Памятка по ЭВМ для первокурсников

Автор: Алексей Коляскин
Вычитка: Анна Попова

5 сентября 2021 г.

1 Настройка окружения

1.1 Установка приложения Ubuntu под Windows

По-хорошему стоит установить себе Linux, например в качестве второй ОС, и не париться. Если всё же хочется писать код под виндой, то делаем следующее.

Запускаем Microsoft Store. В поиске вводим Ubuntu. В найденном выбираем "Ubuntu". Жмем получить, установить.

В меню пуск Windows ищем "Панель управления". Там выбираем "Программы и компоненты". В новом окне слева выбираем "Включение и отключение компонентов WIndows". Активируем галочку "Подсистема WIndows для Linux жмем "OK". Перезагружаем компьютер.

После перезагрузки в поиске меню пуск ищем Ubuntu и запускаем. В новом окошке откроется терминал Ubuntu, в котором начнется установка окружения. Вводим имя будущего пользователя, потом дважды вводим пароль. Установка приложения Ubuntu завершена. Том с Windows примонтирован в директорию /mnt/

1.2 Основные команды терминала

cd – перейти в домашнюю директорию
cd <path> – перейти в директорию, которая находится по пути «path»
cd .. – подняться на директорию выше
ls – вывести содержимое текущей директории
ls -la – вывести содержимое текущей директории списком вместе со скрытыми файлами
clear – очистить окно терминала от всех сообщений
man <command> – выводит справку по использованию команды «command»
cat <filename> – выводит содержимое файла «filename» в терминал
rm <filename> – удалить файл «filename»
rm -r <dir> – удалить файл или директорию по имени «dir»
cp <source> <dest> – копировать файл «source» в файл или директорию «dest»
mv <source> <dest> – переместить файл «source». Если dest – имя директории, то файл «source» будет перемещен в неё. Иначе файл «source» будет переименован в файл «dest»
pwd – написать текущую директорию
grep – команда с кучей параметров, которая используется для поиска по текстовым файлам, выводу консоли и т.д. Советую выполнить man grep и почитать, как ей пользоваться.

Если есть сомнения по поводу использования той или иной команды, то стоит сначала запустить с параметром −−help(«имя команды» −−help) или сделать man «имя команды».
Подробнее про команды терминала можно почитать например на Хабре – `https://habr.com/ru/post/481398/`

В терминале работает автодополнение по нажатию TAB, просто начните вводить команду и нажмите TAB. Двойное нажатие TAB выведет список всех команд, которые начинаются на то, что вы ввели. Копирование и вставка в терминале работает по Ctrl+Shift+C и Ctrl+Shift+V или по Ctrl+Insert и Shift+Insert.
При написании путей можно использовать следующие алиасы:
./ – текущая директория
../ – директория на уровень выше
∼ – домашняя директория

1.2.1 Примеры

Пусть вы находитесь в папке /home/109/homework, где 109 – имя пользователя.
Путь ./12345 раскроется в /home/109/homework/12345
Путь ../ раскроется в /home/109/
Путь ∼/homework/ раскроется в /home/109/homework/

1.3 Установка нужных программ под Windows и под Linux

Команды установки приложений и библиотек для Linux и для приложения Ubuntu под Windows идентичны.
Устанавливаем компиляторы и отладчик.

```
sudo apt install g++ gdb
```

Вы скорее всего увидите, как Кирилл Юрьевич открывает файловый менеджер с синим интерфейсом в терминале. Этот файловый менеджер называется Midnight Commander. Команда установки:

```
sudo apt install mc
```

Запускается вызовом команды mc в терминале.

1.4 Среда разработки

Код можно писать и в блокноте, но это неудобно. Обычно для разработки приложений используют интегрированную среду разработки(Integrated development environment — IDE). IDE включает в себя несколько важных компонент: текстовый редактор, обычно с автодополнением и подсветкой синтаксиса и средства работы с отладчиком, компиляторами.

1.4.1 Windows

Под Windows имеет смысл установить Visual Studio Community или qtcreator.

