1. Языки программирования: определение, аспекты, классификации. Парадигмы программирования.

Язык программирования (ЯП) — формальная знаковая система для планирования поведения компьютеров.

Аспекты:

- *синтаксический*
- *семантический*
- *прагматический*

Классификации:

- *машинный код*
- *языки ассемблера*
- *машинно-независимые языки*
- *предметно-ориентированные языки и среды*
- *пятое поколение*

Парадигмы программирования:

- *императивная*
- *структурная*
- *процедурная*
- *объектно-ориентированная*
- *модульная*
- *декларативная*
    - *функциональная*
    - *логическая*
- *мета-программирование*
    - *рефлексивная*
    - *параллельная*

2. Стандарт языка C++. Видимое поведение программы. Виды поведения элементов языка. Виды ошибок. Инструментирование кода.

2.1. Стандарт языка C++

стандарт является основным документом, определяющим отношения между программистами и создателями сред, в которых могут транслироваться и исполнятся программы на языке C++.

2.2. Видимое поведение программы

- записанная в файлы информация
- динамика интерактивных устройств: запросы к вводу данных должны появляться до ожидания этих данных
- обращения к особо помеченным участкам памяти, которые имеют особое, неизвестное транслятору поведение

2.3. Виды поведения элементов языка

- *определенное поведение (defined behavior)*
- *неуточняемое поведение (unspecified behavior)*
- *зависящее от реализации (implementation-defined behavior)*
- *неопределенное поведение (undefined behavior)*

2.4. Виды ошибок

- *синтаксические ошибки (syntax errors)*
- *семантические ошибки (semantic errors)*
- *логические ошибки (logic errors)*
- *ошибки времени выполнения (runtime errors)*

2.5. Инструментирование кода

- *транслятор*
- *компилятор*
- *интерпретатор*
- *JIT-компилятор*
- *компоновщик*
- *система сборки*
- *отладчик*
- *средства инструментирования*
- *виртуальная машина*

3. Процесс трансляции программы на C++ и работа с драйвером компилятора. Устройство объектных файлов. Связанность.

3.1. Процесс трансляции программы на C++ и работа с драйвером компилятора

- программа состоит из одного или нескольких текстов, хранимых в *файлах исходного текста*
- каждый из них проходит *предварительную обработку*, цель которой – представление модифицированного исходного текста в виде последовательности *токенов* (минимальных неделимых единиц языка, имеющий смысл)
- полученная *единица трансляции* транслируется согласно описанию языка
- результат – *объектный файл*
- все файлы, из которых состоит программа и необходимые *библиотеки* (заранее отранслированных файлов) объединяются для получения *образа программы* (полного объема информации, необходимого для выполнения программы в среде выполнения)

3.2. Устройство объектных файлов

Объектный файл – файл, содержащий часть *образа программы*
Секции:

- ***.text*** – текст
- ***.data*** – данные
- ***.bss*** – нулевые данныею
- *.**rodata*** – содержит данные объектов со статистическим временем хранения, которые не подлежат изменению во время программы (чтение)

Символ
Таблица символов

3.3. Связанность

Связанность – свойство, позволяющее нескольким описаниям одного и того же имени (в одной или разных областях видимости) соответствовать одной или той же сущности.

• Когда связанность отсутствует, соответствующее описание говорит о новой сущности первый и последний раз (и является определением).
• Внутренняя связанность позволяет нескольким описаниям в одной единице трансляции говорить об одной сущности.
• Внешняя связанность позволяет то же в пределах всей программы. Она реализуется компоновщиком.
• Если сущность обладает типом, он должен быть совместимым между всеми описаниями (отличаться только по форме/полноте записи).

- ***Внешняя***

    - Пространства имён (включая глобальное).
    - Функции (можно описать первый раз как extern явно).
    - Описания объектов в пространствах имён.

- За исключением неизменяемых объектов в пространствах имён – они имеют внешнюю связанность только со спецификатором времени хранения **extern.**

- ***внутренняя***
    - неизменяемые объекты в пространствах имён по умолчанию.
    - объекты в пространствах имён и функции, описанные первый раз со спецификатором времени хранения static.
    - анонимные пространства имён и всё описанное в них на любую глубину.

- ***слабая (неформально)***
    - inline

4. Системы сборки.

***Сборка* программы C++** — это компиляция исходного кода из одного или нескольких файлов и последующее связывание этих файлов в исполняемый файл (EXE), библиотеку динамической загрузки (DLL) или статическую библиотеку (LIB)

CMake — это кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода. CMake не занимается непосредственно сборкой, а лишь генерирует файлы управления сборкой из файлов CMakeLists.txt:

- Makefile в системах Unix для сборки с помощью make;
- файлы projects/solutions (.vcxproj/.vcproj/.sln) в Windows для сборки с помощью Visual C++;
- проекты XCode в Mac OS X.

CMake может проверять наличие необходимых библиотек и подключать их, собирать проекты под разными компиляторами и операционными системами.
/* Т.е. у вас есть куча кода и файлик, содержащий информацию для cmake, и чтобы скомпилить это дело где-нибудь еще, вам нужно просто запустить там cmake, который сделает всё сам. */

5. Лексический состав C++. Комментарии. Токены. Литералы.

Базовый набор символов исходного текста

- латинские буквы
- арабские цифры
- символы
- пробельные символы, горизонтальная и вертикальная табуляции, новая строка, перевод строки

должен включать управляющие символы звонка (alert), возврата на один символ (backspace), возврата каретки (carriage return) и нулевой символ (null, не путать с символом цифры ноль!).

5.2. Комментарии

Комментарий – включение в текст программы дополнительной информации, не влияющий на ее смысл с точки зрения самого языка программирования, с точки зрения токенов – один пробел.

- ***однострочный*** – «//»
- ***многострочные «***/*» … «*/»

5.3. Токены

Токен – минимальная неделимая единица языка, имеющая смысл. Разбиение происходит таким образом, что в **очередной** токен забирается максимально длинная последовательность символов, которая может являться токеном.

- ***идентификаторы***
- ***ключевые слова***
- ***операции***
- ***пунктуаторы***
- ***литералы***
  - целочисленные
  - символьные
  - с плавающей точкой
  - строковые
  - булевские
  - литерал указателя
  - пользовательские литералы

5.4. Литералы

литералы – фиксированные значения

- целочисленные
- символьные
- с плавающей точкой
- строковые
- булевские
- литерал указателя
- пользовательские литералы

6. Предварительная обработка. Директивы препроцессора.

6.1. Предварительная обработка

Предварительная обработка – обработка последовательности символов из базового набора исходных символов, которую представляет собой единица трансляции после чтения файла, в котором она содержится.

