Язык С.
Введение.

История создания. Авторы.



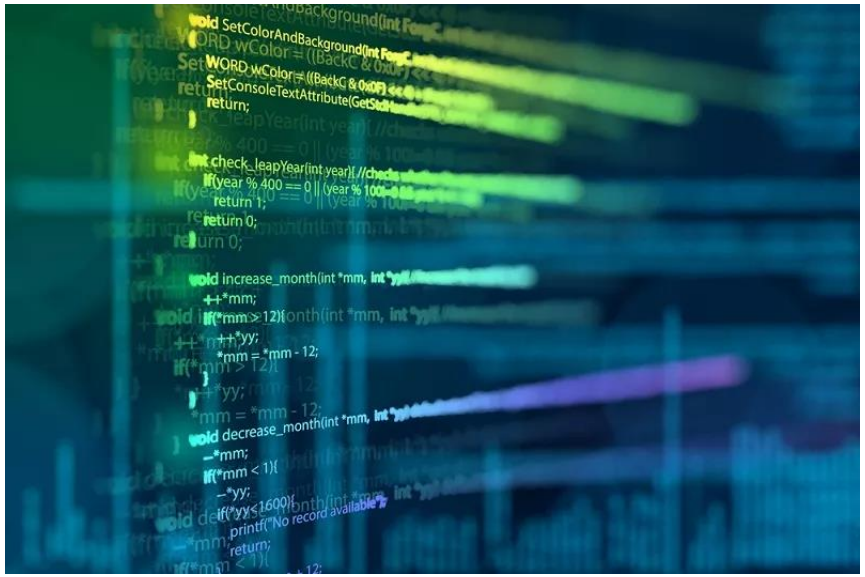
- **Си** — компилируемый статически типизированный язык программирования общего назначения, разработанный в 1969—1973 годах. Разработчики - **Брайан Керниган** (Brian Kernighan), **Деннис Ритчи** (Dennis Ritchie) и **Кен Томпсон** (Ken Thompson). Первоначально он был разработан для реализации операционной системы UNIX, но, впоследствии, был перенесён на множество других платформ.

Основные характеристики

- **Си**— компилируемый статически типизированный язык программирования общего назначения

- Ключевая характеристика языка **Си** – его минималистичность. Авторы задумывали его, как простой инструмент для написания программ, которые будут легко компилироваться с использованием однопроходного компилятора. При этом чтобы в результате для каждого элемента программы было задействовано как можно меньше машинных команд.

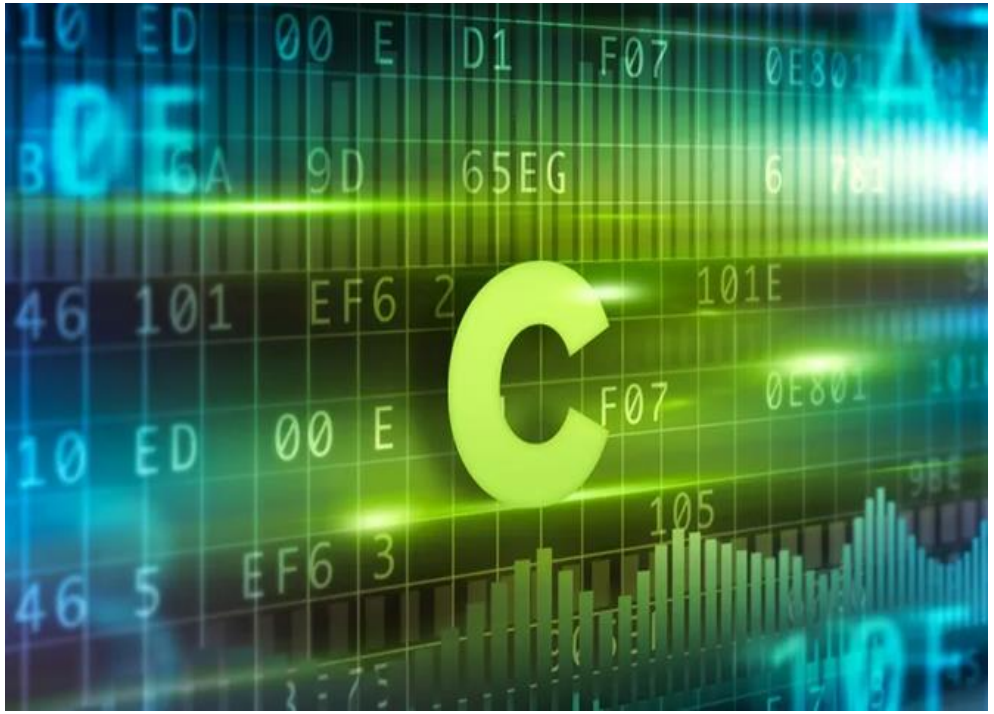
- Встречается формулировка, что **Си** – это «**универсальный ассемблер**», что подчеркивает универсальность стандарта **Си**. Его код компилируется на любом компьютере, практически не меняясь (в то время как в языках ассемблера нет единых стандартов для всех платформ). **Си** относят к языкам **среднего и низкого уровня**, потому что по работе он максимально близок к реальным устройствам.