1.4.2 Linux

Под Linux стоит поставить себе qtcreator.

```
sudo apt install qt5-default qtcreator
```

2 Сборка кода

2.1 Основы

Собирать программы из файлов исходного кода вы будете с помощью компиляторов GCC.
Пусть program_executable - имя исполняемого файла, который вы хотите получить.
Сборка программы из одного файла program_name.cpp с исходным кодом:

```
g++ program_name.cpp -o program_executable
```

Сборка программы из нескольких файлов исходного кода:

```
g++ -c *.cpp
g++ *.o -o program_executable
```

2.2 Флаги сборки

При сборке кода обязательно нужно указывать дополнительные флаги компилятора, собирая без них вы рискуете тем, что код, который собирался у вас локально, не будет собираться у Кирилла Юрьевича. Минимальный набор флагов:
-mfpmath=sse -fstack-protector-all -W -Wall -Wextra -Wunused -Wcast-align -Werror -pedantic -pedantic-errors -Wfloat-equal -Wpointer-arith -Wformat-security -Wmissing-format-attribute -Wformat=1 -Wwrite-strings -Wcast-align -Wno-long-long -Woverloaded-virtual -Wnon-virtual-dtor -Wcast-qual -Wno-suggest-attribute=format
Собирать нужно вот так:

g++ -mfpmath=sse -fstack-protector-all -W -Wall -Wextra -Wunused -Wcast-align -Werror -pedantic -pedantic-errors -Wfloat-equal -Wpointer-arith -Wformat-security -Wmissing-format-attribute -Wformat=1 -Wwrite-strings -Wcast-align -Wno-long-long -Woverloaded-virtual -Wnon-virtual-dtor -Wcast-qual -Wno-suggest-attribute=format program_name.cpp -o program_executable

2.3 Запуск программы

Собранная программа с именем a.out запускается в терминале следующим образом:

```
./a.out <args>
```

Здесь <args> – аргументы командной строки. Они будут переданы в int main(int argc, char **argv) внутри аргумента argv. Важно помнить, что argc всегда не меньше единицы, а argv[0] обычно(это никак не определено в стандарте) путь к исполняемому файлу.

3 Отладка

Не всегда написанный код сразу правильно отрабатывает на всех тестовых вариантах. Ваша программа может упасть или дать неверный ответ. Но даже если вы получили верный ответ, это не значит, что у Кирилла Юрьевича программа отработает точно так же. Поэтому программу необходимо отладить. Существует несколько вариантов отладки, причем многие из них можно использовать совместно.

3.1 Исключения(Exceptions)

По умолчанию в собранной программе допускаются «прекрасные» ситуации в виде деления на ноль, операций с неинициализированными переменными, переполнения и т.д. Такая программа будет нормально отрабатывать у вас, но когда Кирилл Юрьевич соберет и запустит её у себя, то она упадает со словами "Floating point exception". Чтобы отловить такие ситуации, нужно включить обработку исключений у себя.

В файл с исходным кодом, содержащий функцию int main, добавляем

```
#include <fenv.h>
```

В самом начале функции int main пишем следующее

```
feenableexcept (FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW | FE_INVALID);
```

3.2 Отладка печатью

Простейшим способом отладки является отладка с помощью printf. В нужном месте кода добавляется вызов функции printf, которая выводит необходимую информацию. Этого способа может хватить на первые два курса, но есть более оптимальные способы отладки, которые требуют меньше времени, например gdb.

3.3 Отладчик(дебаггер) GDB

gdb – программа для отладки других программ. Позволяет пошагово проходить по вашему коду, выводить значения переменных во время исполнения, узнать место, в котором программа упала и т.д. Чтобы нормально использовать gdb нужно собрать вашу программу с дебажной информацией. Для этого потребуется к вашим флагам сборки добавить еще два флага: -O0 -g
Отладчик запускается следующей командой

```
gdb ./program_executable
```

Чтобы запустить исполнение кода в консоли gdb пишем

```
r argument_list
```

Здесь argument_list – аргументы запуска вашего исполняемого файла.

3.3.1 Основные команды gdb

`r argument_list`

Запуск исполняемого файла с аргументами argument_list

`br program_name.cpp:line`

Поставить точку остановки(breakpoint) на строчку номер line в файле program_name.cpp

`c`

Продолжить исполнение программы

`li`

Показать строчки кода рядом с точкой остановки

`s`

Перейти к следующей строчке(инструкции)

3.4 Санитайзеры

Санитайзеры – инструменты для отлавливания ошибок в коде. Есть несколько видов санитайзеров

- Address Sanitizer(asan) – инструмент для поиска ошибок в работе с памятью. Обнаруживает выход за границы массива, утечки и другие ошибки памяти.
- Thread Sanitizer(tsan) – инструмент для обнаружения дата рейсов в многопоточных программах. Не понадобится вам до третьего курса.
- Undefined Behaviour Sanitizer(ubsan) – инструмент для обнаружения неопределенного поведения. Обнаруживает разыменовывание нулевых указателей, ошибки при приведении типов, выход за границы статических массивов и т.д.

