памяти

3.**Бинарный поиск**В упорядоченном массиве не обязательно искать ключи и/или данные с начала. Можно применять метод "дихотомии", то есть половинного деления

Обращаемся к среднему элементу массива , если он искомый - поиск завершается.

Если меньше - находим элемент по середине между проверенным и концом массива и проверяем его

Если больше - находим элемент по середине между началом массива и проверенным и проверяем его

И так далее

Интервал уменьшается до обнаружения искомого или когда остался один элемент и он нам не подходит ( результат поиска false )  
Его достоинство - гарантированное время поиска, не превышающее Log2(N) + 1 операций

Для упрощения бинарного поиска массивы данных могут формироваться на основе бинарных деревьев , т.е. сами данные хранятся в виде , упрощающем поикс

Но использование таких деревьев требует затрат на их поддержание в корректной форме при каждых изменениях ( добавление , удаление) в наборах данных

**Задача 27**

Определение для языка С++. Отличие от потоков. Виды потоков.

**Файл** - в C++ это средство взаимодействия с программой.

Файлы являются логической концепцией, применимой ко всему, с чем взаимодействует проФайлграмма, начиная от дисковых файлов оканчивая терминалами.

**C ++** - это статически типизированный, скомпилированный, универсальный, учитывающий регистр, язык программирования свободной формы, который поддерживает процедурное, объектно-ориентированное и общее программирование.

**Отличие:**

В системе ввода/вывода C++ все потоки одинаковы, а файлы - нет.

**Виды потоков:**

**По типу данных:** -текстовые; -двоичные(блочные).

**По организации доступа:** -с последовательным доступом; -с произвольным доступом.

**По уникальности информации:** -появляющаяся однократно; -с возможностью неоднократного обращения.

**Текстовые потоки** - это последовательность символов, формируемых из кодов на основе специальных таблиц символов. В текстовых потоках некоторые символы могут преобразоваться согласно требованиям среды.

**Двоичный поток** - это последовательность байт, имеющих однозначное соответствие с байтами во внешнем устройстве. То есть преобразование кодов и символов не производится. Также число байт, записанных или прочитанных из внешнего устройства, совпадает с числом во внешнем устройстве. Тем не менее при физической записи на устройство может добавляться некоторое количество нулевых байт к двоичному потоку. Блочные такие потоки называют, потому, что они используются в режиме чтения информации блоками по несколько байт, а не отдельными байтами.

**Потоки с последовательным доступом** - это связь с файлами (устройствами), которые способны принимать, передавать и записывать только информацию и не имеют адреса считываемых или записываемых данных.

**Потоки с произвольным доступом** - это связь с файлами (устройствами), которые способны принимать, передавать и записывать информацию из/в произвольного места файла, то есть имеют адрес считываемых или записываемых данных.

**Потоки с информацией, появляющейся однократно** - это связь с файлами (устройствами), которые не хранят информацию и не могут её повторить. В таком случае, как правило, речь идет о подвидах устройств с последовательным доступом.

**Потоки с информацие с возможностью неоднократного обращения** - это связь с файлами (устройствами), которые хранят информацию и могут её повторить в режиме произвольного или последовательного доступа. В последнем случае информация будет повторятся с самого начала.

**Задача 28**

Файл. Стандартные операции с файлами.

**1.Открытие файла** — обеспечивает связь между файлом и потоком. Формирует указатель для потока, который будет использоваться для обращения при любых других операциях с ним.

Ключевым словом для такой операции является open.

**2.Закрытие потока** — удаляет связь между между файлом и потоком . Предварительно завершает все незаконченные операции из буфера, освобождает память. Обнуляет указатель для потока.

Ключевым словом для такой операции является сlose.

Чтение данных — обеспечивает чтение блока данных или символов из потока в указанную память, массив или другую структуру. Указатель смещается на считанный размер памяти (если поддерживается произвольный доступ).

Ключевым словом для такой операции является read.

**3.Стандартные операции с файлами Запись данных** — обеспечивает запись блока данных или символов из памяти, массива или другой структуры в поток. Указатель смещается на записанный размер памяти (если поддерживается произвольный доступ).

