**Программирование** -это процесс создания программ (программного обеспечения).

**Пользовательское программирование** - набор методик и инструментов, которые позволяют программировать конечным пользователям — людям, не являющимся профессиональными разработчиками ПО. Люди, не являющиеся профессиональными разработчиками, могут использовать инструменты EUD для создания или модификации программных артефактов (описаний автоматических действий) и сложных объектов данных без знания языков программирования.

**Системное программирование**- это программирование, направленное на взаимодействие с аппаратными ресурсами вычислительной системы. Выполняется преимущественно на низкоуровневых языках программирования

**Технология программирования**- Технология программирования (ТП) - технология разработки программного средства (ПС), включающая все процессы, начиная с момента зарождения идеи этого средства.

**Теоретическое программирование**- то наука о математических моделях понятия «программа»; в частности, программа — это формула в некоторой разновидности языка первого порядка.

**Внешние критерии качества ПО-**

**Внутренние критерии качества ПО-**

**Парадигма программирования-** это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию).

**Процедурная декомпозиция-** представление разрабатываемой программы в виде совокупности вызывающих друг друга подпрограмм. Каждая подпрограмма в этом случае выполняет некоторую операцию, а вся совокупность подпрограмм решает поставленную задачу.

**Объектно-ориентированная декомпозиция -**  разделение системы, выбрав в качестве критерия декомпозиции принадлежность ее элементов к различным абстракциям данной проблемной области. Абстракции описываются в виде объектов. Тогда каждый объект обладает своим собственным поведением, и каждый из них моделирует некоторый объект реального мира.

**Понятие алгоритма-**  это всякая система вычислений, выполняемых по строго определѐнным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи.

**Общее представление работы программы-**

**Соотношение компонентов программ и компьютерной архитектуры-**

**Что такое поток и процесс-**

**С точки зрения пользователя:**  
**Процесс** — экземпляр программы во время выполнения.  
**Потоки** — ветви кода, выполняющиеся «параллельно», то есть без предписанного порядка во времени.

**С точки зрения операционной системы:**  
**Процесс** — это абстракция, реализованная на уровне операционной системы. Процесс был придуман для организации всех данных, необходимых для работы программы.  
**Процесс** — это просто контейнер, в котором находятся ресурсы программы:

* адресное пространство
* потоки
* открытые файлы
* дочерние процессы
* и т.д.

**Поток** — это абстракция, реализованная на уровне операционной системы. Поток был придуман для контроля выполнения кода программы.

**Этапы разработки программы-**

Первый этап – постановка задачи

Второй этап - выбор метода решения

Третий этап - разработка алгоритма решения задачи

Четвертый этап – кодирование алгоритма

Пятый этап – трансляция и компиляция программы

Шестой этап – тестирование программы

Седьмой этап – создание документации

Восьмой этап - сопровождение и эксплуатация

**Алгоритм-**

**Структура программы -** искусственно выделенные программистом взаимодействующие части программы. Использование рациональной структуры устраняет проблему сложности разработки; делает программу понятной людям; повышает надежность работы программы при сокращении срока ее тестирования и сроков разработки вообще.

**Комментарии -** это текст, который предназначен для программистов и не обрабатывается компилятором. Обычно комментарии используются для создания заметок к коду для дальнейшего использования. Компилятор обрабатывает их как пробел.

**Состав языка:**

**Лексемы**

**Символы**

**Выражение**

**Оператор**

**Ключевые слова**

**Константы**

**Препроцессинг -** это *макро процессор*, который преобразовывает вашу программу для дальнейшего компилирования. На данной стадии происходит происходит работа с препроцессорными директивами. Например, препроцессор добавляет хэдеры в код (**#include**), убирает комментирования, заменяет макросы (**#define**) их значениями, выбирает нужные куски кода в соответствии с условиями **#if**, **#ifdef** и **#ifndef**.

Хэдеры, включенные в программу с помощью директивы **#include**, рекурсивно проходят стадию препроцессинга и включаются в выпускаемый файл. Однако, каждый хэдер может быть открыт во время препроцессинга несколько раз, поэтому, обычно, используются специальные препроцессорные директивы, предохраняющие от циклической зависимости.