Обязанности препроцессора:

- удалять из текста пару подряд идущих символов «\» и перевод строки
- добавлять символ конца последней строки
- выполнять директивы препроцессора
- все символы заменяются на соответствующие им значениям в наборе символов в среде выполнения
- смежные строковые литералы объединяются в один

6.2. Директива препроцессора

Директива препроцессора – строки программы, первый не пробельный символ которых «#».

Функции:

- включение в единицу трансляции других текстов
- макроподстановки
- условную трансляцию

Включение других текстов выполняется директивой #include.

Директива using

• Может сократить написание сразу множества имён, но возвращается к проблеме одной глобальной области видимости с конфликтами!

Макроподстановки

• #define.

• **Макросы**-объекты (object-like macro) параметров не имеют и соответствуют замене одного идентификатора другой последовательностью токенов:

• Альтернатива: constexpr-объекты.

Условное включение:

• #if выражение

• #else

• #endif

• Транслируется одна из двух ветвей (вторая опционально). Выражение – константное, вычисляется в типе std::intmax_t/std::uintmax_t. Все идентификаторы, оставшиеся после макроподстановок, заменяются на 0.

• #elif выражение

• defined(идентификатор) – проверка, является ли макросом.

• #ifdef идентификатор / #ifndef идентификатор

• __has_include <имя_файла> - проверка наличия файл

Основные директивы препроцессора

#include — вставляет текст из указанного файла

#define — задаёт макроопределение (макрос) или символическую константу

#undef — отменяет предыдущее определение

#if — осуществляет условную компиляцию при истинности константного выражения

#ifdef — осуществляет условную компиляцию при определённости символической константы

#ifndef — осуществляет условную компиляцию при неопределённости символической константы

#else — ветка условной компиляции при ложности выражения

#elif — ветка условной компиляции, образуемая слиянием else и if

#endif — конец ветки условной компиляции

#line — препроцессор изменяет номер текущей строки и имя компилируемого файла

#error — выдача диагностического сообщения

#pragma — действие, зависящее от конкретной реализации компилятора.

7. Выражения. Вычисление выражений. Характеристики выражений. Категории значений.

Выражением (expression) называют последовательность операций и операндов. Отдельно взятые значения, например, в виде литералов, являются простейшей формой выражений.

Характеристики выражений

1) Результат вычисления выражений вновь является выражением определённого типа
2) категорией значения, критерий применимости
3) побочные эффекты

Вычисления выражений

1) лишь план вычислений
2) процесс вычисления знач и инициализации побочных эффектов
3) не вычисленные

Категории значения

Категория значения – характеристика выражения, определяющая трактовку временности его результата

Категория значения:

- *glvalue (обобщённое леводопустимое)* – идентифицирует объект или функцию
  - lvalue - объект (x = 5; x - lvalue)
  - xvalue - специальная категория значений, иден- тифицирующая объекты, которые требуется трактовать как временные

Результат (result) такого выражения – идентифицируемая сущность.

- prvalue = rvalue (чисто праводопустимые)

Результирующий объект (result object) такого выражения – тот, начальное значение которого им задаётся. Если используется как операнд, то его нет.

8. Константные выражения

В некоторых конструкциях языка требуются выражения, удовлетворяющие дополнительным ограничениям, позволяющим вычислить их без выполнения **программы на этапе компиляции** отдельной единицы трансляции — такие выражения называют ***константными (****constant)*.

Основным константным выражением (core constant expression) называют выражение указанного **синтаксиса**, вычисление которого **не** должно включать:

• Преобразование леводопустимого выражения;
• Изменение значений объектов;
• Вызов функций;
• Слишком большое число шагов, превышающих лимиты, устанавливаемые транслятором.

Целочисленное константное выражение — это выражение целочисленного типа, которое вместе с его неявным преобразованием к чисто праводопустимому значению, является основным константным выражением.

Когда в языке требуется константное целочисленное выражение конкретного типа, используют следующую терминологию: **к *преобразованным константным выражениям*** типа T относят константные выражения, неявное преобразование которых к типу T не содержит преобразований, связанных с плавающей точкой.

из лекций:

1. относится к CV-квалификаторам
2. является спецификатором типа, дополняющим и требующим другой спецификатор типа (не самодостаточен)
3. делает соответствующий объект не изменяем
4. не модифицируемые объекты не могут иметь неопределенные начальные значения и требуют явной инициализации
5. для большинства типов языка квалификатор отбрасывается как часть преобразования lvalue, так как описывает по смыслу свойства операции над объектом

constexpr - применим к функциям, до компиляции
consteval - функция выполняется во время компиляции
constinit - для объектов

9. Система типов C++. Фундаментальные и производные типы (4 глава)

Фундаментальные (fundamental)
Неделимые - **скалярными** (scalar).
Система типов

Язык C++ можно классифицировать как имеющий статическую типизацию силы выше средней с элементами механизма вывода типов.

Фундаментальные типы:

Всего 6

Производные типы:(задаются пользователем)

Конкретная реализация языка может содержать и другие целые типы, называемые **расширенными** (extended).

Символьный тип

Тип данных	Диапазон значений	Размер
char	-128...+127	1 байт
unsigned char	0...255	1 байт
signed char	-128...127	1 байт

Диапазоны значений целочисленных типов

Тип данных	Диапазон значений	Размер(байт)
int signed int signed long int	-2147483648 ... 21474836474	
unsigned int unsigned long int	0 ... 4294967295	4
short int signed short int	-32768 ... 32767	2
unsigned short int	0... 65535	2
long long int	$-(2^{63}-1)\ldots(2^{63}-1)$	8
unsigned long long int	$0\ldots(2^{64}-1)$	8

Диапазоны значений вещественных типов

Тип данных	Диапазон значений	Размер (байт)
float	3.4E-38 ... 3.4E+38	4
double	1.7E-308... 1.7E+308	8
long double	3.4E-4932 ... 3.4E+4932	10

Целочисленные литералы (префиксы и суфиксы)

Префиксы: в основном они представлены в четырех типах..

1. **Восьмерично-буквальное (основание 8)** *045*
2. **Шестнадцатеричный литерал (основание 16)** : — 0x или 0X
3. **Двоично-буквенный (основание 2)** : — 0b или 0B

10. Арифметические типы данных и операции. Побитовые операции и дополнительные побитовые функции стандартной библиотеки.

Основная идея: либо в точности, либо ближайшее, либо, если совсем не попал, UB

Операция приведения типов

static_cast<*type-id*>(*expression*)

Арифметические операции:

У арифметических операции нет побочных эффектов

- В общем случае, если результат математически не определён или не представим в типе, в котором совершается операция, - UB! (но см. арифметику по модулю для беззнаковых целых типов).
- Унарные: сохранение знака +, смена знака -
- Бинарные: сложение +, вычитание –, умножение *, деление /, взятие остатка от деления %.