Ключевым словом для такой операции является write.

**4.Получение текущей позиции в файле** —возвращает значение в количестве символов, байта или (реже) блоков от начала файла. То есть сколько надо их последовательно прочитать, чтобы оказаться в текущей позиции файлового с произвольным доступом. Указатель не смещается. Ключевым словом для такой потока.

Только для потоков операции является роs.

**5.Перемещение в заданную позицию в файле** — перемещает указатель на указанное количество символов, байт или (реже) блоков. Перемещение может производиться от начала файла вперед, от текущей позиции вперед и назад, и от конца файла назад. Только для потоков с произвольным доступом.

Ключевым словом для такой операции является seek.

**6.Принудительная запись буфера** —производит принудительный перенос информации из буфера в файл, не зависимо от количества накопленной там информации, размера буфера, времени предыдущего переноса и т.п. Хотя все изменения из буфера запишутся в файл по команде его закрытия или по завершению приложения в нормальном режиме, важные данные стоит принудительно сбрасывать в файл, если есть вероятность аварийного завершения программы.

Ключевым словом для такой операции является flush.

**7.Стандартные флаги файла**

При работе с любыми файлами и потоками может использоваться признак конца файла ЕOF — например при чтении файла. Иначе придется проверять ошибки и объем реально чтении всего содержимого до конца. считанной информации.

**Задача 29**   
Файл. Режимы доступа к файлам.

**Файл**

Средство взаимодействия с программой. Концепция применимая ко всему с чем взаимодействует программа (и дисковые файлы и терминалы)

Для начала надо открыть файл чаще всего в языках программирования это команда *open*

Пример на QT:

*QFile myfile;*

*myfile.open(режим доступа);*

**Режимы доступа к файлам**

-ReadOnly только чтение (от греха подальше) Маркер позиции на начале файла

-WriteOnly только запись. Маркер позиции на начале. Файл создаётся если файл уже существовал происходит перезапись

-ReadWrite чтение и запись. Если файла не было он создаётся, если был остается без изменений. Маркер позиции на начале

-Append добавление. Добавлять можно только после последней строки или байта. Если файла нет он создается. Маркер позиции на самом конце.

-TextMode текстовый файл – файл связывается с текстовым потоком.

-BinaryMode двоичный режим – файл связывается с двоичным потоком.

-Unbuffered не буферизированный – файл связывается напрямую, без буфера.

Если что-то было открыто это надо закрыть команда *close*

*QT:*

*myfile.close;*

**Задача 30**

Файл. Способы проверки правильности записи и чтения данных в файл/из файла. Контрольные суммы. Хэш-функции.

Лекция №6. time code: 24:05

Файл в C++ - это средство взаимодействия с программой.

1.1) Проверка правильности записи

Самый простой способ проверить на ошибки при записи - сравнить объем записанных данных с переданными на запись. Для более точной проверки можно принудительно сбросить все данные из буфера на носитель, перечитать их и сверить с записываемыми.

1.2) Проверка правильности чтения

Самый простой способ проверить на ошибки при чтении - сравнить объем успешно считанных данных с «‎заказанным» объемом. Для проверки корректности считанных данных необходимо перед использованием убедиться в их допустимости и соответствии предъявляемым требованиям (то есть провести проверку на соответствие типам и ОДЗ)

Если рассматривать случаи сбоев при написании программ чтения и записи или ошибок копирования/хранения/передачи файлов, то одним из вариантов контроля могут быть специальные переменные флаги, записываемые в файл по определённым правилам и проверяемые при чтении. Эти флаги могут помочь найти начало блока данных и, если в них при создании заложили такой функционал, идентифицировать сам блок.

CRC (циклический избыточный код)

Для алгоритма подбираются такие значения степеней полинома и коэффициентов сборки контрольной суммы, чтобы полученный алгоритм был устойчив к искажениям.

