1. **Краткий экскурс в историю**

С 1969 по 1973 перед инженерами Bell labs стояла задача разработать переносимую операционную систему, ей стала всеми известная UNIX. Прародителем языка C был язык B, на котором была написана первая версия UNIX. В последствии был представлен язык C, который стал стандартом на многие годы. Главными разработчиками C являются Брайан Керниган и Деннис Ритчи.

Язык C++ появился в 1980 годы в качестве некоторой надстройка над C. Один из его основных разработчиков Бьерн Страуструп назвал его “C with classes”, что полностью отражает его сущность.

1. **Немного про боль (windows cmd)**

Для многих работа с командной строкой windows является не очень удобным, но это по усмотрению каждого. С появление windows 10 microsoft добавила такую фичу, как subsystem подробнее (как установить и пользоваться) можно почитать тут – <https://losst.ru/ustanovka-bash-v-windows-10>

Аналоги всех bash команд можно посмотреть тут – <https://white55.ru/cmd-sh.html>

Альтернативным и приемлемым решениям является использование windows powershell, про особенности которой можно почитать тут – https://tech-geek.ru/analogues-of-bash-tools-in-powershell/

1. **Первая программа, состоящая из нескольких файлов, на C**

Установка компилятора – <https://sourceforge.net/projects/mingw/>

Существует несколько стандартов языка C. Последним из них является  [ISO / IEC 9899: 2011](http://en.wikipedia.org/wiki/C11_(C_standard_revision)) известный также как C11. Стандарты между собой отличаются поддержкой различных нововведений, кусаемых языка.

Проверка работоспособности компилятора: gсс –-version

! Не забыть прописать в переменную среды PATH !

MINGW gcc – семейство компиляторов (программ преобразующих код в понятные для машины интсрукции). В его состав также входят: as — ассемблер и ld — линкер.

Исходный файл это всего лишь код, его следует скомпилировать в исполняемы файл или статическую/динамическую библиотеку.

Этапы компиляции:

1. Препроцессинг (преобразование команды для дальнейшего компилирования, например, замена всех директив начинающихся с #: 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.);

gсс -E somefile.c -o somefile.ii

1. Компиляция (преобразование кода в ассемблерный, это промежуточный шаг между высокоуровневым языком и машинным (бинарным) кодом.);

gсс -S somefile.c -o somefile.s

1. Ассемблирование (преобразование ассемблерного кода в машинный (бинарный), при этом мы получаем так называемы объектный файл (кусок машинного кода), который в последствии может быть использован для создания исполняемого файла или файла библиотеки (что бы проанализировать этот файл можно использовать утилиту nm: nm -f sysv))

as somefile.s -o somefile.o (в Windows .obj)

1. Компоновка или линковка (связь всех объектных файлов и файлов библиотек в один исполняемый файл с помощью таблицы символов)

gсс somefile.o -o somefile

Напишем простенькую программу (Hello, world!), состоящую из нескольких файлов, и посмотрим на каждый этап компиляции в отдельности. Весь код лежит на гите – <https://github.com/qqunity/qproject>

Хорошим тоном является отделение объявления от реализации (declaration и definition). Связано это с раздельной компиляцией, и ошибки по типу различных вызовов одной и той же функции будут выявлены только на этапе линковки. Реализовать можно с помощь создания прототипа, но лучше это объявить в заголовочном файле \*.h . Но возникает проблема циклического включения. Решением является директива препроцессору

#ifndef SOME

#define SOME

…

#endif

Программа скомпилирована на linux не запустится под windows. Под windows это решается кросс-компиляторами или же установкой subsystem. Под linux же существует mingw-w64. И процесс компиляции будет проходить так:

x86\_64-w64-mingw32-gcc somefile.c -o somefile (для компиляции под 64-рязрядную систему)

i686-w64-mingw32-gcc somefile.c -o somefile (для компиляции под 32-рязрядную систему)

Но при изменении одного файла «пересобрать» (выполнять все этапы компиляции) все – это очень затратно, ведь достаточно пересобрать только изменённый файл и запустить процесс линковки. Для таких целей существует makefile и утилита make. Синтаксис утилиты make такой:

цель: зависимости

[tab] команда

Запуск утилиты на linux происходить следующим образом: make -f file [target]

А на windows:

mingw32-make -f file [target]

Переменны в makefile объявляются следующим образом:

[имя переменной]=[значение]

А вызов (подстановка значения) происходит так:

$([имя переменной])

Кто писал когда-нибудь bash скрипты найдут в этом что-то похожее. Напишем более универсальный makefile. Переменная $< зарезервирована в синтаксисе утилиты make, она обозначает цель этапа, а переменная $@ – зависимости (см. синтаксис выше). Также можно использовать шаблоны:

%.[расширение]: %.[расширение]