**Компиляция -** На данном шаге g++ выполняет свою главную задачу — компилирует, то есть преобразует полученный на прошлом шаге код без директив в ассемблерный код. Это промежуточный шаг между высокоуровневым языком и машинным (бинарным) кодом.

**Ассемблерный код -** это доступное для понимания человеком представление машинного кода.

**Ассемблирование -** Так как x86 процессоры исполняют команды на бинарном коде, необходимо перевести ассемблерный код в машинный с помощью **ассемблера**.

Ассемблер преобразовывает ассемблерный код в машинный код, сохраняя его в *объектном файле*.

**Объектный файл -** это созданный ассемблером промежуточный файл, хранящий кусок машинного кода. Этот кусок машинного кода, который еще не был связан вместе с другими кусками машинного кода в конечную выполняемую программу, называется объектным кодом.

**Линковка -** связывает все объектные файлы и статические библиотеки в единый исполняемый файл, который мы и сможем запустить в дальнейшем. Для того, чтобы понять как происходит связка, следует рассказать о таблице символов.

**Таблица символов -** это структура данных, создаваемая самим компилятором и хранящаяся в самих объектных файлах. Таблица символов хранит имена переменных, функций, классов, объектов и т.д., где каждому идентификатору (символу) соотносится его тип, область видимости. Также таблица символов хранит адреса ссылок на данные и процедуры в других объектных файлах.  
Именно с помощью таблицы символов и хранящихся в них ссылок линкер будет способен в дальнейшем построить связи между данными среди множества других объектных файлов и создать единый исполняемый файл из них.

**Динамическая типизация** — приём, используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной.

**Стати́ческая типиза́ция** — приём, широко используемый в языках программирования, при котором переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменён позже (переменная или параметр будут принимать, а функция — возвращать значения только этого типа)

**Интерпретатор** — программа (разновидность транслятора), выполняющая интерпретацию. Интерпретация — построчный анализ, обработка и выполнение исходного кода программы или запроса (в отличие от компиляции, где весь текст программы, перед запуском, анализируется и транслируется в машинный или байт-код, без её выполнения).

**В вычислительной технике компилятор**-это компьютерная программа, которая переводит компьютерный код, написанный на одном языке программирования (исходный язык), на другой язык (целевой язык).

**Сема́нтика** **в** **программировании** **—** **дисциплина, изучающая** **формализации** **значений** **конструкций** **языков** **программирования** **посредством** **построения** **их** **формальных** **математических** моделей.

**Рефакторинг** — это процесс изменения кода, призванный упростить его обслуживание, понимание и расширение, при этом не изменяя его поведение.

**Debug** — это процесс **отладки** (проверки) кода, когда в процессе его выполнения можно остановиться в обозначенном месте и посмотреть за ходом выполнения.

**Псевдокод** — компактный (зачастую неформальный) язык описания алгоритмов, использующий ключевые слова императивных языков программирования, но опускающий несущественные подробности и специфический синтаксис.

**Транслятор -** Программа перевода записи алгоритма с одного алгоритмического языка на другой (в частности, на язык вычислительной машины)

**Машинный код** — система команд (набор кодов операций) конкретной вычислительной машины, которая интерпретируется непосредственно процессором или микропрограммами этой вычислительной машины.

**Сопроцессор** - это компьютерный процессор, используемый для дополнения функций основного процессора (центрального процессора)

**Обратная разработка** — исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы; например, чтобы обнаружить недокументированные возможности (в том числе программные закладки), сделать изменение или воспроизвести устройство, программу или иной объект с аналогичными функциями, но без прямого копирования.

**Дизассемблирование** – **это** операция, обратная ассемблированию, т.е. восстановление текста программы на ассемблере из исполняемой программы в машинных кодах.

**Ассемблерной вставкой** называют возможность компилятора встраивать низкоуровневый код, написанный на ассемблере, в программу, написанную на языке высокого уровня, например, Си или Ada.