2) Другим способом проверки является использование «‎контрольных сумм». Идея данного метода: посчитать и сложить каким-либо способом все данные и сравнить результат первоначального сложения с результатами сложения данных после их передачи и/или чтения. Несовпадение говорит об искажении файлов.

P.S. может выявить только нечетное количество ошибок.

Циклические контрольные суммы - более простой способ, чтобы убедиться в отсутствии ошибок. Содержимое файла интерпретируется как набор числовых данных. Эти числа суммируются так, что при переполнении числами выделенного объема памяти, счет начинается сначала.

3) Хэш-функции представляют собой дальнейшее развитие алгоритмов контроля целостности, использующее пароли, криптографические алгоритмы или однонаправленные функции, значительно снижающие вероятность успешных активных действий по подмене содержимого файла. Также применяется в решении задач поиска, защиты паролей от чтения, обеспечения неоспоримости.

**Задача 31**

Файл. Структуризация данных в файле. Типы. Достоинства и недостатки.

1. Жестко структурированными с — фиксированным

размером файла.

Используется для хранения единого блока данных. Для других блоков создаются другие файлы. Даже если данные в блоке меньше максимального размера, записывается

максимально предусмотренный объем.

+: самое простое в использование

-: сложность поиска

2. Жестко структурированные с фиксированным размером

блока и переменным размером файла.

Используется для хранения нескольких блоков данных

единого типа в одном файле. Размер файла кратен размеру

блока, исходя из чего можно вычислить количество блоков в

файле.

Также легко обратиться к любому блоку.

+:простота при работе

-:данные могут не влтзать в выделенную область памяти или использовать своё пространство не эффективно

3. Жестко структурированные с изменяемым размером

блока.

Каждый блок имеет специальный идентификатор,

позволяющий определить его размер или факт завершения.

Это может быть как идущая перед блоком переменная с его

размером, так и специальный завершающий флаг в конце

блока.

Данные внутри блока обязательно идут в фиксированной

последовательности. Если данных нет, то либо идут нулевые

размеры, либо последовательности конца данных. Вариант со

сразу идущим признаком конца блока больше относится к

следующему типу, так как требует усложненного алгоритма

анализа считываемой информации.

В этом типе нельзя обратиться к произвольному блоку.

+:чёткость структуры

-:нельзя обратиться к произвольному блоку

4. Гибко структурированные со статическим описателем.

В начале файла (или блока) присутствует специальный

блок (раздел), который описывает параметры всех

(следующего) блоков (набора данных).

Внутри каждый блок может иметь как жесткую, так и

гибкую структуру с фиксированным или изменяемым

размером.

Сложность поиска произвольного блока зависит от

организации файла. Может потребовать как чтения только

описателей, так и чтения всего файла.

+:универсальность

-:сложность использования

5. Гибко структурированные с переменным описателем.

Формально имеют жесткий описатель, но он содержит

лишь параметры/правила чтения главного описателя.

Отличается от предыдущего типа тем, что для каждого

описателя блока/записи есть свой описатель с правилами их

чтения, приведенный либо в главном описателе, либо в

фиксированном «мини-описателе» в начале блока/записи.

Если в главном описателе нет прямых адресов

размещения каждого описателя, то переход к произвольной

записи невозможен.

+: ???

-: сложность использования, нельзя перейти к произвольному элементу

**Задача 32**

Динамическое выделение памяти. Правила выделения и освобождения. Операторы C++.

**Динамическое определение памяти** – это способ работы с памятью, когда размер запрашиваемой для обработки данных памяти определяется и изменяется в процессе работы программы. В отличии от статических переменных и массивов, не имеют выделенной памяти под сами данные. Выделяется память только для хранения указателя.

Работа с динамической памятью включает следующие операции:

· Выделение памяти

· Освобождение памяти

Указатель, который указывает на освобожденную память(не проинициализированный указатель), должен быть обязательно установлен в **NULL**.

При работе с указателями надо обязательно их проверять на неравенство **NULL**.

