**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Программирование»**

**Тема: Сборка программ в C**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 | | Федоров И.А |
| Преподаватель |  | Круглик А. Д. |

Санкт-Петербург

2018

**Цель работы.**

Ознакомиться с работой препроцессора и научиться собирать программы

при помощи Make-файлов

**Основные теоретические положения.**

**Препроцессор** -это программа,которая подготавливает код программы дляпередачи ее компилятору.

Команды препроцессора называются директивами и имеют следующий формат:

**#ключевое\_слово параметры**

Основные действия, выполняемые препроцессором:

* Удаление комментариев
* Включение содержимого файлов (*#include*)
* Макроподстановка (*#define*)
* Условная компиляция (*#if, #ifdef, #elif, #else, #endif*)

**#include**

* Препроцессор обрабатывает содержимое указанного файла и включает содержимое на место директивы. Включаемые таким образом файлы называются заголовочными и обычно содержат объявления функций, глобальных переменных, определения типов данных и другое.
* Директива может иметь вид #include "...." либо #include <...>. Для <...> поиск файла осуществляется среди файлов стандартной библиотеки, а для "..." - в текущей директории.

**#define**

Позволяет определить макросы или макроопределения. Имена их принято писать в верхнем регистре через нижние подчеркивания, если это требуется:

#define SIZE 10

Такое макроопределение приведет к тому, что везде, где в коде будет использовано SIZE, на этапе работы препроцессора это значение будет заменено на 10. Макросы отличаются только наличием параметров:

#define MUL\_2(x) x\*2

2

Таким образом, каждый макрос MUL\_2 в коде будет преобразован в выражение x\*2, где x - его аргумент. Следует обратить особое внимание, что define выполняет просто подстановку идентификатора (без каких-то дополнительных преобразований), что иногда может приводить к ошибкам, которые трудно найти.

**#if, #ifdef, #elif, #else, #endif**

Директивы условной компиляции допускают возможность выборочной компиляции кода. Это может быть использовано для настройки кода под определенную платформу, внедрения отладочного кода или проверки на повторное включение файла.

**Компиляция** -процесс преобразования программы с исходного языка высокогоуровня в эквивалентную программу на языке более низкого уровня (в частности, машинном языке).

**Компилятор** -программа,которая осуществляет компиляцию.



Большая часть компиляторов преобразует программу в машинный код, который может быть выполнен непосредственно процессором. Этот код различается между операционными системами и архитектурами. Однако, в некоторых языках программирования программы преобразуются не в машинный, а в код на более низкоуровневом языке, но подлежащий дальнейшей интерпретации (байт-код). Это позволяет избавиться от архитектурной зависимости, но влечет за собой некоторые потери в производительности.

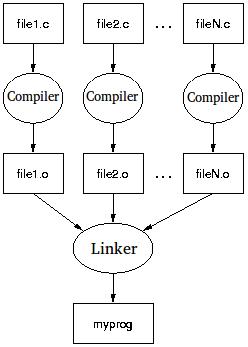
Компилятор языка C принимает исходный текст программы, а результатом является объектный модуль. Он содержит в себе подготовленный код, который может быть объединён с другими объектными модулями при помощи линковщика для получения готового исполняемого модуля.

**Линковка**

Мы уже знаем, что можно скомпилировать каждый исходный файл по отдельности и получить для каждого из них объектный файл. Теперь нам надо получить по ним исполняемый файл. Эту задачу решает линковщик (компоновщик) - он принимает на вход один или несколько объектных файлов и собирает по ним исполняемый модуль.

3

Работа компоновщика заключается в том, чтобы в каждом модуле определить и связать ссылки на неопределённые имена.



**Make-файлы**

Сборка проекта - это процесс получения **исполняемого файла** из **исходного кода**.

Сборка проекта вручную может стать довольно утомительным занятием, особенно, если исходных файлов больше одного и требуется задавать некоторые параметры компиляции/линковки. Для этого используются **Makefile** - список инструкций для утилиты **make**, которая позволяет собирать проект сразу целиком.

Если запустить утилиту

make

то она попытается найти файл с именем **Makefile** в текущей директории и выполнить из него инструкции.

Если требуется задать какой-то конкретный **Makefile**, это можно сделать с помощью ключа **-f**

make -f AnyMakefile

Любой make-файл состоит из

* списка **целей**
* **зависимостей** этих целей
* **команд**,которые требуется выполнить,чтобы достичь эту цель

цель: зависимости

4

[tab] команда

Для сборки проекта обычно используется цель all, которая находится самой первой и является целью по умолчанию. (фактически, первая цель в файле и является целью по-умолчанию)

Также, рекомендуется создание цели clean, которая используется для очистки всех результатов сборки проекта

Использование нескольких целей и их зависимостей особенно полезно в больших проектах, так как при изменении одного файла не потребуется пересобирать весь проект целиком. Достаточно пересобрать измененную часть

Часто бывает необходимо изменить какие-то параметры сборки. Это может стать проблемой, если придется все изменять вручную. Что бы избежать этого, полезно использовать переменные. Для этого достаточно присвоить им значения до момента их использования и в месте использования обратиться к ним как $(VARNAME). Имена переменных принято писать в верхнем регистре.

**ПОСТАНОВКА ЗАДАЧИ:**

* В текущей директории создайте проект с make-файлом. Главная цель должна приводить к сборке проекта. Файл, который **реализует главную функцию**, должен называться menu.c; **исполняемый файл** - menu. Определение каждой функции должно быть расположено в **отдельном файле,** название файлов указано в скобках около описания каждой функции.
* Реализуйте функцию-меню, на вход которой подается одно из **значений** 0, 1, 2, 3 и **массив** целых чисел **размера не больше** 100. Числа разделены пробелами. Строка заканчивается символом перевода строки.
* В зависимости от **значения**, функция должна выводить следующее:

0 : максимальное по модулю число в массиве. (abs\_max.c)

1 : минимальное по модулю число в массиве. (abs\_min.c)

2 : разницу между максимальным по модулю и минимальным по модулю элементом. (diff.c)

3 : сумму элементов массива, расположенных после максимального по модулю элемента (включая этот элемент). (sum.c)

5

иначе необходимо вывести строку "Данные некорректны".

*Ошибкой в данном задании считается дублирование кода!*

*Подсказка: функция нахождения модуля числа находится в заголовочном файле stdlib.h стандартной библиотеки языка Си.*

*При выводе результата, не забудьте символ переноса строки*

**Экспериментальные результаты.**

**Проверка :**

|  |  |
| --- | --- |
| **Ввод** | 0 -8 -23 -30 -11 -28 15 -20 -24 |
| **Вывод** | **0** |

|  |  |
| --- | --- |
| **Ввод** | 1 3 5 6 7 8 11 32 31 -2 1 42 -6 21 |
| **Вывод** | **13** |

|  |  |
| --- | --- |
| **Ввод** | 2 7 4 9 14 51 -7 8 -3 14 -3 94 |
| **Вывод** | **119** |

|  |  |
| --- | --- |
| **Ввод** | 3 32 31 -2 1 42 -6 21 14 51 -7 |
| **Вывод** | **10** |

|  |  |
| --- | --- |
| **Ввод** | 4 14 51 -7 14 51 -7 14 51 -7 14 51 -7 |
| **Вывод** | **Не корректные данные!** |

**Выводы.**

* ходе данной работы ознакомился с работой препроцессора, получил необходимые знания для сборки программ при помощи Make-файлов.