Преобразование к общему типу:

1. Если один из операндов имеет тип с плавающей точкой, побеждает он (для обоих – большей ширины).
2. Иначе целочисленно повысить оба. Если одинаковые – это результат.
3. Если знаковость одинаковая, результат – тот, что шире.
4. Если ранг беззнакового не меньше(>=) ранга знакового, побеждает беззнаковый.
5. Если знаковый может представить все значения беззнакового, побеждает знаковый.
6. Иначе результат – беззнаковый тип того же ранга, что и знаковый тип.

Деление:

- Деление на ноль – UB.
- Деление в целых типах отбрасывает дробную часть (округление в сторону нуля).
- Взятие остатка от деления применимо только к целочисленным операндам.
- Если a/b представимо в типе результата, то гарантируется (a/b)*b+a%b == a

Побитовые операции

- Операции & (побитовые И), | (побитовое ИЛИ) и ^ (побитовое исключающее ИЛИ) – бинарные инфиксные операции
- Операция ~ (побитовое отрицание) – унарная префиксная

- Операции << (побитовый сдвиг влево) и >> (побитовый сдвиг вправо) – бинарные инфиксные операции

Нахуя вообще эти побитовые операции:

- умножение на степени двойки (как положительные, так и отрицательные). Если степень двойки - какая-то константная херня, не трожь эти операции, всё сломаешь
- остаток от деления числа на степень двойки(алгоритм выше)
- вернуть какую-то часть из записи всего числа, по типу маски(хуета)
- чисто ускориться

11. Логические операции. Операции сравнения и равенства. Трёхстороннее сравнение.

Операции с логическими значениями:

- Сравнения == (равно), != (не равно), отношения <, <=, >, >=.
  Прикол: -3<3u - false (тип результата unsigned, -3 unsigned = дохуя)

- Логические операции обозначаются
  отрицание !, И &&, ИЛИ ||.

- Бинарные логические операции вычисляются по короткой схеме.

Вычисления по короткой схеме

Трехстороннее сравнения

• Операция <=> возвращает отношение порядка между двумя значениями как значение одного из типов из <compare>:
• std::strong_ordering::equal/less/greater – порядок с подстановкой
• std::weak_ordering::equivalent/less/greater – порядок без подстановки
• std::partial_ordering:: equivalent/less/greater/unordered – частичный порядок без подстановки

12. Перечисления

Перечисления – это производный тип, множество значений, которого состоит из явно заданных поименованных элементов

enum struct имя-типа-перечисления {перечислители}

Перечислители- список идентификаторов, через запятую, задающих имена значений данного типа.

Представление перечислений-

1. тип представления- целочисленный тип
2. Тип по умолчанию- int. Другой, имя-тип после имени
3. значение на 1 больше предыдущего или 0 для самого первого.
4. имена отражают прагматику, кроме этого перечисления- отдельные типы не могут быть спутаны, собственно с числами

//тип перечисления в памяти представлен как 1 байт
enum struct mode : unsigned char

Дополнительные использования перечислителей

1. Перечисления могут быть явно преобразованы в свой тип с использованием static_cast:
   -Допустимо переводить перечислитель в тип в котором, предыдущее значение не представимо, но скорее всего это смэрть
0. Можно использовать с switch

Перечисления без области видимости

Перечисление без области видимости вводится ключом enum вместо enum struct:

1. Если не указан лежащий тип, то он не обычный int, а некоторый не уточняемый, в котором представлены все перечислители.
2. Имена перечислителей видны не только в области видимости перечисления, но и в окружающей области видимости
3. Преобразование из типа-перечисления в нижележащий тип неявное и входит в состав целочисленное повышения

13. Указатели и леводопустимые ссылки

Указатель – конструкция создания производных типов, значениями которой являются адреса в памяти объектов базового типа (хранит адрес объекта). Относятся к скалярным типам, но арифметическими не считается. Имеет базовый тип – тип объекта, адрес которого содержит.

- **&** - операция взятия адреса – унарная префиксная операция, возвращающая адрес своего операнда (леводопустимы)
- ***** - операция разыменования – унарная префиксная операция, выполняющее действие обратное, обратное адресу

Висячий указатель – указатель на объект, адрес которого не существует. Разыменовывать–UB.
Нулевой указатель – std::nullptr – константа нулевого указателя
Многоуровневые указатели: указатели строятся на базе типа данных или функции и сам является типом данных, так что можно применять его несколько раз.

Указатели и const

Леводопустимые ссылки

Леводопустимая ссылка – конструкция создания производных типов, предназначенная для создания сущностей, идентифицирующих объекты и функции.
Ссылка – отдельный вид сущностей

- временем хранения не обладает
- область видимости определяется как обычно
- связанность, описание/определение – как для объектов.
- позволяет дать дополнительное имя объекту или функции
- инициализация ссылки – привязка, обязательна
- lvalue

Висячие ссылки
Как и указатель **Ссылки и const**

- Если ссылка применяется к немодифицируемому типу данных, через её имя запись в объект недоступна независимо от модифицируемости исходного lvalue.
- Привязать к ссылке на модифицируемый тип немодифицируемое lvalue нельзя.

- В разрешении перегрузок привязка с добавлением const хуже таковой без:

ссылки на функции

Как и имена самих функций, имена ссылок на них или разыменование указателей на них может стоять слева от операции вызова функции для идентификации вызываемой. При привязке ссылки/взятии адреса перегруженной функции контекст использования должен предоставить информацию о требуемом типе/количества аргументов для разрешения перегрузок.

	Указатели	Ссылки
Вид типа	Тип данных	Ссылочный тип
Применение	К объектам и функциям (включая несколько уровней)	К объектам и функциям
Обязательная инициализация	Нет	Да, привязка
Смена идентифицируемой сущности	Если модифицируем	Нет
Идентификация временных объектов	Нет, взятие адреса применимо только к lvalue	Да, возможно с расширением их времени хранения
Специальное состояние «не идентифицирую ничего»	Да, нулевой указатель	Нет, привязка обязательна и неизменна
Переход от идентифицирующего значения к идентифицируемому	Взятие адреса и разыменование	Не требуется, ссылка сама является другим именем идентифицируемой сущности

14. Массивы и арифметика указателей. Классы выделения и идентификации последовательностей объектов в памяти.

Массив (array) — конструкция создания производного типа данных, представлением, которого в памяти является непустая последовательность расположенных подряд объектов базового типа, называемых его элементами (element).

Арифметика указателей
указатель - конструкция создания производных типов, значениями которой являются адреса в памяти объектов базового типа

== и !=. В данном случае значения указателей сравниваются численно.

(<, >, <=, >=) дают результат сравнения индексов.