**Основные операторы:**

· **New** – выделение памяти

· **Delete** – для освобождения памяти(освобождает связь между указателем и памятью)

**Пример:**

**\*Переменная\***

int \*pint;

pint = new int;

delete pint;

pint = Null;

**\*Массив\***

double \*massive;

massiv = new double[20];

delete []massiv;

massiv = Null;

Операция new «снимает» одну звездочку:

int \*\*massiv;

massive new \*int[20];

**\*Структуры\***

struct T\_INFO

{

int size;

char \*text;

};

T\_INFO \*info;

info = new T\_INFO;

info -> size = k;

info->text = new char[info->size];

**Копирование:**

**memcpy**(адрес памяти назначения, адрес памяти источникаб размер копируемых данных)

int a[10];

int b[10];

memcpy(&a, &b, 40); //40 получается исходя из размера элемента массива(4\*10)

Может быть применена как к динамическим, так и к статическим типам, а так же структурам. Размер данных в байтах(int = 4 байта)

Размер структуры может быть определен с помощью функции **sizeof**(структура);

Особенности sizeof:

int b[20];

sizeof(b); //80

int \*c;

sizeof(c); //4

c = new int[20];

sizeof(c); //4

**sizeof()** возвращает размер памяти, занимаемый указателем. Размер динамического массива можно посчитать самостоятельно.

Так же надо помнить, что оператор new запрашивает память

заданного объема у ОС, но этой памяти может не быть. Не быть вообще или в виде непрерывного курса. В этом случае появляется исключение, которое надо перехватывать

То есть, по-хорошему, код выделения памяти должен быть в блоке try-catch, например:

try

{

buf = new chat[file\_size]; // запросить память

}

catch(std::bad\_alloc &ba) // не дали

{

fclose(file); // закрыли файл

size = 0; // обнулили счетчик

return(false); // вышли из функции

}

Заполнение памяти:

Заполнение памяти может применяться как к динамическим, так и статическим массивам.

memset(адрес памяти назначения, заполняемые значения, размер заполняемых данных);

int a[10];

int b[10];

memset(&a, 1, 40);

memset(&b, 0, 80); // занулили массив «а»

«Самая опасная функция в мире»

**Задача 33**

Динамическое выделение памяти. Списки. Их достоинства и недостатки.

Динамическое выделение памяти необходимо для эффективного использования памяти компьютера. В C++ для динамического распределения памяти используются операции **new** и **delete**. Хочется заметить, что чаще их используют именно в работе с массивами. Например, чтобы в данном случае выделить память, следует воспользоваться операцией **new**.

int \*dinarr = new int[];

Чтобы высвободить память, нужно применять операцию **delete**.

delete [] dinarr;

Стоит заметить, что высвобождая память, мы её **НЕ УДАЛЯЕМ**. Чтобы удалить память, можно, например, заполнить массив nullptr или NULL.

Преимущества динамического выделения памяти для массивов на лицо. Например, пользователь не знает, сколько элементов он хочет ввести. Тогда ввод можно реализовать циклом do while.

int main()

{

int n = 0;

int \*dinarr = *new* int[n];

string ans = "y";

*do*

{

cout << "\nEnter num: ";

int num; cin >> num;

dinarr[n++] = num;

cout << "\nDo you want to continue?(y/n)\n";

cin >> ans;

}*while* (ans == "y");

*for* (int i = 0; i < n; i++) cout << dinarr[i] << " ";

cout << "\n";

*delete* [] dinarr;

}

Также хочется отметить один существенный недостаток. Объекты массива могут часто копироваться в другие места, а многие указатели становиться недействительными. Если требуется вносить частые изменения в середине динамического массива, то для этого надо использовать более подходящий тип линейной структуры данных.

**Задача 34**  
Препроцессор. Назначение. Директивы препроцессора.

**Препроцессор** — в общем случае программа осуществляющая предварительную обработку какой-либо информации для другой программы.  
 Препроцессор в программировании — программа подготавливающая исходный код к компиляции.  
 В процессе своей работы препроцессор добавляет, изменяет, заменяет и/или удаляет программный код на основании алгоритмов своей реализации и специальных директив.  
 **Директивой препроцессора** называется инструкция специального вида, предназначенная для сообщения препроцессору каким образом необходимо обработать код программы или его часть.  
  