Однопроходный компилятор - не возвращается к тексту, который уже прошел компиляцию. Поэтому сначала идет объявление функции, а затем уже следует ее использование.

Отличительные черты

- **понятная языковая база**, отдельные библиотеки с матем. функциями, с функциями управления файлами, и т.д.;
- **направленность на процедурное программирование**;
- **система типов**, исключающая появление бессмысленных операций;
- есть **специальные указатели**, дающие доступ к памяти компьютера;
- **ключевых слов – по минимуму**;
- **параметры передаются в функцию по значению**, не по ссылке. А для передачи по ссылке используются указатели;
- **указатели для функций и статических переменных**;
- обозначены **области действия имен**;
- **записями** (пользовательские типы данных) можно управлять как **единой структурой**.

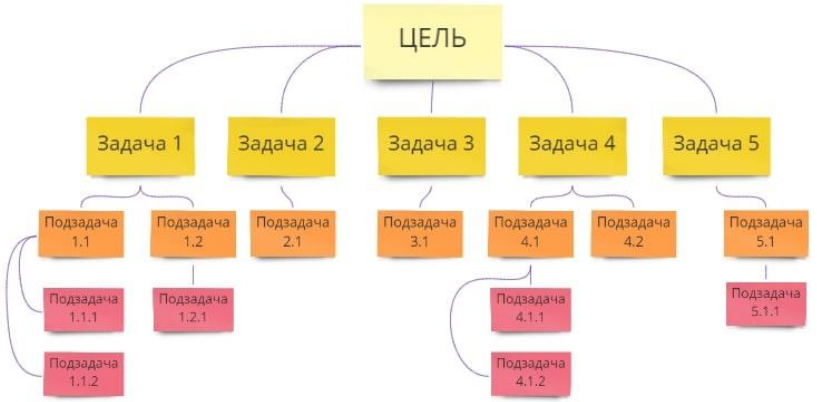


«Процедурный подход». I

Этапы создания ПО

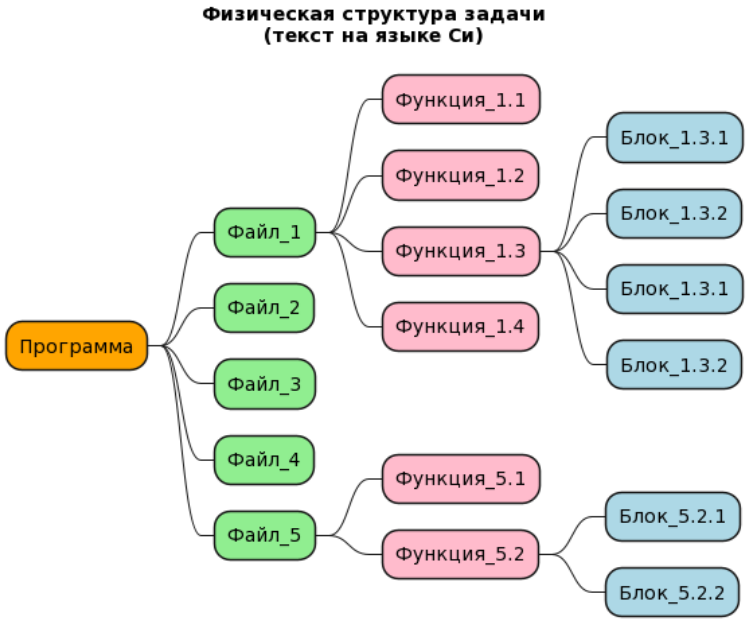
«Процедурный подход». II

Декомпозиция задачи и Модульность проекта



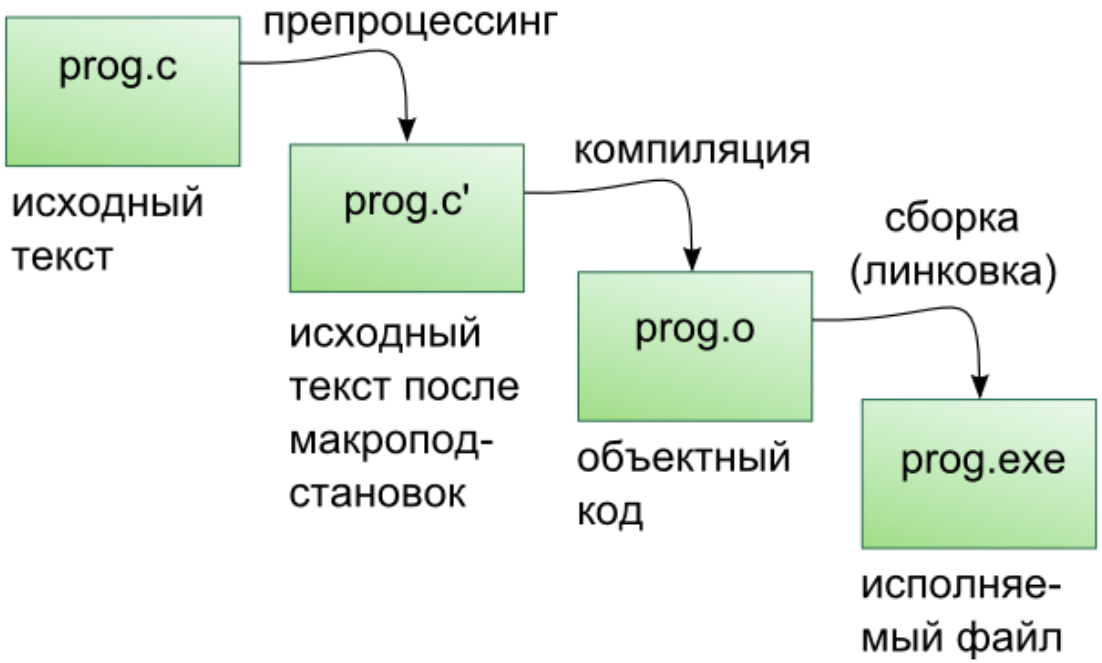
Структурное программирование как метод создания текста программы в котором:

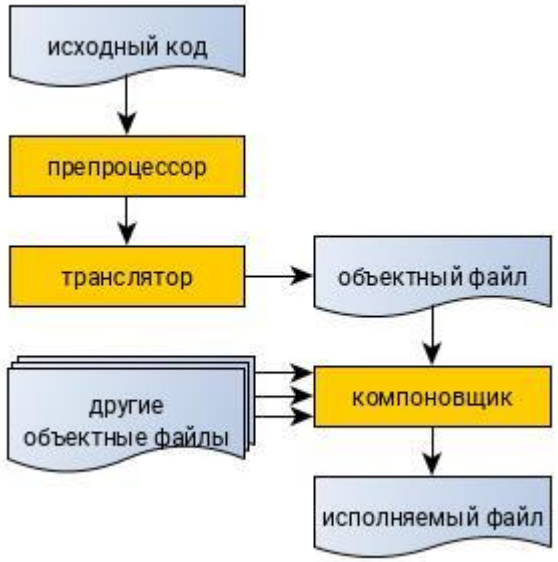
- Отображается структура решаемой задачи (логическая структура).
- Разбиение программы на файлы для лучшего понимания другими программистами, а не только автором (модульность).



«Процедурный подход». III

Рецепт получения исполняемого файла.





На этапе компиляции программисты могут получить: синтаксические ошибки и ошибки неописанных внешних объектов.
На этапе компоновки(сборки) – ошибки неразрешенных или неуникальных внешних зависимостей.

Некоторые термины и определения

- **Исходный текст** – текст программы выполненный в текстовом редакторе на ЯП;
- **Препроцессинг** – предварительная обработка текста исходного файла программой – препроцессором. Результат работы препроцессора – окончательно сформированный текст программы с которым будет работать компилятор;
- **Компиляция** – синтаксический, лексический анализ, затем трансляция в машинный код в результате получается промежуточный формат файла в объектном формате (*.o, *.obj). На этом этапе компилятору нужно знать только прототип функции и тип переменной. Компилятор «откладывает на потом» разрешение внешних зависимостей;
- **Линковка** (син. - **сборка, компоновка**) – соединение всех ранее откомпилированных частей (а так же статич. библиотек: *.lib) в единый исполняемый файл. На этом этапе происходит окончательное распределение памяти и формирование адресов всех команд;