Сумма двух указателей a[i] заменяет *(a+i)

Разностью двух указателей
std::ptrdiff_t, равное разнице их индексов — возможны отрицательные значения. Этот тип также является псевдонимом, описанным в заголовочном файле cstddef.

std::array:
Главный его плюс: это класс. Не нужно помнить каких-то доп правил при работе с ним, можно спокойно передавать в функцию напрямую и по ссылке. Также можно получить его размер. Можно присваивать один std::array другому.

Связано с темой: есть еще свободные функции std::begin(mas), std::end(mas), std::data(mas), std::empty(mas), std::size(mas). Можно применять к container и array. Выполняют то же самое, что и методы класса array, но могут быть применены и к обычным массивам. Сделаны такие функции для обобщенного программирования (с 20-го страндарта).

15. Классы как агрегаты и их представление в памяти. Члены данных и функции-члены классов

Агрегаты:
-> подобъект - объект, входящий в представление другого объекта

->агрегаты - типы объектов, множество значений которых есть декартово произведение множеств значений всех своих подобъектов

Класс - это производный тип, полностью определяющий свое представление в памяти и допустимые операции. Класс вводится спецификатором типа, состоящим из ключа класса (struct), опциональным именем и содержимым внутри фигурных скобок
-> имя класса становится спецификатором типа, позволяющим в дальнейшем именовать этот же тип - без него повторное идентичное задание такого же спецификатора является другим типом!!
-> описание, содержащее спецификатор класса с имением вводит это имя и, чаще всего, описателей не содержит. оно является определением этого типа, если имеется содержимое в фигурных скобках

Классы создаются с помощью ключевого слова class. Объявление класса определяет новый тип, связывающий код и данные между собой. Таким образом, класс является логической абстракцией, а объект — ее физическим воплощением. Иными словами, объект — это экземпляр класса.

***функциями-членами**.*

Существует несколько ограничений на применение членов класса. Нестатические члены класса не могут иметь инициализатор. Объявление класса не может содержать объявление своего объекта. (Хотя членом класса может быть указатель на объект данного класса.) Члены класса не могут объявляться с ключевыми словами auto, extern и register.

доступ к подобъектам классов:

-> операция . (точка, “прямая выборка”) - бинарная инфиксная операция над gvalue классового типа (слева) и именем подобъекта (справа)

-> ищет имя, заданное правым операндом в типе объекта классового типа слева, для нестатических членов данных возвращает значение той же категории, что и левый операнд, имеющий соответствующий подобъект

нестатические функции - члены классов:

-> описание функций без спецификатора static в определении класса - нестатические функции - члены класса, задающие алгоритмы, оперирующие над конкретным объектом класса

-> если определены внутри определения класса, являются inline по умолчанию. Если только описаны, то определение можно дать как в самом классе, так и в любом окружающем пространстве имен, имя функции при этом должно быть квалифицировано именем самого класса

функции, не являющиеся члеными класса, для контраста называют свободными

->нестатический функции-члены класса имеют дополнительный явно не указанный в их описании параметр - неявный параметр-объект, задающий объект на котором вызывается функция

->его тип по умолчанию леводопустимая ссылка на тип самого класса

->для вызова функции на объекте, используют ту же операцию выборки, указывая справа имя вызываемой функции. Результат такой выборки специальный и соответствует найденному множеству перегрузок плюс объекту, на котором вызывается функция (левый операнд). такое значение может использоваться только как правый операнд операции вызова функции, в котором с точки зрения разрешения перегрузок левый операнд выборки сопоставляется неявному параметру-объекту.

->нестатическую функцию-член класса можно поименовать квалифицированным именем, но без указания аргумента для неявного параметра-объекта выборкой вызвать не удастся

- В описаниях членов класса поиск имён, упомянутых после имени самой описываемой сущности-члена класса просматривает класс как до, так и после точки использования (целиком). Это справедливо даже для определений членов класса вне его области видимости, поиск имён просматривает класс а дальше продолжает поиск имён в окружающих класс, а не определение, областях видимости.
- Если поиск имени из тела нестатической функции-члена класса находит член того же класса (объект или функцию), считается, что имела место выборка из неявного параметра-объекта.

16. Инкапсуляция.Уровни доступа к членам классов. Конструкторы. Акцессоры. Друзья классов.

Инкапсуляция (incapsulation) – скрытие деталей реализации объекта. Запрет прямого доступа к представлению в памяти позволяет предоставить произвольный интерфейс, от него не зависящий и гарантирующий инварианты. /* Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя. */

Инвариант (invariant) – свойство объекта, которое он сохраняет на протяжении всего своего существования. Инварианты могут быть нарушены модификаторами временно в процессе их работы.

Все члены классов имеют один из уровней доступа.

- Открытые (public)
- Закрытые (private)
- protected («защищённый»)

Статистические члены класса:

- Члены данных или функции, описанные в классовой области видимости со спецификатором static – опять в новом смысле, отличном от его применения в блоках и пространствах имён!
- Концептуально статические члены классов относятся не к конкретным экземплярам объектов классового типа, а к самому классу в целом (или ко всем объектам сразу), используя класс как область видимости, аналогично пространству имён.
- Статические члены данных доступны не только через операцию выборки, но и просто по квалифицированным именам, что и следует предпочесть.

Статистические члены данных класса:

- НЕ входят в представление класса, являются просто объектами со статическим временем хранения в классовой области видимости.
- Инициализируются как в пространствах имён. При использовании без инициализатора являются описаниями и требуют определений с квалифицированными именами в окружающей области видимости пространства имён.
- Могут сразу являться определениями, если inline. Для статических членов данных классов constexpr подразумевает inline.

Статистические функции-члены класса:

- НЕ имеют неявного параметра объекта.
- Множество перегрузок функций в классовой области видимости допускает смесь статических/не статических функций, но не допускаются перегрузки, отличающиеся только по наличию/отсутствию static.

- При вызове с объектом для привязки к неявному параметру объекту (выборка, явная или неявная), статические функции его игнорируют. При вызове без, не статические устраняются из множества перегрузок.

Конструкторы (constructor) — особые функции-члены класса, отвечающие за инициализацию объектов его типа. Они вызываются автоматически сразу после выделения под хранение объекта памяти в результате определения объекта или использования операции new. Обычным образом, как функция-член объекта, они вызваны быть не могут.

Виды конструкторов:
1. конструктор по умолчанию (без параметров)

2. параметрический (в нем - параметры)

3. конструктор копированием (нужен для создания копий объектов того же класса)

Деструктор - функция член класса, который вызывается когда объект перестает существовать

Для классов с инкапсуляцией применяют ключ class, члены данных напрямую не доступны. Доступ осуществляется через нестатические функции-члены класса – **акцессоры**.

Дружественная функция — это функция, которая имеет доступ к закрытым членам класса, как если бы она сама была членом этого класса.

Дружественные классы: Один класс может быть дружественным другому классу. Это откроет всем членам первого класса доступ к закрытым членам второго класса.