**Регистры** – **это** специальные ячейки памяти, расположенные непосредственно в процессоре. Работа с **регистрами** выполняется намного быстрее, чем с ячейками оперативной памяти, поэтому **регистры** активно используются как в программах на языке **ассемблера**, так и компиляторами языков высокого уровня.

**Эзотерический язык программирования** — язык программирования, разработанный для исследования границ возможностей разработки языков программирования, для доказательства потенциально возможной реализации некой идеи (так называемое «*доказательство концепции*), в качестве произведения программного искусства или в качестве шутки (компьютерного юмора).

**Функциональные языки**

**Логические языки**

**Идентификатор** — это последовательность символов, используемая для обозначения одного из следующих элементов:

* Имени объекта или переменной
* Имени класса, структуры или объединения
* Имени перечисленного типа
* Члена класса, структуры, объединения или перечисления
* Функции или функции члена класса
* Имени определения типа (typedef)
* Имени метки
* Имени макроса
* Параметра макроса

**Юникод** — **это** 16-разрядная **кодировка** символов, предоставляющая достаточное количество **кодировок** для всех языков. Все символы ASCII включены в **Юникод** **в** виде расширенных символов

**ASCII** — **это** таблица в которой символы сопоставлены числовым кодам. Каждый символ **ASCII** состоит из семи разрядов или семи бит, каждый из которых может принимать значение 0 или 1.

**UTF**-**8**-**это** кодировка символов переменной ширины, используемая для электронной связи. Определяемое стандартом **Unicode**, название происходит от формата преобразования **Unicode** (или Универсального кодированного набора символов) – 8-битного. **UTF**-**8** способен кодировать все 1,112,064 допустимые кодовые точки символов в Юникоде, используя от одного до четырех однобайтовых (8-битных) кодовых единиц.

**Язык** **низкого** **уровня** – **это** **язык** **программирования**, предназначенный для определенного типа компьютера и отражающий его внутренний машинный код; **языки** **низкого** **уровня** часто называют машинно-ориентированными **языками**.

**Язык среднего уровня**

**Высокоуровневый язык программирования** — язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков — это абстракция, то есть введение смысловых конструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде (или другом низкоуровневом языке программирования) очень длинны и сложны для понимания.

**Компилируемый язык программирования** — язык программирования, исходный код которого преобразуется компилятором в машинный код и записывается в файл с особым заголовком и/или расширением для последующей идентификации этого файла, как исполняемого операционной системой (в отличие от интерпретируемых языков программирования, чьи программы выполняются программой-интерпретатором).

**Интерпретируемый язык программирования** — язык программирования, исходный код на котором выполняется методом интерпретации

**Нативные языки программирования**

**Аллокация** - Означает выделение блока памяти, чаще всего в куче. Калька с английского **allocation** - распределение. Смысл дословный. Управлять памятью, распределять ее и т.п.

**Декларативное программирование** — это парадигма программирования … которая выражает логику вычисления без описания его потока управления.  
**Императивное программирование** — это парадигма программирования, в которой используются утверждения, которые изменяют состояние программы.

**Концепция типа данных -** Концепция типа очень важна в C++. Каждая переменная, аргумент функции и возвращаемое значение функции должны иметь тип, чтобы их можно было скомпилировать. Кроме того, перед вычислением каждого выражения (включая литеральные значения) компилятор неявно назначает ему тип.

**Переменная -** поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (то есть по данному адресу памяти), называются **значением** этой переменной (короче: переменная — именованный участок памяти, обладающий некоторым типом).

**Тип данных –**

* Тип данных — класс данных, характеризуемый членами класса и операциями, которые могут быть к ним применены (ISO/IEC/IEEE 24765-2010)[[2]](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#cite_note-2).
* Тип данных — категоризация абстрактного множества возможных значений, характеристик и набор операций для некоторого атрибута (IEEE Std 1320.2-1998)[[3]](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#cite_note-3).
* Тип данных — категоризация аргументов операций над значениями, как правило, охватывающая как поведение, так и представление (ISO/IEC 19500-2:2003)[[4]](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#cite_note-4).
* Тип данных — допустимое множество значений[[5]](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#cite_note-DateWritings-5).