 **Директивы препроцессора**

* Директива начинается с символа «решетка» #, далее один или несколько затем следует ключевое слово, параметров. Точка с запятой НЕ ставится. Основные ключевые слова препроцессора:  
   — include;  
   — define;  
   — undef;  
   — if;  
   — ifdef;  
   — ifndef;  
   — else;  
   — elif;  
   — endif;  
   — error;  
   — warning;  
   — pragma.

**Задача 35**

Препроцессор. Правила создания макросов

#define-объявление констант и макросов

После директивы #define указывается имя макроса, за которым в скобках (без пробелов) параметры, отделенные запятыми и определение макроса, отделенное пробелом.

Пример ввода макроса:

#define round(x) (((x)>=0)?int((x)+0.5):int((x)-0.5)) — макрос.

Параметры макросов лучше писать в скобках, т.к иногда отсутствие скобок приводит к разным результатам.

#define MACRO1(x) x \* x

#define MACRO2(x) ( x ) \* ( x )

int a =2;

int b=3; cout

результат:

MACRO1- 11//2+3\*2+3

MACRO2- 25//(2+3)\*(2+3)

Достоинством использования макросов — является отсутствие вызова функций и, как следствие, расхода ресурсов на сохранение и восстановление контекста.

Недостаток использования макросов — замена осуществляется как есть, без проверок.

Это приводит к:

— затруднению поиска ошибок;(Ошибка будет показываться в строке вызова макроса, а не в строке ввода макроса)

— потери производительности, если макрос оперирует вызовом функций.

Так, если вызвать y=round(func(x)), то после подстановки будет:

Y=((func(x)>=0)?int(funс(x)+0.5):int(funс(x)-0.5))

то есть funс(x) вызовется ДВА раза

В таком случае вызов стоит осуществлять через переменную:

tmp=func(x);

y=round(tmp);

**Задача 36**

Перечисляемые типы. Назначение. Объявление и Использование.

Назначение:

Как ясно из названия тип используются для перечисления значений и символьного обозначения каждого из значений.

Перечисляемый тип — что-то среднее между структурой и макросом.

Описывается как структура, задавая имя типа и перечисляя возможные числовые варианты и их символьные обозначения. И так же, как у макросов, при компиляции символьное описание перечисляемых типов заменяется на числовые значения.

В C++ ключевым обозначением перечисляемого типа является слово enum

enum имя\_типа

{

Символьное значение 1,

…

Символьное значение N

};

По умолчания перечисляемые значения нумеруются с 0 и каждый последующий увеличивается на 1. Но конечно перечисляемым значениям можно присваивать и свои значения.

Пример:

enum ERRORS

{

OK = 0,

NO\_FILE = -1,

NO\_MEMORY = -2,

NO\_RIGHTS = -3

};

Если какому-либо обозначению значение не присвоено принудительно, то оно определяется как предыдущее +1.

Всего есть два основных подхода формирования перечисляемых типов:

1) Перечисление не комбинируемых значений

2) Перечисление комбинируемых значений

Перечисление комбинируемых значений используется в тех случая, когда задание режима и/или получение результата предполагает необходимость комбинации нескольких вариантов параметров.

Комбинируемый тип предполагает побитное обозначение перечислений(через 16-теричный код):

перечисление 0 = 0x00,

перечисление 1 = 0x01,

перечисление 2 = 0x02,

перечисление 3 = 0x04,

перечисление 4 = 0x08,

перечисление 5 = 0x10,

…

Значение комбинаций получается как логическое сложение (ИЛИ) нескольких значений.

Пример:

перечисление 1 | перечисление 4 = 0x09 (00001001)

Для разбора значений используется операция логического умножения (И)

**Задача 37**

Перечисляемые типы. Комбинируемые и не комбинируемые типы. Объявление и использование.

Лекция №8. time code: 0:00-10:30

Перечисляемый тип является чем-то средним между структурой и макросом. Его надо описывать как структуру, задавая имя типа и перечисляя возможные числовые варианты и их символьные обозначения, а также, как у макросов, при компиляции символьное описание перечисляемых типов заменяется на числовые значения.