17. Владение и специальные функции-члены классов. Динамическая память, операции new/delete. Семантика перемещения.

Владение (ownership) – ответственность за освобождение запрошенного ресурса, предоставленного в пользование.

Динамическая память – память выделяемая ядром системы по запросу программы прямо во время исполнения (обращение к ядру ВСЕГДА накладо по реурсам процессора: выделить 1 мб примерно равно выделить 100 мб) ,один из наиболее часто встречаемых видов ресурсов.

Владеть ресурсом (отвечать за освобождение) могут различные элементы программы: алгоритмы и их части (функции), структуры данных (объекты) и др.

В процессе работы программы эта ответственность может переходить от одного элемента программы к другому, что является **передачей владения** (ownership transfer).

Чтобы освободить ресурс, нужно иметь возможность его **идентифицировать**. Объекты, хранящие значение, идентифицирующие ресурсы с целью их последующего освобождения, называют владеющими. После использования значения владеющего объекта для освобождения ресурса, он перестаёт быть владеющим. Невыполнение обязанности по освобождению ресурса (уничтожение владеющего объекта, замена его значения и др.) – логическая ошибка, называемая утечкой (leak)

Владеющие указатели (owning pointer) хранят адреса, полученные в результате вычисления new. Остальные являются не владеющими (non-owning).

Специальные функции члены классов:

Конструкторы по умолчанию,

конструкторы перемещения,

конструкторы копирования,

операции присваивания перемещением,

операции присваивания копированием и

(потенциальные) деструкторы –

специальные функции-члены класса (special member functions)

18 Наследование классов. Виртуальные функции. Чисто виртуальные функции.

Наследование - отношение объектов классовых типов, в котором один из них (производный) включает другой в качестве неименованного подобъекта, называемого базовым подобъектом.
Наследование – механизм повторного использования кода, объединяющий классы в иерархию, устанавливаемое этим механизмом соотношение называют is-a. Частный класс, включающий в себя элементы общего называют унаследованным от исходного или производным от него, либо **дочерним**, а исходный класс – **базовым или родительским**

Применение:

В C++ наследование в первую очередь – способ повторного использования кода.

Наследование в C++ не соответствует принципу подстановки (substitution principle): вместо любой сущности можно подставить производную с сохранением семантики – при скрытии имён в производных классах нет никаких требований соответствовать базовой семантике в производных классах.

Виртуальные функции — специальный вид функций-членов класса. Виртуальная функция отличается об обычной функции тем, что для обычной функции связывание вызова функции с ее определением осуществляется на этапе компиляции. Для виртуальных функций это происходит во время выполнения программы.

Для объявления виртуальной функции используется ключевое слово virtual. Функция-член класса может быть объявлена как виртуальная, если

● класс, содержащий виртуальную функцию, базовый в иерархии порождения;

● реализация функции зависит от класса и будет различной в каждом порожденном классе.

Чистая виртуальная функция

Базовый класс иерархии типа обычно содержит ряд виртуальных функций, которые обеспечивают динамическую типизацию. Часто в самом базовом классе сами виртуальные функции фиктивны и имеют пустое тело. Определенное значение им придается лишь в порожденных классах. Такие функции называются чистыми виртуальными функциями.

Чистая виртуальная функция — это метод класса, тело которого не определено.

20. Операторы. Ветвления и циклы в машинном коде.

Операторы (statements) - фрагменты программы C++, выполняемые последовательно. По факту они - вообще могут быть всем чем угодно:

1) expression statements -их большинство в программах. Это - всё, что кончается на “ ; ” (вызовы функций, присваивания)

2) compound statements; Это группа стейтметов в фигурный скобках

```
for (int i = 0; i<=6; ++i)

{

std::cout<<”I love Palace”<<std::endl;

int x = 0;

}
```

То, что в фигурных скобках(вместе с фигурным скобками) - compound statement.

{ } - compound statement без statements внутри - тоже compound statement.

3) selection statements; if\switch

4) iteration statements; Циклы

5) jump statements; 1) break statement; 2) continue statement; 3) return statement with an optional expression; 4) return statement using list initialization; 5) goto statement.

6) declaration statements; - Сюда входят все описания типо int x;

7) try blocks; - try конструкции.

21. Описания и определения. ODR. Простое описание. Инициализация. Время хранения объектов.

Описание - конструкция или элемент языка, которые вводят в программу имена объектов и их сущности.

Определения - частный случай описания, который помимо перечисления свойств имени задает содержание сущности, ему соответствующей в частности, определение объектов отвечает за выделение памяти в среде выполнения, в определение функций содержат запись последовательных действий.

ODR

C++ требует, чтобы любая функция была определена не более одного раза – One Definition Rule. Как только вы определяете функцию с одним и тем же именем и сигнатурой в разных единицах трансляции (файлах .cpp), вы получаете индикацию ошибки на этапе линковки.

Простое описание это совокупность:

- Последовательности спецификаторов описания (declaration specifier) в любом порядке, включая спецификатор типа (type specifier) – базовые характеристики описываемых сущностей;
- Последовательности описателей (declarator), разделённых запятыми, – имена сущностей и конструкции создания производных типов, применяемые к ним относительно типа, заданного спецификатором типа описания;
- Точка с запятой

Инициализация

- По умолчанию – минимум действий.
    - Если нет других видов.
    - Для известных нам типов ничего не делает.
- Для объектов со статическим временем хранения:
    - Статическая инициализация – на этапе компиляции.
        - Константная инициализация – если инициализатор константен.
        - Иначе нулём – запись нулевых байтов в представление.
    - Динамическая инициализация – на этапе выполнения программы, если ещё требуется.
- Копированием
    - В инициализаторе в форме с =.
    - Вызов функции/возврат значения.
    - При инициализации с несовпадением типа рассматривает только неявные преобразования

Время хранения - характеристика объекта, которая определяет в какой момент для него выделяется и освобождается память

- Автоматическое – от выполнения определения до выхода из блока
    - Видимость – блок и нет static/extern или параметр функции
- Статическое – всё время выполнения программы
    - Область видимости – пространство имён или в блоке с static/extern.

22. Области видимости и пространства имён. Поиск имён Квалифицированные и не квалифицированные имена. Описания и директивы using

Область видимости - та часть текста программы, где данное имя может использоваться для идентификации этой сущности

Область программы, в которой может использоваться имя – **область видимости**
Пространства имен – именованная область описания, они вводятся в программу конструкцией:
namespace name{declaration}
Пространства имен с кастомными именами, имеют с глобальным внешнюю связанность, то есть к ним можно обратиться из любой единицы трансляции. К сущности, имеющей внутреннее связывания можно обратиться только из единицы трансляции, в которой она определена.
Квадифицированные – прописан путь через ::

Описания (declaration) вводят в программу имена сущностей и определяют их свойства. Тем самым описания задают смысл используемых в программе имён.

