**Теория для подготовки к экзамену по ВП за первый семестр**

**Алгоритм и его свойства**

Алгоритм – конечная последовательность действий, приводящая к ожидаемому результату. Также можно отметить, что эта последовательность действий над определёнными данными, которые по результату применения алгоритма меняются определённым образом.

Свойства алгоритма:

* Дискретность – шаги алгоритма могут делиться на более мелкие шаги, таким образом решение большой задачи разбивается на более мелкие подпункты. Из этого следует, что любой шаг алгоритма является сам алгоритмом и так далее, пока мы не дойдём до логически простого шага.
* Понятность – любой алгоритм должен быть понятен для исполнителя команд, если он для него написан. Другими словами, исполнитель должен знать, что ему делать.
* Определённость – каждый шаг алгоритма должен пониматься однозначно. Например, если вы говорите сходи в магазин и купи хлеб – то это будет плохой алгоритм, ведь мы не указали какой хлеб, как добраться до магазина и т д Данный алгоритм не определённый. Также, хорошим примером не определённого алгоритма будет указание: «Найди то - не знаю что». Где искать, что искать? – не понятно.
* Результативность – алгоритм всегда должен приводить к определённому результату, который мы от него ожидаем.
* Конечность – алгоритм должен состоять из конечного числа команд, что, собственно, логично.
* Эффективность – алгоритм должен приводить к желаемому результату за наименьшее число шагов.
* Массовость – алгоритм должен обрабатывать любые данные поставленной задачи. Например, если вы хотите добраться до дома, то ваш алгоритм должен вести вас из любой точки, где вы находитесь и в любую другую точку, где бы не находился ваш дом.

Эффективным и правильным алгоритмом является тот, который удовлетворяет всем своим свойствам.

**Этапы компиляции программы на языке C++**

Компилятор языка C++ состоит из следующих модулей:

* **Препроцессор** — это макро процессор, который преобразовывает вашу программу для дальнейшего компилирования. На данной стадии происходит работа с препроцессорными директивами. Например, препроцессор добавляет заголовки в код (#include), убирает комментирования, заменяет макросы (#define) их значениями, выбирает нужные куски кода в соответствии с условиями #if, #ifdef и #ifndef. Полученный препроцессированный код в выходном файле **driver.ii** (прошедшие через стадию препроцессинга C++ файлы имеют расширение **.ii**), используя флаг **-E**, мы сообщим компилятору, что нам нужно получить только препроцессорный файл.
* **Компилятор** - 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. Используя флаг **-S**, который сообщает компилятору остановиться после стадии компиляции, получим ассемблерный код в выходном файле **driver.s**:
* **Ассемблер** — это доступное для понимания человеком представление машинного кода. Ассемблирование кода, это перевод его из представления понятного человеком полностью в машинный код, который храниться 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. Далее возможно сохранение данного объектного кода в статические библиотеки для того, чтобы не компилировать данный код снова. Преобразовываем машинный код понятный человеку с помощью ассемблера (as) в выходной объектный файл **driver.o**
* **Линкер (компоновщик)** - 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.

**Директивы препроцессора**

Как было сказано ранее, препроцессор — это некий настройщик файлов исходного кода для компилятора. С его помощью можно на этапе сборки программы каким – то образом обработать написанный код, например заменить одни слова на другие, вставить, или удалить определённые участки кода из готовой программы, или добавить зависимости. Рассмотрим самые часто используемые директивы макро – процессора. Напомню, что каждая директива начинается с символа решётки – «#»