Комбинируемый тип используется в тех случаях, когда задание режима и/или получение результата предполагает необходимость комбинации нескольких вариантов параметров.

Предполагает побитовое обозначение перечислений. Например:

enum имя типа

{

перечисление0 = 0x00, //0

перечисление1 = 0x01, //1

перечисление2 = 0x02, //2

перечисление3 = 0x04, //4

перечисление4 = 0x08, //8

перечисление5 = 0x10 //16

}

Не комбинируемый тип используется для задания конкретного режима или получения единственного возможного результата из множества. Структура не комбинируемого типа:

enum имя типа

{

перечисление0,

...

перечислениеN

}

**Задача 38**

Преобразования типов. Явные и неявные преобразования. Назначение.

Преобразование - в [информатике](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) преобразование значения одного [типа](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) в значение другого типа.

Выделяют приведения типов:

· явные ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *explicit*);

· неявные ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *implicit*).

Явное приведение задаётся [программистом](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82) в тексте программы с помощью:

· конструкции языка;

· [функции](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), принимающей значение одного типа и возвращающей значение другого типа.

Неявное приведение выполняется [транслятором](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) ([компилятором](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) или [интерпретатором](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80)) по правилам, описанным в стандарте языка.

В [слабо типизированных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8_%D1%81%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [объектно-ориентированных](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) языках, таких как [C++](https://ru.wikipedia.org/wiki/C%2B%2B), механизм [наследования](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) реализуется посредством приведения типа указателя на текущий объект к базовому классу.

Пример неявного преобразования:

float res = 11/3;

//res = 3.0 – так получилось, потому что С++ сначала выполнил операцию деления 11/3 = 3 (int), а только потом перевел его в float и получил 3.0

Пример явного преобразования:

int a1 = 11;

int a2 = 3;

float res = (float)a1/a2;

//res = 3.666666 – так получилось, потому что С++ сначала преобразовал а1 в тип float, что автоматически преобразует а2 в тип float, и выполняется операция деления с плавающей точкой (float)

**Задача 39**

Преобразование типов. Примеры неявного преобразования.

Лекция №8. time code: 10:45

Преобразование типов - это процесс переноса числовых, символьных или иных данных из одного типа в другой.

Примеры неявного преобразования типов:

- арифметическое выражение с операндами разных типов:

все операнды приводятся к наибольшему типу из встретившихся (арифметическое преобразование);

int + double → *double* + double

- присваивание значения выражения одного типа объекту другого типа. В этом случае, результирующим является тип объекта, которому значение присваивается.

int = double → int = *int*

- передача функции аргумента, тип которого отличается от типа соответствующего формального параметра. Тип фактического аргумента приводится к типу параметра:

void f1(double x);

int a;

f1(a); → f1(int) → f1(*double*)

- возврат из функции значения, тип которого не совпадает с типом возвращаемого результата, заданным в объявлении функции. Тип фактически возвращаемого значения приводится к объявленному.

double f2(void)

{

int b;

. . .

return(b); → return(int) → return(*double*)

}

P.S. неявное преобразование типов возможно только тогда, когда где-то указали его правила.

**Задача 40**

Преобразование типов. Правила проведения явного преобразования.

**Правила проведения явного преобразования.**

Явное преобразование типов предполагает, что программист сам указывает какие данные к какому типу надо преобразовать.

Есть два метода устаревший:

int a;

double b;

a=int(b);

или

a=(int)b;

и современный: использование static\_cast и dynamic\_cast

double d=97.0;

char ch=static\_cast<char>(d);

Явное преобразование

-Требует внимательности и осторожности ведь указание типа неверно может вызвать неверный результат

-Использование нового метода упрощает поиск в коде int переменных много, а static/dynamic\_cast меньше

**Задача 41**

Перегрузка функций. Назначение. Правила использования.