Чтобы собрать вашу программу с санитайзером, нужно добавить один из следующих флагов компиляции.

```
-fsanitize=address
-fsanitize=thread
-fsanitize=undefined
```

Не стоит добавлять эти флаги одновременно. Ходят слухи, что совместно они работают некорректно. Подробнее про asan можно почитать здесь – `https://clang.llvm.org/docs/AddressSanitizer.html`, про tsan здесь – `https://clang.llvm.org/docs/ThreadSanitizer.html`, а про ubsan здесь – `https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html`.

4 Статический анализ кода

Это дополнительная глава, по факту вы и без нее можете закрыть все задачи Кирилла Юрьевича, но она может помочь оптимизировать написание вами кода.

Если не получается сходу найти в коде ошибку или просто хочется лишний раз удостовериться, что он написан без ошибок, то можно попытать счастья со статическими анализаторами кода. Статический анализатор кода – это утилита, которая получает себе на вход исходный код или файл с правилами сборки, а потом проверяет исходники на наличие ошибок. Статические анализаторы помогают быстро исправить простые ошибки, типа использования памяти после вызова delete, разыменовывания нулевого указателя и т.д. Таких анализаторов немного, открытых-бесплатных еще меньше, а тех, что действительно нормально работают вообще кот наплакал.

Достойны отдельного упоминания Cppcheck и Clang Static Analyzer.

4.1 Clang Static Analyzer

Если используете qtcreator, то в нем по умолчанию включен плагин Clang Code Model. Его можно донастроить на большее число проверок, добавив флаги в настройках. Для этого нужно зайти в Tools->Options->C++->Code Model и в Clang Code Model поменять Diagnostic Configuration, например выбрать там Check for questionable constructions.

4.2 Cppcheck

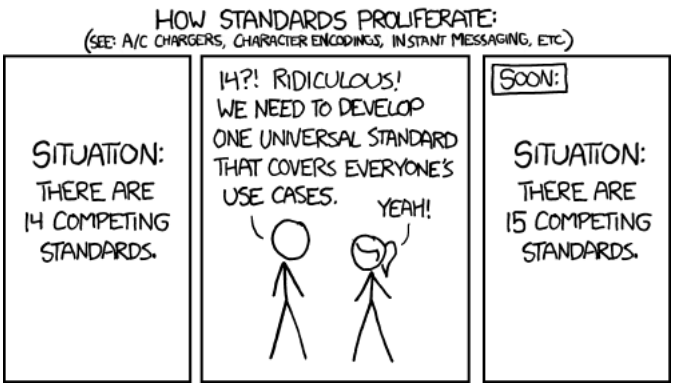
Если у вас Linux или приложение Ubuntu, то можно просто написать в терминале.

```
sudo apt install cppcheck
```

Если не используете qtcreator, то можно почитать на сайте мануал по использованию - `https://cppcheck.sourceforge.io/manual.pdf`

Если вы используете qtcreator, то все гораздо проще. Заходим в Help->About Plugins и включаем там cppcheck(experimental). Перезапускаем qtcreator, заходим в Tools->Options->Analyzer->Cppcheck и проставляем там нужные настройки.

5 О форматировании кода



Куда же без мемасов?

Это глава для тех, кто хочет в будущем заниматься профессиональным программированием или просто писать красивый код. По факту Кирилл Юрьевич смотрит на форматирование кода только на третьем курсе и только у тех, кто у него под научным руководством.

Существует много разных стандартов форматирования кода. По факту можно конечно писать, как душе угодно, лишь бы самому понятно было, и никто вам по этому поводу ни слова не скажет. Но если вы хотите писать хороший код, то стоит придерживаться какого-то единого стиля. Одним из таких стилей является GNU Coding Standards, его придерживается в том числе Кирилл Юрьевич. Сам по себе этот стандарт довольно большой, если захотите его использовать, то вам понадобится лишь кусок оттуда про C/C++. Ссылка на стандарт – `https://www.gnu.org/prep/standards/standards.pdf` и на соглашения по стилю для C/C++ – `https://gcc.gnu.org/codingconventions.html`