**Директивы:**

* **#define** - директиву **#define** можно использовать для создания макросов. Макрос — это правило, которое определяет конвертацию идентификатора в указанные данные.
* **#undef** – противоположность **#define**, удаляет макрос, созданный этой директивой.
* **#ifdef; #endif** – позволяет препроцессору проверить, было ли значение ранее определено с помощью директивы **#define**. Если да, то код между **#ifdef** и **#endif** скомпилируется. Если нет, то код будет проигнорирован.
* **#ifndef; #endif** — это полная противоположность к **#ifdef**, которая позволяет проверить, не было ли значение ранее определено.
* **#include** – данная директива вставлять содержимое файла в место, где была использована. Она имеет две формы: **#include <filename>**, которая сообщает препроцессору искать файл в системных путях (в местах хранения системных библиотек языка С++). Чаще всего вы будете использовать эту форму при подключении заголовочных файлов из Стандартной библиотеки C++. **#include "filename"**, которая сообщает препроцессору искать файл в текущей директории проекта. Если его там не окажется, то препроцессор начнет проверять системные пути и любые другие, которые вы указали в настройках вашей IDE. Эта форма используется для подключения пользовательских заголовочных файлов.
* **#pragma** – данная директива запускает какие – то встроенные сценарии из макро – процессора. Чаще всего её можно увидеть с запуском сценария – **“once”**.
* **#pragma once** – данная инструкция, вставленная в начале заголовочного файла позволяет избежать дублирования кода при множественном вызове **#include**, но **#pragma once** не является официальной частью языка C++, и не все компиляторы её поддерживают (хотя большинство современных компиляторов поддерживают). Вместо неё рекомендуется использовать **«header guards»,** или, иными словами, явную проверку **#ifndef**. Синтаксис и описание смотреть по ссылке: [https://ravesli.com/urok-23-header-guards/](https://ravesli.com/urok-23-header-guards/#toc-2)

Для полного понимания работы препроцессора, обязательно переходим по следующей ссылке и читаем: <https://ravesli.com/urok-22-preprotsessor-direktivy/>

**Заголовочные файлы**

Полная информация по заголовочным файлам находится по ссылке: <https://ravesli.com/urok-21-zagolovochnye-fajly/>

Далее в документе будут часто появляться ссылки на информацию по темам, ибо зачем дублировать то, что и так есть?

**Типы данных**

Типы данных в программировании — это некие единицы информации. Вспомним, что память компьютера (ОЗУ) является памятью произвольного доступа, а это значит, что к любой её части можно обратиться напрямую. Минимальная часть памяти, к которой можно обратиться это 1 байт. Но не все данные могут поместиться в 1 байт, например возьмём число 25, да, оно превосходно уляжется в этот размер, а если 25 в 4 – ой степени? Для такого числа уже будет недостаточно ячейки в 1 байт, что делать? Для таких случаев и придумали типы данных, некие контейнеры, которые группируют байты памяти, в которых лежат части полезной информации. Так число 25 в 4 – ой степени, вполне поместиться в тип **int** – это самый распространённый тип данных, который хранит числовую информацию. Его размер – 4 байта, исходя из этого, можно легко вычислить, какое максимальное число можно в него поместить. Но, также стоит отметить, что тип **int** является числовым типом со знаком, это значит, что он хранит положительные и отрицательные значения. Эти значения делятся пополам, таким образом размер максимального числа, которое может хранить **int** уменьшается.

Помимо размещения данных в памяти, тип данных отвечает и за операции, которые можно выполнять над данными. Так, над целыми числами типа: **int**, можно выполнять операции взятия остатка от деления и целой части. Но взятие остатка от деления в типе **double** уже будет нельзя. Также, тип данных иногда характеризует расположение в памяти объекта. В какую ячейку его эффективнее положить.

**Характеристика типов данных в C++**

Типы данных в C++ глобально делятся на **простые** и **составные**.

Простые типы данных – это фундаментальные единицы компилятора, из которых строятся более сложные типы – структурные (составные). Как правило простые типы данных хранят только одно значение. Базовые типы предопределены стандартом языка, указываются зарезервированными ключевыми словами и характеризуются одним значением. Их не надо определять и их нельзя разложить на более простые составляющие без потери сущности данных. Базовые типы объектов создают основу для построения более сложных типов.

Простые типы данных подразделяются на **целочисленные, вещественные, символьный, логический и тип void**. По-хорошему, символьный относится к целочисленным типам, т к хранит код символа, но его выделяют в отдельную категорию, т к он обрабатывается компилятором со своими особенностями. Стоит заметить, что логический тип данных, также не относится к целочисленным! В ходе написания программ вам может показаться, что это одно и тоже, по сколько единица и нуль спокойно могут стать сигналами true и false в логических выражениях, но это не так. Это всего лишь неявные преобразования типов, которые очень часто происходят в языке C++. Чтобы не углубляться в механику работы компилятора, просто следует запомнить, что тип **bool** отличается от целочисленных типов. Также отдельного внимания заслуживает тип **void**. Его можно с трудом отнести к типам данным, по сколько он имеет в своём арсенале два узкоспециальных применения: как сигнал пустого значения в заголовках функций и как тип неопределённого указателя. Его также относят к типам простым типам данных, но по факту это просто ключевое слово.