**Диапазон типов по стандарту.**

**Знаковые типы.**

**Беззнаковые типы.**

**Типы фиксированной длины.**

**Тип представления размера.**

**Правила формирования общего типа.**

**Составные типы данных -** Для формирования других типов данных используют *основные* типы + так называемые *спецификаторы*. Типы данных, созданные на базе стандартных типов с использованием спецификаторов, называют *составными типами данных*. В C++ определены четыре спецификатора типов данных:

1. short - короткий;
2. long - длинный;
3. signed-знаковый;
4. unsigned-беззнаковый.

**Структура - это** сгруппированные (ая) под одним именем одна или несколько переменных (возможно, различных типов). В качестве этих переменных могут быть также массивы данных, указатели на различные типы данных, также переменные других типов **структур** и т.д.

**Объединение -** это объект, позволяющий нескольким переменным различных типов занимать один участок памяти. Объявление **объединения** похоже на объявление структуры: union union\_type {int i; char ch;}; Как и для структур, можно объявить переменную, поместив ее имя в конце определения или используя отдельный оператор объявления.

**Перечисление -**  это определяемый пользователем тип данных, состоящий из целочисленных констант. Для определения **перечисления** используется ключевое слово enum.

**Битовые поля**

**Ссылки.**

**Указатели.**

**Понятие, история и характеристика STL.**

**Ассоциативные контейнеры.**

**Последовательные контейнеры.**

**Стандартные функции ввода вывода.**

**Описание идентификатора.**

**Область видимости.**

**Класс памяти auto - Класс** **памяти** **auto** определяет автоматические переменные. По умолчанию, **это** локальные переменные, которые размещаются в стеке или внутренних регистрах процессора. «Время жизни» автоматической переменной ограничено временем выполнения функции (метода) или блока, в котором **эта** переменная определена.

**Класс памяти extern -** Если спецификатор extern размещается перед именем переменной, компилятор будет "знать", что переменная имеет внешнюю привязку, т.е. что память для этой переменной выделена где-то в другом месте программы. Внешняя "привязка" означает, что данный объект виден вне его собственного файла. По сути, спецификатор extern сообщает компилятору лишь тип переменной, не не выделяя для неё области памяти. Чаще всего спецификатор extern используется в тех случаях, когда одни и те же глобальные переменные используются в двух или более файлах.

**Класс памяти static -** это ключевое слово в C++, используемое для придания элементу особых характеристик. Для статических элементов выделение памяти происходит только один раз и существуют эти элементы до завершения программы. Хранятся все эти элементы не в heap и не на stack, а в специальных сегментах памяти, которые называются *.data* и *.bss* (зависит от того инициализированы статические данные или нет). На картинке ниже показан типичный макет программной памяти.

**Класс памяти register -**  Задание ключевого слова register – это есть указание компилятору выделить для сохранения данных объекта не ячейке стека, а внутренние регистры процессора. Но это не означает, что компилятор обязательно разместит данные объекта в регистрах процессора. Кроме того, при указании register, компилятор может разместить данные в кэш-памяти. Основной целью объявления переменной (объекта) с ключевым словом register есть обеспечение максимально быстрого доступа к этой переменной и обработки этой переменной.

**Пространство имен - это** декларативная область, в рамках которой определяются различные идентификаторы (**имена** типов, функций, переменных, и т. д.). **Пространства** **имен** используются для организации кода в виде логических групп и с целью избежания конфликтов **имен**, которые могут возникнуть, особенно в таких случаях, когда база кода включает несколько библиотек. Все идентификаторы в пределах **пространства** **имен** доступны друг другу без уточнения.

**Классы идентификаторов.**

**Основные правила использования переменных.**

**Конвенции именования С++.**

**Основные операции С++.**

**Поразрядные операции.**

**Операции инкремента и декремента.**

**Безопасное преобразование типов.**

**Опасное преобразование типов.**

**Форматирующие флаги и методы.**

**Тернарная операция.**

**Условный оператор.**

**Обработка запросов по множеству enum.**

**Оператор Switch.**

Оператор цикла: тело цикла, параметр цикла, условие цикла, итерация, счетчик цикла. Условие цикла. Цикл с предусловием. Цикл с постусловием. Цикл с параметром (формат и определение каждого элемента условия). Правила использования циклов. 5 операторов передачи управления в цикле.