Определение (defininition) — частный случай описания, который помимо перечисления минимально необходимых свойств имени задаёт содержимое сущности, ему соответствующей.

директива using

Делает описание из указанного пространства имён видимыми при неквалифицированном поиске имён в ближайшем пространстве имён, окружающем текущую область видимости и номинируемое пространство имён. Транзитивна, если в номинируемом пространстве имён тоже есть директивы using.

Для квалифицированного поиска объединение номинируемых пространств имён просматривается, если в указанной компоненте следующее имя не найдено.

Директива using только влияет на поиск имён, а сама ничего не описывает.

23. Функции и их вызов. Соглашения о вызовах. Встраивание функций.

Функция (function) — конструкция создания производного типа, соответствующая частям алгоритма программы.

Как и большинство имён, функция может быть вызвана только после своего описания.

Операция вызова функции – операция, синтаксис которой состоит из значения, идентифицирующего вызываемую функцию (имя короче), за которым в круглых скобках следует список аргументов, разделенных запятыми.

Формально операция вызова функции имеет арность, равную числу передаваемых вызываемой функции аргументов (argument) — значений параметров — плюс один, необходимый для идентификации вызываемой функции.

Операция вызова функции осуществляет следующие действия:

1. Происходит вычисление значений всех аргументов в неуточняемом порядке.

2. Происходит приостановка выполнения текущей функции, и управление передаётся функции, идентифицируемой первым операндом (перед скобками) операции вызова функции.

3. Перед началом выполнения вызываемой функции объекты, соответствующие её параметрам, получают начальные значения, вычисленные на первом шаге.

4. Происходит выполнение вызванной функции до окончания её тела или выполнения ей оператора return.

5. Значение выражения в операторе return в вызываемой функции после приведения к типу возвращаемого функцией значения становится результатом операции вызова функции, и выполнение функции, содержащей эту операцию, продолжается.

Положения соглашений о вызовах функций

1. Параметры передаются через стек, куда помещаются по порядку, начиная с последнего.

2. Возвращаемое функцией значение передаётся вызывающей через регистр EAX.

3. Содержимое всех остальных регистров, кроме ECX и EDX должно сохраняться вызываемой функцией, если ей необходимо их использовать, она отвечает за сохранение и восстановление их значений.

4. Освобождение места на стеке, выделенного для хранения параметров функции, осуществляет вызывающая функция.

Это не полные соглашения о вызовах, которые даже в рамках языка C++ на платформе

x86 различаются между операционными системами и трансляторами, но они дают первое представление о том, какие варианты в организации взаимодействия функций возможны

Встраиваемая функция – функция, в описании которой присутствует спецификатор inline. Формально, стандарт языка трактует это как просьбу транслятору максимально быстро ускорить вызов этой функции.

Как работает: тело функции копируется непосредственно в точку вызова функции.

Ограничения, накладываемые на встраиваемую функцию:

1. Определение встраиваемой функции должно иметься во всех единицах трансляции, где она используется, чтобы у транслятора был доступ к её коду для встраивания. Встраиваемые функции нередки в интерфейсах единиц трансляции, в таком случае в заголовочный файл помещаются именно их определения.

   Первое описание функции со спецификатором inline должно быть перед определением функции в единице трансляции или быть этой единицей.

24. Перегрузка функций и операций

Перегрузка функций (function overloading) – описания одноимённых функций, различающихся по набору входных параметров.

Выбор из этого множества перегрузок (overload set) – дополнительный шаг алгоритма поиска имён.

Выбор осуществляется исходя из типов аргументов, выясняемых в контексте использования имени функции, в нашем случае – в операции вызова функции.

Разрешение перегрузок:

1. Оставить в множестве перегрузок только годные (viable) функции: совпадающие по числу параметров с числом аргументов так, что из каждого типа аргумента есть неявное преобразование в тип соответствующего параметра.

2. Если годных не осталось – семантическая ошибка. Если годная одна – это результат.

3. Иначе годных больше одной и требуется выяснение наилучшей годной (best viable). Наилучшая годная та, которая лучше всех остальных (если такая есть, то она только одна по построению). Чтобы быть лучше другой функции, данная должна быть лучше по хотя бы одному параметру, и не хуже по всем остальным. Для конкретного типа аргумента и двух типов соответствующих параметров функций требуемые преобразования классифицируются по рангам в порядке ухудшения:

● Точное совпадение (exact match) – преобразований типа не требуется.

● Повышения (promotions) – integer/floating promotion

● Преобразования (conversions) – всё остальное из рассмотренного нами. (int --> short и т.д.)

Перегрузка операций:

● Классы позволяют задать семантику операций над ними (кроме ., :: и ?:). Некоторые перегружать не следует (,,&&,||). Следует действовать по принципу «наименьшего сюрприза»

● Перегрузки операций – функции с именами вида operator@. Постфиксные инкремент/декремент для отличия от префиксных – бинарные, со вторым фиктивным операндом 0.

● Могут быть свободными функциями (кроме =, [] и ()) или нестатическими членами класса (первый операнд – неявный параметр-объект, непригодно, если требуется определить не для вашего класса). Нельзя ввести новые операции, нельзя поменять приоритет и ассоциативность (бинарные логические теряют вычисление по короткой схеме.

● Одна перегрузка <=> перегрузит операции:(>,>=, <, <=, ==, !=)

25. Концепции (до формальных в C++20). Шаблоны. Виды шаблонных сущностей и определение значений параметров шаблона. Виды инстанции и специализации шаблонов.

Концепция-набор требований к типу в виде:
-> синтаксиса допустимых конструкций языка с использованием этого типа (обычно выражений, включающих значения этого типа)

-> семантики этих конструкций

->ограничений по ресурсоемкости этих конструкций

- при их соблюдении говорят, что типа “удовлетворяет концепции”
- концепция может ссылаться и тем самым включать требования других концепций без их повторения - в таком случае она их “уточняет”
- до c++20 концепции не имеют синтаксических проявлений в самом языке, а используются только в документации

если мы не знаем требований к каким-то объектам, то не можем выдвинуть и требования к целому. некоторые требования до сих пор нельзя выразить на уровне языка.

Шаблоны (template) - форма описания, задающая общий вид семейства описаний функций, классов, псевдонимов типов или объектов.