Для простых типов данных существую спецификаторы типов, уточняющих внутреннее представление и диапазон базовых типов:

* short (короткий)
* long (длинный)
* signed (знаковый)
* unsigned (беззнаковый)

Составные типы – это типы данных, которые являются контейнерами для простых типов, они их могут объединять, тем самым создавая новые структуры данных. К ним относятся **массивы,** **структуры, классы, перечисления, указатели, ссылки**.

В отдельную категорию типов данных также выносят объекты типов класса и структур. Созданные переменные этих типов считаются носителями своего, уникального типа данных. Сюда можно отнести потоки ввода/вывода, файловые потоки и т д

Остальная информация про типы данных находится по ссылке: <https://metanit.com/cpp/tutorial/2.3.php>

Все операторы языка C++, которые можно применять при работе с объектами, вы можете уточнить по ссылке: <http://cpp-cpp.blogspot.com/2013/10/c_4.html>

Полную информацию про все типы данных в языке C++, вы можете найти здесь: <http://rjaan.narod.ru/docs/main_operations_with_types_in_cpp.html>

**Ввод и вывод данных на консоль**

Про ввод/вывод на консоль вы можете прочитать здесь:

<https://ravesli.com/urok-11-sout-cin-i-endl/>

Про потоки ввода/вывода вы можете прочитать здесь: <https://ravesli.com/urok-207-potoki-vvoda-vyvoda/>

**Операторы условий**

Прочитать про оператор условия можно здесь:

<https://ravesli.com/urok-64-operatory-uslovnogo-vetvleniya-if-i-else/>

<https://ravesli.com/urok-65-operator-switch/>

**Операторы циклов**

Прочитать про операторы циклов и работу с ними можно здесь:

<https://ravesli.com/urok-67-tsikl-while/>

<https://ravesli.com/urok-68-tsikl-do-while/>

<https://ravesli.com/urok-69-tsikl-for/>

<https://ravesli.com/urok-70-operatory-break-i-continue/>

**Специальные операторы**

<https://ravesli.com/urok-41-sizeof-zapyataya-i-uslovnyj-ternarnyj-operator/>

**Массивы данных и работа с памятью**

Про работу с памятью, массивы, динамические переменные читаем весь блок статей (внизу каждой статьи есть кнопочка перехода к следующей теме)

<https://ravesli.com/urok-74-massivy-chast-1/>

**Функции и работа с ними в языке C++**

Блок про функции и работу с ними читаем здесь, также переходим по всем темам далее:

<https://ravesli.com/urok-96-parametry-i-argumenty-funktsij/>

**Работа со структурами**

Всё про структуры:

<https://ravesli.com/urok-61-struktury/>

**Объединения**

Всё про тип объединения:

<https://metanit.com/cpp/c/6.6.php>

**Перечисления**

Всё про тип перечисления:

<https://ravesli.com/urok-58-perechisleniya-tip-enum/>

**Работа с текстовыми файлами**

Текстовыми файлами считаются все файлы, когда при работе с ними не используется модификатор открытия файла: **binary.**

Информация по работе с текстовыми файлами по ссылкам ниже:

<https://ravesli.com/urok-207-potoki-vvoda-vyvoda/>

<https://ravesli.com/urok-208-funktsional-klassa-istream/>

<https://ravesli.com/urok-212-bazovyj-fajlovyj-vvod-vyvod/>

<https://ravesli.com/urok-213-randomnyj-fajlovyj-vvod-vyvod/>

**Работа с файлами в бинарном режиме**

Вся информация здесь:

<http://ci-plus-plus-snachala.ru/?p=86>

**Динамические структуры данных: стек и списки**

Информация здесь:

<http://ci-plus-plus-snachala.ru/?p=86>

<https://younglinux.info/c/stack>

<http://elisey-ka.ru/c/devcpp_4.pdf>

**Вопросы для самоконтроля:**

1. Что такое алгоритм?
2. Какие свойства алгоритма вы знаете?
3. Каким должен быть алгоритм?
4. Назвать этапы компиляции программы на C++
5. Назвать расширения файлов на различных этапах компиляции
6. Что такое заголовочный файл?
7. Для чего нужны заголовочные файлы?
8. Что помещают в заголовочный файл?
9. Как подключить заголовочный файл?
10. Что такое препроцессор?
11. Что такое директивы препроцессора?
12. Назовите три любых директивы препроцессора
13. Что такое тип данных в C++?
14. Назовите классификацию типов данных в C++
15. Назовите базовые типы данных в C++
16. Перечислите все операции с простыми типами данных
17. Что такое переменная?
18. Область видимости переменных
19. Пространство имён, что такое?
20. Для чего нужно пространство имён?
21. Как производится ввод и вывод данных в C++
22. Какими способами можно произвести вывод на экран?
23. Что такое поток вывода?
24. Какими способами можно считать данные с консоли?
25. Что такое объект ввода?
26. Как производится проверка ошибочного ввода?
27. Что такое оператор условия?
28. Как работает оператор условия в C++
29. Сколько существует операторов условия в C++
30. Способы ветвления программы в C++
31. Циклы в C++
32. Виды циклов
33. Структура памяти программы
34. Разделы памяти
35. Назначения разделов памяти
36. Что такое массив данных?
37. Как представлен в памяти?
38. Что представляет из себя переменная, хранящая массив?
39. Какие массивы бывают по глубине?
40. Как реализуется глубина массива?
41. Что такое указатель?
42. Что хранит указатель?
43. Какой тип у указателя?
44. Указатель на константу
45. Константный указатель
46. Что такое ссылка?
47. Отличия ссылки от константного указателя
48. Арифметика указателей
49. Динамические переменные
50. Как создать динамическую переменную?
51. Как использовать?
52. Как удалить?
53. Как создать динамический массив?
54. Отличие динамического массива от статического массива
55. Отличие указателя от имени массива (статического)
56. Отличие указателя от имени динамического массива
57. Создание двумерного динамического массива
58. Удаление одномерного массива
59. Удаление двумерного массива
60. Представление строк в C++
61. C-строки, что такое?
62. Как работать?
63. Чем отличаются от массива?
64. Динамические C-строки
65. Другие составные типы данных
66. Что такое перечисление
67. Как использовать?
68. Что такое объединение
69. Как использовать?
70. Что такое структура?
71. Как называются переменные в структуре?
72. Как называются функции в структуре?
73. Как создать структуру?
74. Как создать объект структуры?
75. Как инициализировать структуру?
76. Как обращаться к переменным и функциям объекта структуры?
77. Как создать динамический объект структуры?
78. Как обращаться к переменным и функциям у указателя на объект?
79. Что такое функция?
80. Для чего нужна функция?
81. Для чего нужно функциональное программирование?
82. Назовите части функции
83. Какие есть функции?
84. Как вызывается функция?
85. Как передаётся значение в функцию?
86. Виды передачи значений в функцию
87. Как происходит возврат значения из функции?
88. Что такое рекурсивная функция?
89. Как работает?
90. С какими файлами можно работать в C++?
91. Общий подход к работе с файлами
92. Способы открытия файла
93. Работа с текстовыми файлами
94. Работа с бинарными файлами
95. Файловый указатель, что это?
96. Как получить позицию файлового указателя?
97. Как переместить файловый указатель?
98. Как открыть файл на чтение и запись одновременно?
99. Потоки данных
100. Приведите классификацию динамических структур данных
101. Что такое линейная структура данных?
102. Примеры
103. Что такое не линейная структура данных?
104. Примеры
105. Что такое массив?
106. Что такое циклический массив?
107. Что такое стек?
108. Что такое очередь?
109. Что такое дек?
110. Что такое список?
111. Какие списки бывают?
112. Как организовать список на C++?

**Всем ни пуха не пира, прочитав данный конспект вы точно сдадите экзамен!**