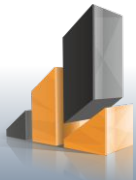
## Компиляция

Большинство реализаций языка Си - компиляторы. Компиляция программ происходит в три стадии:

- обработка препроцессором
- компиляция в объектный код (иногда сначала в ассемблер или псевдокод для упрощения портирования на другие архитектуры)
- компоновка (сборка) всех модулей в выполняемый файл или разделяемую библиотеку
- Набор программ для компиляции называют toolchain. Часто сюда включается и стандартная библиотека Си

# Препроцессор

Препроцессор выполняет предварительную обработку текста программы. В его функции входит:

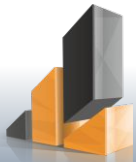
- замена констант на значения и разворачивание макросов
- обработка директив условной компиляции
- включение в текст внешних файлов
- Вообще говоря, препроцессор не имеет отношения к языку Си и может обрабатывать любые тексты

После обработки получается текст, который может быть скомпилирован в объектный файл без использования каких либо внешних файлов.

Посмотреть результат обработки препроцессором можно, например, командой

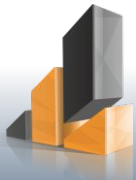
*gcc -E main.c -o main.out.c*

(Файл main.out.c будет содержать результат работы препроцессора)

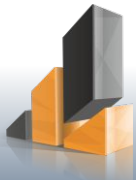
# Компиляция

В процессе компиляции файл на языке Си преобразуется в объектный файл, содержащий машинный код, таблицу содержащихся в файле символов и таблицу необходимых этому файлу внешних символов.

# Компоновка

В процессе компоновки один или несколько объектных файлов преобразуются в выполняемый образ. На этом этапе происходит разрешение ссылок на внешние символы.

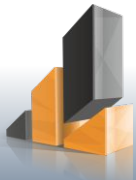



## Условная компиляция

Синтаксис:

```
#ifdef ИМЯ
текст-1
#endif

#ifdef ИМЯ
текст-1
#else
текст-2
#endif
```

Оставляет текст-1 если константа ИМЯ определена при помощи #define и текст-2 в противном случае.

## Организация модулей

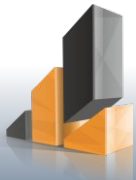
Компиляция программы на Си происходит в три этапа:

- обработка текста препроцессором
- компиляция каждого файла с исходным текстом
- сборка скомпилированных файлов в один (компоновка)

Препроцессор позволяет не усложнять язык Си средствами работы с модулями и их подключением.

Обычно реализация модуля находится в файле с расширением .c, а прототипы, константы, описания типов, которые могут потребоваться для других модулей - в файлах с расширением .h. Таким образом, если модуль А использует функции модуля Б, то достаточно подключить файл Б.h при помощи директивы #include и после обработки получится файл, пригодный для компиляции, т.к. содержимое Б.h с прототипом функций будет подставлено вместо #include.

Обычно определения модуля A.c выносятся в одноимённый h-файл, однако ничто не запрещает объединить определения модулей a.c и b.c в файл my-headers.h или разделить описания модуля a.c на файлы a1.h и a2.h.

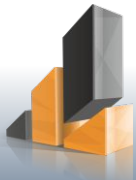



**Makefile**

Для автоматизации сборки множества модулей используются системы сборки. Стандартная система сборки – **make**. (VS – nmake, GNU – gmake, BSD – bmake, Watcom – wmake, и т.д.)

Программа make читает Makefile в текущей директории и выполняет указания, содержащиеся в нём.

```
myprogram: module1.o program.o
        gcc -o myprogram module1.o program.o

module1.o: module1.c module1.h
        gcc -c -o module1.o module1.c

program.o: program.c module1.h
        gcc -c -o program.o program.c
```

По умолчанию выполняется первая цель.

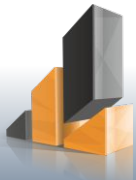
Отслеживаются даты изменения файлов, модули не пересобираются без необходимости.

Отслеживаются зависимости.

Возможна параллельная сборка

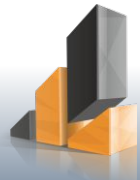
Обычно используют цель all. Есть возможность сократить запись используя переменные $@ и $<.

```
all: myprogram

myprogram: module1.o program.o
	gcc -o $@ $<

module1.o: module1.c module1.h
	gcc -c -o $@ $<

program.o: program.c module1.h
	gcc -c -o $@ $<
```

GNU Make Manual

https://www.gnu.org/software/make/manual/

## Система контроля версий

Предназначена для сохранения истории изменений текстовых файлов.
В общем случае подразумевает наличие

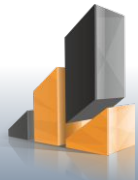
- Команды скачать изменения из репозитория
- Команды для сохранения изменения в репозиторий (комита)
- Возможность разрешения конфликтов, если файл был исправлен несколькими разработчиками

Типы VCS:

- Распределенные
- Централизованные

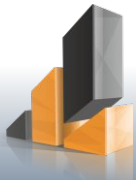
Реализации:

- Subversion
- Git
- CVS
- Mercurial
- Darcs
- Bazaar
- rcs

## Git/Svn

- получить копию репозитория
  - git clone <url>
  - svn checkout <url>
- Обновить файлы с сервера, совмещая с локальными изменениями
  - git pull
  - svn up
- Добавить файл в VCS
  - git add <file>
  - svn add <file>
- Закомитить свои изменения
  - git commit -m ‘комментарий’ && git push
  - svn commit -m ‘комментарий’
- Посмотреть список правок
  - git log
  - svn log
- Документация
  - https://git-scm.com/docs/gittutorial
  - http://svnbook.red-bean.com/en/1.7/index.html

## Отладчик

Запуск: gdb <file.elf>
Файл должен быть собран с ключом -g для компиляции и компоновки gcc

Команды gdb

- tui enable
- br <function> или br <file.c:123>
- r
- c
- n
- s
- p <expression>

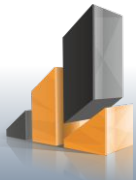
Удаленная отладка

- target remote <ip:port>
- monitor reset

Emacs имеет встроенный интерфейс к gdb:

- M-x gdb

Для vim: http://cgdb.github.io/

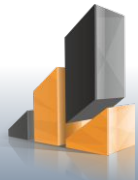
## Теги

Специальная программа разбирает исходные тексты и сохраняет информацию о найденных функциях, переменных и т.д. и их расположении.

Затем текстовой редактор, используя результат работы этой программы, позволяет перейти к реализации функции по её имени.

Пример построения тегов в Makefile:

```
tags:
	(find . -name \*.h -print; find . -name \*.c -print; find . -name \*.cpp -print) | grep -v ".*~" |grep -v distr|grep -v '\.\/sdk' | grep -v build| grep -v '^\.\#.*' | (ctags --version | grep "Exuberant Ctags" && ctags -e -L - || etags -)
```

## X11

X11 сервер (Xorg) включает нужный видеорежим и ждёт запросов на сокетах

- unix:/tmp/.X11-unix/X0
- 6000/tcp

- unix:/tmp/.X11-unix/X1
- 6001/tcp

Графическое приложение отправляет ему запросы на отрисовку элементов и получает ответы и события.

Локально для ускорения используется разделяемая память.

Xorg не занимается оформлением окна и не управляет его положением и размерами. Для этого существует специальная программа – оконный менеджер, которая также общается с X11-сервером (например icewm)