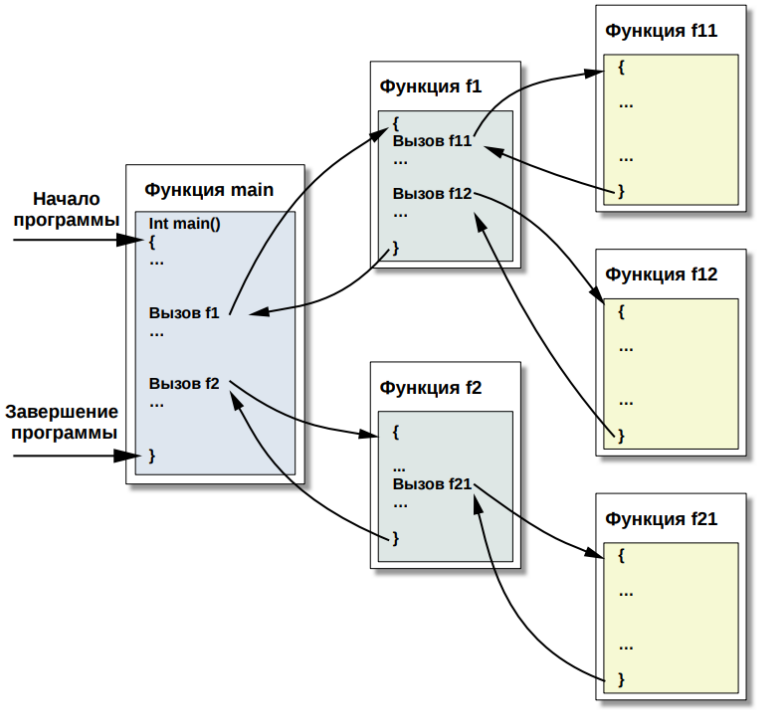
«Процедурный подход». IV

Функция и функциональная декомпозиция.

- **Функция** – фрагмент кода, оформленный определенным образом и выполняющий некоторое законченное действие. На нее можно передать управление (вызвать) из любого места в программе. По завершению работы функции, выполнение программы будет продолжено за точкой ее вызова. Функции позволяют использовать один и тот же код для работы с разными наборами данных. Являются базовым понятием процедурного программирования.

- **Функциональная декомпозиция** – представление программы в виде иерархий вложенных вызовов функций.

- Обычно несколько функций помещаются в один файл (модуль), **объединяя их** по какому–то **логическому принципу**. При этом вызов функции может быть в другом файле (модуле). В этом случае программист должен объяснить компилятору где искать ту или иную функцию, вызванную из данного модуля и которая является внешней по отношению к этому модулю.



CodeStyle. I

- **Названия макросов и макрофункций** пишутся капсом, слова в названиях отделяются друг от друга нижним подчеркиванием.

```
#define MAX_ARRAY_SIZE    32
#define INCORRECT_VALUE   -1
```

- **Названия переменных** записываются в нижнем регистре, а слова в названиях отделяются нижним подчеркиванием

```
int my_int_variable = 0;
char *hello_str = "hello_habrahabr";
pid_t current_pid = fork();
```

- **Названия библиотечных функций** общего пользования пишутся одним словом, иногда сокращенным, но это слово передает суть функции — что она должна делать.

```
read();
fork();
write();
```

- **Специализированные функции** (которые вызываются в пределах работы внутри какого-то специфичного контекста) лучше называть так, чтобы было определенно ясно, что делает эта функция.
- i, j, k — стандартные названия для **итераторов цикла**
- Соблюдайте **однородность переноса скобок**
- Объявляйте переменные в начале функции.

CodeStyle. II

Однострочные комментарии — это комментарии, которые пишутся после символов //

Многострочные комментарии — это комментарии, которые пишутся между символами /* */

Никогда не используйте вложенные комментарии!

Пишите комментарии с умом:

- Плохой комментарий хуже, чем его отсутствие.
- Не стоит комментировать то, что и так очевидно.
- На уровне библиотек/программ/функций комментарии отвечают на вопрос «ЧТО?»: «Что делают эти библиотеки/программы/функции?».
- На уровне выражений (однострочного кода) комментарии отвечают на вопрос «ПОЧЕМУ?»: «Почему код выполняет задание именно так, а не иначе?».

Алфавит языка С

- Набор символов (знаков), который допустим в данном языке.

