Семинар #1: Компиляции. Домашнее задание.

В некоторых задачах этого домашнего задания необходимо будет создавать скрипты командной строки. Такие скрипты могут содержать несколько команд командной строки. При исполнении такого скрипта будут исполняться все эти команды. Если в задании говорится, что нужно создать скрипт, то подразумевается следующее:

- Если вы используете Windows:
  Нужно будет создать batch-скрипт – это файл с расширением `.bat`. Этот файл должен содержать все команды, которые обычно вводятся в командную строку. Запустить этот скрипт можно просто напечатав его имя в командной строке. Например, если файл называется `dog.bat`, то для его запуска нужно перейти в командной строке в папку, где этот файл находится, и написать следующее:

  ```
  dog.bat
  ```

- Если вы используете Linux или macOS:
  Нужно будет создать bash-скрипт – это файл с расширением `.sh`, первая строка которого имеет вид:

  ```
  #!/bin/bash
  ```

  После этого следуют все команды, которые обычно вводятся в командную строку. Если файл называется `dog.sh`, то для его запуска нужно перейти в командной строке в папку, где этот файл находится, и написать следующее:

  ```
  chmod +x ./dog.sh        # изменяем права доступа, нужно выполнить только 1 раз
  ./dog.sh                 # запускаем файл
  ```

Все материалы к этому заданию находятся в папке `seminar1_libs/homework/code`
!! Делайте каждую задачу в отдельной папке

Задача 1. Раздельная компиляция: функции и структуры

В папке `homework/code/01sep_dynarray` лежит файл `dynarray.c` в котором находится простейшая реализация динамического массива, написанная на языке C. Вам нужно разделить этот файл на три части:

- Файл `dynarray.h` – должен содержать все объявления, относящиеся к динамическому массиву.
- Файл `dynarray.c` – должен содержать определения всех функций, относящихся к динамическому массиву.
- Файл `main.c` – должен содержать только функцию `main`.

Убедитесь, что программа состоящая из этих трёх файлов компилируется:

```
gcc main.c dynarray.c
```

Задача 2. Раздельная компиляция: класс

В папке `02sep_time` лежит файл `time.cpp` в котором находится класс `Time`. Вам нужно разделить этот файл на три части:

- Файл `time.hpp` – должен содержать все объявления, относящиеся к классу `Time`.
- Файл `time.cpp` – должен содержать определения всех функций, относящихся к классу `Time`.
- Файл `main.cpp` – должен содержать только функцию `main`.

Задача 3. Раздельная компиляция: шаблоны

В папке `03sep_template` лежит файл `tmath.cpp` в котором находится шаблонные функции `sum` и `mult`. Предполагается, что эти функции будут работать только с типами `int`, `unsigned int`, `float` и `double`. Вам нужно разделить этот файл на три части:

- Файл `tmath.hpp` – должен содержать объявления шаблонных функции.
- Файл `tmath.cpp` – должен содержать определения шаблонных функций.
- Файл `main.cpp` – должен содержать только функцию `main`.

Задача 4. Этапы компиляции

В папке `04stages` лежит файл `hello.c`. Вам нужно написать скрипт, который бы создавал промежуточные файлы со всех этапов компиляции для этого файла. После запуска скрипта должны создаваться:

1. Файл `hello.i` – код после этапа препроцессинга.
2. Файл `hello.s` – код ассемблера после этапа компиляции.
3. Файл `hello.o` – машинный код после этапа ассемблирования.
4. Исполняемый файл (`hello.exe` на Windows или просто `hello` на Linux)

Задача 5. Definitions

В папке `05definitions` лежит программа, состоящая из пяти файлов. В файле `alice.c` содержится функция `cat`, которая печатает на экран либо строку `ALICE:CAT` либо строку `alice:cat` в зависимости от того, был ли определён макрос `UPPERCASE`. В файле `bob.c` содержится аналогичная функция `dog`. В файле `main.c` просто вызываются функции `cat` и `dog`. Вам нужно написать скрипт, который бы компилировал эту программу таким образом, чтобы на экран выводились строки:

```
ALICE:CAT
bob:dog
```

Изменять сами файлы программы нельзя.

Задача 6. Функции с одинаковым названием

В папке `06function_same_name` лежит программа, состоящая из пяти файлов. В файле `alice.c` содержатся две функции: `cat` и `hello`, причём функция `cat` вызывает функцию `hello`. В файле `bob.c` содержатся две функции: `dog` и `hello`, причём функция `dog` вызывает функцию `hello`. В файле `main.c` просто вызываются функции `cat` и `dog`. Но данная программа не компилируется из-за ошибки:

```
multiple definition of `hello'
```

Исправьте данную программу, не меняя имена функций.

Задача 7. Общая переменная

В папке `07common_variable` лежит программа, состоящая из пяти файлов. В файле `alice.c` содержится глобальная переменная `value` и функция `cat`, которая изменяет эту глобальную переменную. В файле `bob.c` также содержится глобальная переменная `value` и функция `dog`, которая изменяет эту глобальную переменную. В файле `main.c` просто вызываются функции `cat` и `dog` несколько раз и печатается значение переменной `value`.