- Допустимы только в областях видимости пространства имен и классов. //пр: локальные классы внутри блоков не допускают
- являются описаниями или определениями по правилам сущностей, семейства которых представляют. //шаблон сущности не является самой сущностью
- синтаксически является описанием одной из вышеупомянутых сущностей, перед которой в форме template<> перечислены параметры шаблона (в угловых скобках через запятую форма аналогична параметрам функции, включая возможное отсутствие имени и аргумента по умолчанию)

виды параметров шаблонов:

- **типовые** “typename”(“class”) вместо имени типа, соответствуют основной идее обобщенного программирования - параметризации по типам
- **нетиповые** - обычное описание с типом (целочисленным, перечислением или ссылкой)
- **шаблонные** “template ... “ - это когда шаблонами оперируют еще одни шаблоны

значения параметров шаблона по умолчанию:

- аналогичны аргументам параметров функций, исполняются, только если предыдущие 2 способа выяснения параметров шаблона не дали результата
- в отличие от аргументов по умолчанию, могут ссылаться на предыдущие:
  template <typename T, typename U=T>

  void f();

- значения параметров шаблонов по умолчанию не являются входными сведениями о типе аргументов для дедукции (алгоритм определения значения

параметров шаблона путем сопоставления типа аргумента из контекста использования шаблона функции форме типа параметра из описания шаблона)

26. Заполнители. Прозрачная передача и возврат значений. Вариадические шаблоны, пачки параметров, распаковка и свёртка пачек.

Прозрачная передача – передача по универсальной ссылке плюс коррекция категорий для дальнейшей передачи, которая сохраняет все характеристики первоначального аргумента.

Вариадические шаблоны – шаблоны, имеющие пачки параметров.
Пачка параметров шаблона – вид параметра шаблона, когда один параметр шаблона соответствует произвольной упорядоченной последовательности аргументов шаблона того же вида (включая пустой)
Единственная операция – sizeof… - возвращает константное значение, соответствующее числу аргументов в пачке

Распаковка пачки – единственный способ доступа к отдельным элементам пачки - … (троеточие)

- аргумент – образ (некоторая конструкция языка, содержащая ещё не распакованные пачки с одинаковым количеством аргументов в каждой)
- результат эквивалентен записи стольких копий образца, сколько аргументов в каждой пачке, разделённых запятыми

Может применяться:

- в списке аргументов операции вызова функции пачка распаковывается в последовательность аргументов функции.
- в списке аргументов шаблона пачка распаковывается в последовательность аргументов шаблона.
- в списке инициализации в фигурных скобках пачка распаковывается в последовательность инициализаторов.
- в качестве типа параметра функции

Пачка параметров функции – аргумент, который соответствует последовательности аргументов функции, типы которых определяются входящими в образец ее типа пачками параметров шаблонов данной функции.

- так же операция sizeof…
- если пачка *последний параметр функции*, то она может принять произвольное количество аргументов и дедуцировать пачки параметров шаблонов, входящих в описание ее типа по типу этих параметров
- если пачка *не последний параметр функции*, она не дедуцируем и считаются пустыми последовательностями
- можно указать значение соответствующих пачек параметров шаблона
- при явном указании аргументов для пачек параметров шаблонов по достижении первой такой пачки все последующие аргументы в списке аргументов шаблона захватываются ей, так что все остальные должны иметь значения по умолчанию или быть дедуцируемы

- шаблон класса может иметь только одну пачку параметров, которая должна быть последним параметром шаблона

27. Контракты и их ширина. Предусловия и постусловия. Исключения и механизм их обработки. Гарантии при наличии исключений.

Контракты
Основная идея контрактов в том, что по аналогии с контрактами в бизнесе, для каждой функции или метода описывают договорённости. Эти договорённости должны соблюдать как вызывающая сторона, так и вызываемая.
Неотъемлемой частью контрактов является как минимум два режима сборки- отладочный и продуктовый. В зависимости от режима сборки контракты должны вести себя по разному. Наиболее распространённой практикой является проверка контрактов в отладочный сборке и их игнорирование в продуктовой.
Ширина контрактов //широкие контракты - когда определены границы для параметра (их прописывают. по сути сейчас все функции с узкими контрактами)

Проверка предусловий.
Стандартный подход к обработке предусловий следует философии языка, направленный на максимальную производительность: функция пишется в предположении, что значения её аргументов удовлетворяют предусловиям, а их нарушение ведёт к неопределенному поведению. Вызывающий функцию код, который алгоритмически построен так, что вызывает её только для удовлетворяющих предусловиям аргументам, не тратит лишнее время на дополнительную проверку. Если входные аргументы функции зависят от внешних источников (файлы, пользовательский ввод, и др.), обязанность проверки предусловий лежит на вызывающей функции. Если проверить аргументы функции на корректность ресурсоёмко и/или такая проверка фактически дублирует часть алгоритма самой функции, такие требования корректности в предусловия не вносят, вместо этого, если в процессе работы функции выясняется некорректность аргументов, об этом сообщается вызывающей стороне (через выходные параметры или иным способом).

Исключение- специальным образом созданный объект, содержащий информацию о ситуации в программе, требующей прерывания её последовательности выполнения по обычным правилам.
Механизм обработки исключений- средства языка по созданию исключений и поиску их обработчиков в последовательности окружающих точку бросания блоков

Гарантия безотказности- свойство функции не бросать исключения никогда
- Сильная гарантия при исключениях- свойство функции выполнять свою работу полностью или при выбрасывании из себя исключения оставлять программу в исходном состоянии
- Базовая гарантия при исключениях- свойство функции при выбрасывании исключения оставлять программу в корректном, но не уточненном состоянии с сохранением инвариантов
- Для корректности работы программы при наличии возможности бросания исключения требуется предоставлять как минимум базовую гарантию, сильную гарантию делать довольно трудоемко

28. Сложность вычислений. Асимптотическая нотация. Структуры данных на базе последовательности объектов в памяти.

Сложность вычислений

Два основных ресурса, которые используют все программы — процессорное время и память. Хотя на практике наиболее интересным является фактическое время работы, например, в секундах, при рассмотрении сложности вычислений рассматривают абстрактные шаги алгоритма, а при оптимизации реализации — число тактов процессора. При оценке объёма памяти, требуемого для работы алгоритма, в качестве единицы измерения обычно используют элемент входных данных, при этом сама память, занимаемая этими данными, не учитывается — говорят только о дополнительных расходах.

В теории сложности вычислений конкретная единица измерений не имеет значения, поскольку рассматривается асимптотическое поведение алгоритма при усложнении задачи — обычно увеличении объёма входных данных. Это позволяет сравнивать поведение алгоритмов относительно друг друга даже тогда, когда точно вычислить их ресурсоёмкость не удаётся.