- прописные и строчные буквы латинского алфавита (A ... Z, a ... z)
- цифры: 0 - 9
- специальные знаки: " , { } | [ ] ( ) + - / % \ ; ' . : ? > < = _ & ! * # ~ ^
- неотображаемые символы (пробел, табуляция, переход на новую строку)
- В комментариях, строках и символьных константах могут использоваться другие литеры (например, русские буквы).

Лексемы языка С

Из символов алфавита составляются лексемы - минимальные единицы языка, имеющие самостоятельный смысл:

- константы;
- имена объектов;
- ключевые слова;

} **Идентификаторы**

- знаки операций;
- разделители(скобки, точка, запятая, пробельные символы).

Идентификаторы С

Идентификатором называется имя, сопоставленное объекту языка, (например: *переменной*, *функции* или *служебным словам языка*).

Идентификатор представляет собой последовательность строчных, прописных букв латинского алфавита, цифр и знака подчеркивания.
Причем идентификаторы должны начинаться либо с буквы, либо со знака подчеркивания.

Строчные и прописные буквы рассматриваются компиляторами языка Си как несовпадающие.

Примеры идентификаторов:

- `int`
- `_test_id128`

Ключевые слова C/C++

auto	case	const	default
double	enum	float	goto
int	register	short	sizeof
struct	typedef	unsigned	while
break	char	continue	do
else	extern	for	if
long	return	signed	static
switch	union	void	volatile

catch	try	this
protected	friend	new
class	virtual	template
delete	inline	operator
public	throw	private

Операторы С

- **Оператор** — это символ (несколько символов), который дает команду компилятору выполнить определенные матем. или логические функции.

  Унарный – 1 операнд,

  Бинарный – 2 операнда,

  Тернарный – 3 операнда

Тип операции	унарные	бинарные	тренарные
Арифметические	+, -	+, - , *, / , %	
Логические	!	||, &&	
Сравнение		<, >, >=, <=, ==, !=	
Условные			x ? y : z
Битовые	~	&, |, ^, >>, <<	
Присваивание	++, --	=, +=, -=, *=, /=, %=	
Последовательность выражений		, (запятая)	
Преобразование типа	(тип)		

Операнд - аргумент операции; данные, которые обрабатываются командой

Приоритет операторов

Приоритет оператора – величина, определяющая преимущественное право на выполнение той или иной операции.

Ассоциативность оператора – порядок выполнения вычислений при одинаковом приоритете.
A=B=C ⇔ A=(B=C) – справа налево;
A+B+C ⇔ (A+B)+C – слева направо;

Приоритет	Оператор(-ы)	Описание	Ассоциативность
Наивысший, 1	::	Оператор определения контекста переменных (разрешение области видимости)	
2	++, --, (), [], ., ->, typeid(), …-cast	постфикс-, вызов ф-ции, индексация массива, выбор элем. по ссылки, выбор эл. по указателю, инф. о типе, приведение типа	Слева - направо
3	++, --, +, -, !, ~, (type), *, &, sizeof(), new [], delete []	Префикс, унарный плюс, унарный минус, логическое нет, побит. отр., преобр.типа, выбор знач. по ссылке, взятие адреса, размер объекта указ. типа, динамическое выделение и освобождение памяти	Справа -налево
4	.*, ->*	Указатель на член класса	Слева - направо
5	*, /, %	Умножение, деление, остаток от деления	Слева - направо
6	+, -	Сложение , вычитание	Слева - направо

Приоритет	Операторы	Описание	Ассоциативность
7	**>>, <<**	Побитовый сдвиг вправо, побитовый сдвиг влево	Слева - направо
8	**>>, >>=, <<, <<=**	Операторы сравнения	Слева - направо
9	**==, !=**	Оператор равенства, оператор неравенства	Слева - направо
10	**&**	Побитная И	Слева - направо
11	**^**	Побитная исключ. ИЛИ	Слева - направо
12	**|**	Побитная ИЛИ	Слева - направо
13	&&	Логическое И	Слева - направо
14	**||**	Логическое ИЛИ	Слева - направо
15	**?:**	Тернарный	Справа - налево
16	**=, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=**	Операторы присваивания	Справа - налево
17	**throw**	Оператор генерации исключения	Справа - налево
Самый низкий, 18	**,**	Запятая	Слева - направо

Анатомия С программы

```
#include <stdio.h>              // Подключение библиотеки
int main()                      // Главная функция
{
   printf("Hello, world!");     // Вывод сообщения
   getchar();                   // Задержка окна консоли
   return 0;                    // Выход из функции
}


int main(){}                    // Тоже работает !!!
```

Требует ознакомления!

- https://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf

- https://www.onlinegdb.com/online_c_compiler