[tab] $(CMD) $(FLAGS) $< -o $@

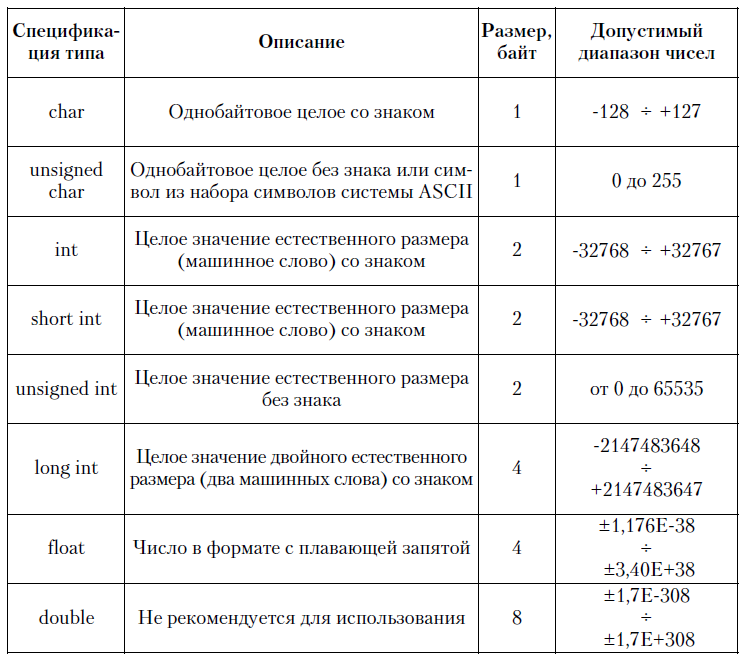
1. **Установка и настройка IDE**

Программировать в командной строке, конечно, круто, но не так удобной для этих целей лучше использовать IDE (интегрированную среду разработки). Мы будем пользоваться QTCreator. Установить IDE можно по ссылке – <https://www.qt.io/download-open-source>

Для того чтобы создать проект надо перейти в Файл–Создать файл или проект, далее выбрать Проект без Qt, Приложение на языке C. Далее необходимо выбрать название и расположение проекта, затем выбрать сборщик (можно оставить без изменения – qmake) и указать компилятор – MinGW. По завершении будет создан проект, в котором будет файл main.c содержащий “Hello, world”.

1. **Немного про типы данных**

Всё, что нужно знать о типах данных в C:



Приведение типов:

([тип к которому приводим])[исходный тип]

Статические массивы:

[тип элементов массива] [имя массива][[кол-во элементов]];

Перечисления:

enum [имя]{

[имя поля 1], //0

[имя поля 2], //1

[имя поля 3] = 0, //0

[имя поля 3], //1

[имя поля 4], //2

…

};

Структуры:

struct [имя]{

[тип поля 1] [имя поля 1];

[тип поля 2] [имя поля 2];

[тип поля 3] [имя поля 3];

…

};

Объединения:

union {

[тип поля 1] [имя поля 1];

[тип поля 2] [имя поля 2];

[тип поля 3] [имя поля 3];

…

}[имя];

Указатели:

[тип]\* [имя перемнно];

Разыменовывание указателя – \*ptr, взятие указателя – &elem.

Иногда удобно использовать инструмент объявления для того, чтобы сложному типу дать короткое название:

typedef [тип] [alias];

1. **Форматный ввод/вывод в C**

Библиотека, отвечающая за ввод/вывод в C: <stdio.h>. Ввод/вывод в C осуществляется с помощью следующих инструкций:

printf(“…”,…);

scanf(“…”, &…);

Где в двойных кавычках задаётся форматная строка, включающая сочетание символов %\*, отвечающее за тип выводимого объекта:

%d — вывести в виде десятичного целого числа;  
%o — печатать как восьмеричное целое;  
%u — печатать как беззнаковое десятичное целое;  
%x — печатать как шестнадцатеричное целое;  
%s — печатать как строку символов;  
%c — печатать как один символ;

После двойных кавычек через запятую перечисляются переменные или выражения, участвующие в форматной строке в порядке их появления.

1. **Операции и логические выражения в C**

*Арифметические операции:*

Основные бинарные операции, расположенные в порядке уменьшения приоритета:

* (\*) умножение;
* (/)  деление;
* (+)  сложение;
* (-) вычитание;
* (%) остаток от целочисленного деления.

Основные унарные операции:

* (++) инкрементирование (увеличение на 1);
* (--) декрементирование (уменьшение на 1);
* (-) изменение знака.

*Основные операции отношения:*

* (==) эквивалентно — проверка на равенство;
* (!=) не равно — проверка на неравенство;
* (<) меньше;
* (>) больше;
* (<=) меньше или равно;
* (>=) больше или равно.

*Побитовые операции:*

* (&) конъюнкция;
* (|) дизъюнкция;
* (~) инверсия;
* (^) исключающее или;
* (>>) побитовый сдвиг вправо;
* (<<) побитовый сдвиг влево.

*Условные логические операции:*

* (&&) И;
* (||) ИЛИ;
* (!) НЕ.

Многие бинарные операции можно комбинировать с оператором присваивания.

Всё, что не 0 – false.

1. **Структурное программирование на C**

Структурное программирование является парадигмой, которая вводит понятия шаблонов написания императивного кода (кода состоящего из последовательных инструкций). Данные шаблоны включают в себя следуюие конструкции:

1. Ветвление
2. Оператор выбора

if ([условие]) {

…

}

else if ([условие]) {

…

}

else {

…

}

1. Оператор множественного выбора

switch ([переменная, по которой идёт выбор]){

case [1 вариант выбора]:

…

case [2 вариант выбора]:

…

default:

…

}

1. Циклы
2. Цикл for

for ([инициализация]; [условие]; [выражение]) {

…

};

1. Цикл while (с предусловие)

[инициализация]

while([условие]) {

…

[выражение];

}

1. Цикл do while (с пост условием)

[инициализация]

do {

…

[выражение];

} while([условие]);

1. **Процедурное программирование в C**

Это парадигма, которая подразумевает возможность создания локальных переменных, функций и процедур. Прототипом функции является её объявление, т.е. указание возвращаемого типа и списка аргументов:

[тип возвращаемого значения] [имя функции]([тип 1 аргумента] [имя 1 аргумента],…){

…

return …;

}

Также функция может не возвращать ничего, тогда для этого случая предусмотрен тип void (подробнее о нём можно узнать в теме указатели).