Структуры данных на базе последовательности объектов в памяти

1. (Динамические) массивы – доступ по индексу за O (1), поиск за O(n) или за O(log n) для отсортированных последовательностей, вставка и удаление с конца O(1), вставка и удаление в другие места за O(n)

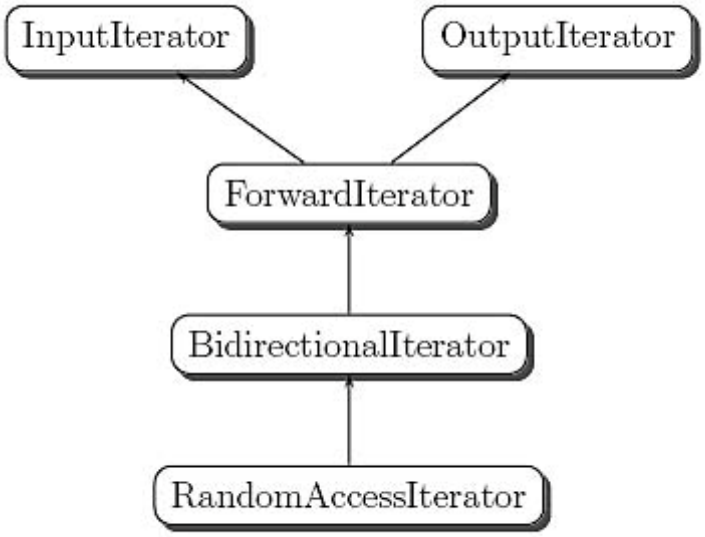
2. Стек – массив, в котором используется только некоторое количество элементов в начале, позволяя добавлять и удалять элементы с конца этой последовательности. Push и Pop за O(1), peek и доступ к элементу относительно вершины за O(1)

3. Буферное окно - В отличие от обычного массива, неиспользуемое место располагается не всегда в конце массива, а сдвигается при каждой операции вставки или удаления на место этой операции. В общем случае элементы массива содержат последовательно: первую часть данных, неиспользуемое пространство и вторую часть в конце, где первая и последняя части могут отсутствовать, когда окно расположено строго в начале или конце массива. Такие последовательности операций характерны, например, при работе с буфером символов текстового редактора: между перемещениями курсора обычно происходит множество нажатий клавиш с символами или Backspace и Delete. За ускорение этих операций приходится платить усложнением алгоритмов работы с такой структурой данных: помимо числа используемых элементов и общего их количества необходимо хранить и учитывать положение «окна» из неиспользуемых элементов.

29. Концепции итераторов и контейнеров. Std::vector.

Итератор — это объект, который способен перебирать элементы контейнерного класса без необходимости пользователю знать реализацию определенного контейнерного класса

Концепция итераторов – концепция, более широкая, чем концепция указателей. Её смысл — понятие объекта, «указывающего» на последовательность других объектов одного типа, и позволяющего по ней перемещаться.



30. Алгоритмы стандартной библиотеки. Лямбда-выражения.

Стандартная библиотека – набор средств, самих являющихся конструкциями языка C++, обеспечивающих базовую функциональность программ.

Алгоритм – последовательность явных инструкций, ведущих к цели за конечное число шагов.

Алгоритмы стандартной библиотеки – алгоритмы, расположенные в библиотеке алгоритмов (заголовочный файл algorithm).

Каждый алгоритм – функция, оперирующая над диапазоном элементов, определяющимся как [first, last).

Алгоритмы стандартной библиотеки обычно относятся к одной из 3х категорий:

1. Инспекторы – используются для просмотра данных в контейнере без изменения

2. Мутаторы - используются для изменения данных в контейнере

3. Фасилитаторы – используются для генерации результата на основе значений элементов данных

Лямбда-выражения

Функтор – объект, к которому применима операция вызова функции.

Если необходим сложный функтор, используются **лямба-выражения**. Эти выражения – краткий способ записи значения некоторого классового типа, являющегося функциональным объектом. Этот тип называется замыканием.

31. Основы системы ввода/вывода – потоки языка Си++.

Поток (stream) – это абстракция, отражающая перемещение данных от источника к приемнику. Реализация потоков в языке C++ выполнена таким образом, что потоки для различных источников и приемников выглядят единообразно.

● Ввод интерпретируется как чтение данных в поток (с клавиатуры).

● Эти потоки определяет класс istream (input stream). Вывод интерпретируется как запись данных из потока (на дисплей).

Эти потоки определяет класс ostream (output stream).

Наследником классов istream и ostream является класс iostream, следовательно им наследуются и объекты чтения и объекты записи. Для потоков ввода/вывода в библиотеке IOStream определены глобальные объекты чтения и записи:

● объект cin читает данные со стандартного устройства ввода (клавиатура);

● объект cout записывает данные на стандартное устройство вывода (монитор).

32. Форматированный ввод/вывод. Часто используемые манипуляторы std::endl, std::setw, std::setfill, std::dec, std::hex

Для ввода:

Аналогично выводу, для std::cin, операции >> и леводопустимого правого операнда.

1. Проверить флаг ошибки. Если стоит – ничего не делать.
2. Пропустить все пробельные символы.
3. Вводить символы, пока накопленные есть корректное представление требуемого типа, аналогичное литералам, кроме:
   - Нет префиксов систем счисления (всегда основание 10) и суффиксов ширин/знаковости.
   - Целочисленные по форме литералы подходят и плавающей точке.
   - Допускается знак величины перед ней.
   - Булевские значения ожидают 0/1.
   - Узкие символьные типы вводят один символ как его код.

   Первый неподходящий символ остаётся не считанным.

0. Производится попытка преобразовать последовательность считанных символов в значение требуемого типа.
   - Удалось (в рамках значений типа) – побочный эффект: запись в правый операнд этого значения.
   - Не удалось: выставление флага ошибки на потоке. Побочный эффект – запись в объект минимального/максимального представимого значения, если неудача из-за непредставимости, или 0, если преобразуемая последовательность была пустой/некорректной.

- std::cin контекстно преобразуем (is contextually convertible) к bool со значением отсутствия флага ошибки.

Для вывода:

- Включить заголовок <iostream> для получения описания std::cout.
- Побочный эффект от применения бинарной инфиксной операции << к std::cout и арифметическим типам/строковым литералам – вывод их представления в стандартный файл (поток/stream) вывода.
  Результат – левый операнд (ассоциативность слева направо).
- То же для std::cerr для стандартного файла вывода ошибок.
- bool выводится как 0/1, узкие символьные типы как символы с соответствующими кодами.
- При работе с терминалом буферизация построчно.

33. Неформатированный ввод/вывод. Методы put, get, read, write, seek/tell.

Каждый **класс файлового ввода/вывода** содержит **файловый указатель**, который используется для отслеживания текущей позиции чтения/записи данных в файле. Любая запись в файл или чтение содержимого файла происходит в текущем местоположении файлового указателя. По умолчанию, при открытии файла для чтения или записи, файловый указатель находится в самом начале этого файла. Однако, если файл открывается в режиме добавления, то файловый указатель перемещается в конец файла, чтобы пользователь имел возможность добавить данные в файл, а не перезаписать его.