Перегрузка функций - способ объявления нескольких одноименных функций с разным набором данных, осуществляющих однотипные преобразования этих данных, при условии, что их неудобно, нежелательно, нелогично и т.п. создавать под разными именами. Компилятор присвоит этим функциям разные идентификаторы и подберет ту функцию, которая подходит под тип данных.

Правила использования:

- преобразование типов осуществляется после выбора наиболее подходящей функции для оставшихся аргументов и параметров

- лучше самостоятельно явно преобразовать (или задать) типы

- если в одну функцию были внесены правки и/или изменения, надо внести изменения (или проверить их необходимость) в другие одноименные функции

- нельзя объявлять функции с одним набором типов данных (определение и распределение происходит по последовательности типов аргументов/параметров)

- при использовании параметров "по умолчанию", необходимо избегать коллизии наборов перегружаемых функций

Перегрузку функций также можно использовать для формирования новых версий функций с измененным набором параметров, постепенно выводя из использования устаревшие варианты и для формирования недостающих данных новой версии функции при вызове старого прототипа (если возможно).

**Задача 42**

Оптимизация программного кода. Критерии оптимизации.

Оптимизация – поиск решения, дающего лучшее значение каких-либо характеристик.

Имея одни и те же исходные данные, можно получать различные результаты в процессе оптимизации в зависимости от выбранного критерия.

**Основные критерии оптимизации программного кода:**

1) Минимизация времени разработки(стоимости разработки)

2) Минимизация времени выполнения(быстрота работы)

3) Минимизация используемой памяти

4) Лёгкая портируемость и переносимость(можно использовать в других системах и задачах)

5) Универсальность применения

При всех подходах предполагается, что программа будет решать поставленную задачу и содержать минимум ошибок.

**Задача 43**

Оптимизация программного кода по времени разработки. Главный критерий и принципы.

Лекция №9. time code: 6:10

1. Главный критерий: чем раньше - тем лучше
2. Основные принципы написания программ под этот критерий оптимизации:
3. Решать задачи известными способами и использовать типовые и имеющиеся решения.
4. Первоначальная разработка и демонстрация заказчику интерфейсной части.
5. Разработка фокусируется на основных функциях. Затем добавляются новые пожелания заказчика в рамках новых, независимых функций.
6. Использование подпрограмм и функций для решения однотипных задач.
7. Одновременно фокусировка на решении конкретных задач в ущерб универсальности.
8. Обучение пользователей (проще обучить пользователя работать с «‎особенностями программы», чем заставить/успеть сделать все нормально в срок). Кроме того, процесс обучения и доработки может быть запараллелен.
9. Возврат и доработка функций в пределах текущей версии не желателен, надо сразу делать законченное решение и не дорабатывать его до новой версии и/или новых положений пользователя. (Если нашлись серьезные ошибки - исправлять надо, но это значит, что функция была разработана плохо).

**Задача 44**

Оптимизация программного кода по времени выполнения. Главный критерий и принципы.

Лекция №9. time code: 9:30

1) Главный критерий: чем быстрее получен результат - тем лучше.

2) Основные принципы написания программ под этот критерий оптимизации:Наиболее часто возникающие ситуации и чаще всего использующиеся элементы программного кода должны:

1.1. Выполняться за минимальное возможное время.

1.2. Поступать на выполнение и проверку условий в первую очередь.

1.3. Не ждать в процессе своей обработки решения более долгих задач.

2. Перестроение логики алгоритма, приводящее уменьшению повторения одинаковых действий, за счёт:

2.1. Увеличения длины программного кода.

2.2. Увеличения объема используемой памяти.

2.3. Введения макросов.

3. Сокращение количества вызовов функций.

4. Сокращение количества операций в циклах.

5. Сокращение количества проверяемых в типовом режиме работы условий (вкупе с 1.2).

6. Упрощение визуального интерфейса (если возможно).

7. Распараллеливание программного кода (если допускает аппаратура).