**Определение функции.**

**Типичный механизм возврата значений.**

**Объявление функций (виды).**

**Возврат нескольких значений из функций.**

**Метод передачи параметров функции по значению: плюсы , минусы, технология.**

**Метод передачи параметров функции по ссылке: плюсы , минусы, технология.**

**Метод передачи параметров функции по константной ссылке: плюсы , минусы, технология.**

**Метод передачи параметров функции по указателю: плюсы , минусы, технология.**

**Стек.**

**Куча.**

**Принцип выбора метода передачи параметров функции. Возможности функций: перегрузка функций.**

**Рекурсия.**

**Правила использования рекурсии.**

**Шаблон функций.**

**Лямбда функция (пример, алгоритмы, итераторы).**

**Перегрузка операторов.**

**Перегрузка оператора ввода, вывода: особенности, примеры.**

**Перегрузка арифметических операторов: особенности, примеры.**

**Директивы.**

**Макросы.**

**Препроцессор.**

**Макро-константы.**

**Макрофункции (многострочные).**

**Predefined defines.**

**Минусы использования макросов.**

**Массив.**

**Статический массив: определение, использование.**

**Виды перебора элементов массива.**

**Обработка элементов массива по два элемента.**

**Многомерные массивы.**

**Хранение, описание, доступ к элементам.**

**Пример инициализации.**

**Передача параметров одномерного массива (виды).**

**Передача параметров многомерного массива (виды).**

**Понятие указатели и динамическая память.**

**Схема работы стек-куча при указателях.**

**Оператор new.**

**Инициализация объектов в куче.**

**Контейнеры (динамика).**

**Доступ к элементу в куче.**

**Указатель на функцию.**

**Указатель на объект.**

**Указатель на void.**

**Инициализация указателей (4 вида).**

**Освобождение памяти.**

**Операции с указателями.**

**Арифметические операции с указателями.**

**Ссылки.**

**Взаимосвязь ссылки и указателя.**

**Указатели и массивы.**

**Динамические одномерные массивы.**

**Динамические двумерные массивы: порядок создания и освобождения памяти.**

**Контейнер vector. назначение, инициализация, принцип работы.**

**Понятие указатели и динамическая память.**

**Схема работы стек-куча при указателях.**

**Инициализация объектов в куче.**

**Контейнеры (динамика).**

**Доступ к элементу в куче.**

**Указатель на функцию.**

**Указатель на объект.**

**Указатель на void.**

**Инициализация указателей (4 вида).**

**Освобождение памяти.**

**Операции с указателями.**

**Арифметические операции с указателями.**

**Ссылки.**

**Взаимосвязь ссылки и указателя.**

**Указатели и массивы.**

**Динамические одномерные массивы.**

**Динамические двумерные массивы: порядок создания и освобождения памяти.**

**Итераторы.**

**Концепция полуинтервалов.**

**Арифметические операции с итераторами.**

**Шаблонная функция вывода контейнера по двум итераторам.**

**Алгоритмы и итераторы (5 примеров, любых).**

**Исключения: механизм, примеры.**

**Типы исключений.**

Алгебра (группа, кольцо, ассоциативное кольцо). Отношение сравнения (свойства). Криптография. Стеганография. Классическая задача передачи сообщений от некоторого отправителя А к получателю В. Классическая система секретной связи. Односторонняя функция. Дискретных логарифм. Первая система с открытым ключом Криптопротокол Диффи-Хеллмана. Криптопротокол Шамира. Криптопротокол Эль-Гамаля. Криптопротокол RSA. Простое число. Распределение простых чисел (Эратосфен). Основная теорема арифметики. Функция Эйлера, основные свойства. Каноническое разложение числа. Функция Эйлера. Теорема Эйлера. Теорема Ферма. Наибольший общий делитель – алгоритм Евклида. Расширенный алгоритм Евклида. Обобщенный алгоритм Евклида. Алгоритм возведения в степень по модулю.