Предполагалось, что переменная `value` будет одна на всю программу и функции `cat`, `dog` и `main` должны работать с одной и той же глобальной переменной. Однако, при попытке компиляции данной программы происходит ошибка:

```
multiple definition of `value'
```

Исправьте эту программу так, чтобы функции `cat`, `dog` и `main` работали с одной и той же глобальной переменной.

Задача 8. Создание статической библиотеки

В папке `08create_static` лежит программа, состоящая из пяти файлов. Вам нужно будет написать скрипт, который бы создавал статическую библиотеку под названием `house`. Библиотека должна будет содержать скомпилированный код из файлов `alice.c` и `bob.c`. Название библиотеки должно иметь вид:

- `libhouse.a` – если вы компилируете на Linux, macOS или если вы используете компилятор MinGW на Windows.
- `house.lib` – если вы компилируете на Windows, используя компилятор MSVC.

Задача 9. Подключение статической библиотеки

В этой задаче нужно будет подключить статическую библиотеку из предыдущей задачи. В папке `09link_static` лежит заготовка проекта. Вам нужно скопировать эту папку себе и добавьте файл статической библиотеки в подпапку `external/lib`. Структура проекта в папке `09link_static` должна иметь вид:

```
09link_static
|--main.c
|--external
   |--include
      |--alice.h
      |--bob.h
   |--lib
      |--libhouse.a   (или другое имя, в зависимости от системы)
```

В этой задаче вам нужно будет написать скрипт, который бы компилировал программу, подключая статическую библиотеку к исходному файлу `main.c`.

Задача 10. Создание динамической библиотеки

В папке `10create_dynamic` лежит программа, состоящая из пяти файлов. Вам нужно будет написать скрипт, который бы создавал динамическую библиотеку под названием `house`. Библиотека должна будет содержать скомпилированный код из файлов `alice.c` и `bob.c`. Название библиотеки должно иметь вид:

- `libhouse.so` – если вы компилируете на Linux.
- `libhouse.dylib` – если вы компилируете на macOS.
- `house.dll` – если вы компилируете на Windows (MSVC или MinGW).

Задача 11. Подключение динамической библиотеки

В этой задаче нужно будет подключить динамическую библиотеку из предыдущей задачи. В папке `11link_dynamic` лежит заготовка проекта. Вам нужно скопировать эту папку себе и добавьте файл динамической библиотеки в подпапку `external/lib`. Структура проекта в папке `11link_dynamic` должна иметь вид:

```
11link_dynamic
|--main.c
|--external
   |--include
      |--alice.h
      |--bob.h
   |--lib
      |--libhouse.so   (или другое имя, в зависимости от системы)
```

В этой задаче вам нужно будет написать 2 скрипта. Первый скрипт должен компилировать программу, подключая динамическую библиотеку к исходному файлу `main.c`. Второй скрипт должен запускать скомпилированный исполняемый файл.

Задача 12. Подключение динамической библиотеки во время выполнения

В предыдущей задаче динамическая библиотека подключалась во время запуска программы, но динамическую библиотеку можно подключить и в произвольный момент во время выполнения. Чтобы это сделать, нужно воспользоваться специальными библиотеками, представляемыми операционной системой.

- Если вы используете Linux или macOS, то вам нужно будет использовать библиотеку `dlfcn.h`.
- Если вы используете Windows, то вам нужно будет использовать функции `LoadLibrary`, `GetProcAddress` и `FreeLibrary` из библиотеку `windows.h`.

Скопируйте папку `12link_dynamic_runtime` и добавьте в подпапку `external/lib` файл динамической библиотеке, созданный в задаче 10. После этого изменить файл `main.c` так, чтобы он подключал динамическую библиотеку во время выполнения.

Задача 13. Общая функция

В папке `13common_function` лежит программа, написанная на C++, состоящая из шести файлов. В файле `hello.hpp` содержится функция `hello`. В файле `alice.cpp` содержатся функция `cat`, которая вызывает функцию `hello`. В файле `bob.cpp` содержатся функция `dog`, которая также вызывает функцию `hello`. В файле `main.cpp` просто вызываются функции `cat` и `dog`. Но данная программа не компилируется из-за ошибки:

```
multiple definition of `hello'
```

Исправьте данную программу, не меняя имена функций и не используя `static`.

Задача 14. Проект Image

В папке `14image` содержится простой проект, написанный на C++ (стандарт C++20). Проект содержит:

- Файл `main.cpp`, содержащий функцию `main`. Этот файл использует класс `Image`.
- Файлы `image.cpp` и `image.hpp`, в которых содержится код класса `Image`.
  Этот класс использует стороннюю библиотеку `stb`.
- Сторонняя библиотека `stb` также находится в нашем проекте (в папке `external/stb`).

Результатом данного проекта должны являться:

- Статическая библиотека, содержащая код класса `Image`.
- Исполняемый файл.

Напишите скрипт, который будет собирать проект (то есть создавать библиотеку и исполняемый файл). Менять код и структуру проекта нельзя.