**Задача 45**

**Оптимизация программного кода по объему используемой памяти. Главный критерий и принципы.**

Исходное требование базируется на ограничениях имеющейся вычислительной базы (доступной/используемой аппаратной базы и/или платформы), а также на требованиях экономии при приобретении аппаратных платформ. Чаще всего возникает при разработке программ для микросхемотехники (например, микроконтроллеры). В зависимости от задач может идти как вместе с требованиями по минимизации времени выполнения, так и без них.

**Главный критерий** : чем дешевле необходимая элементная база – тем лучше.

Проблема применения критерия – необходимо экономить как память для данных, так и память для кода программы.

Поэтому критерии будут противоречивы и могут применяться одновременно в одной разработке для разных участков кода.

Основные принципы написания программ под этот критерий оптимизации:

1. Построение алгоритмов и программ ведется не по их логической или тематической последовательности в решении задачи, а путем сокращения лишних операций: если уж поданы данные в вычислительный модуль, то надо совершить над ним все возможные операции, а потом переходить к другим данным.
2. После использования данных их ячейки памяти тут же передаются другим данным.
3. константы лучше размещать непосредственно в программном коде, а не в памяти данных.
4. Необходимо выяснить закономерности в данных и при их наличии вычислять данные, вместо хранения. Ведет к расходованию памяти для программного кода.
5. Ограничение принимаемой и обрабатываемой информации блоками допустимого объема.
6. Выгрузка редко используемых данных в медленную память (если она есть).
7. Уход от избыточных, типовых и универсальных типов данных и структур к индивидуальным, частным решениям.
8. Использование минимально подходящих по объему типов данных ( вплоть до формирования побитовых комбинаций нескольких значений внутри одного типа данных).
9. Использование битов для логических флагов (никакого bool).
10. Упрощение графического интерфейса или его отсутствие ( передача функций интерфейса другому устройству или программе “большого” компьютера).

**Задача 46**

Оптимизация программного кода по простоте переносимости. Главный критерий и принципы.

Исходное требование базируется на росте интереса ккроссплатформенным решениям, разнообразии программного обеспечения и желания максимально быстро и просто охватить весь спектр пользователей и потенциальных клиентов.

Главный критерий: Вам нужен продукт для Вашей новой

«железяки» и /или операционной системы — сейчас

«скомпилим» («5 минут и готово»).

Основные принципы написания программ под этот

критерий оптимизации:

1. Минимизация затрат на переделку под новое

системное или аппаратное обеспечение.

2. Разработка и поддержка единого решения, а не

нескольких решений для каждого заказчика

3. Использование готовых кроссплатформенных сред

разработки.

4. Использование интерпретируемых языков

программирования.

5. Использование трансляторов или условной

компиляции и/или библиотек для замены системно

привязанных участков кода.

6. Уход от системных вызовов, специфичных для

конкретных операционных систем и платформ там, где это

возможно.

7. Использование эмуляторов и сред виртуализации.

8. Создание и/или использование платформо-зависимых

библиотек и платформо-независимого кода, использующего

эти библиотеки.

9. Деление программы на легко заменяемые блоки.

**Задача 47**Оптимизация программного кода по универсальности применения. Главный критерий и принципы.

Исходное требование формируется на основе заказчика заплатить один раз и получить "все" , и желании программиста "сделать один раз хорошо и "стричь купоны".

Главный критерий: "Лучше день потерять , потом за пять минут долететь"

**Основные принципы написания программ под этот критерий оптимизации:**

1. Для каждой задачи предполагается и реализуется весь спектр её возможных решений и применений. Для любых типов данных их наборов и комбинаций

2. Код программы содержит блоки "самоадаптации" под среду запуска: определение языка , локализации , системных библиотек и каталогов, программ-конкурентов и т.п.

3. Возможно реализация отдельного интерфейса для "тонкой" настройки программы опытным пользователем и/или администратором

4. По каждому желанию заказчика ( на которое не смогли гордо ответить "у нас это все давно есть") реализуются все возможные ( приходящие в голову) варианты дальнейших пожеланий ( "хотелок")

5. Вводятся механизмы поддержки дополнительных программных модулей расширения(плагинов)

6. Публикуется документация для разработки этих модулей. Могут проводиться обучающие курсы.