**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

ФАИС

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

по дисциплине «**Операционные системы и среды**»

на тему: «**Знакомство со стандартной утилитой Gnu make для построения проектов в**

**ОС Unix/Linux**»

Выполнил: студент гр. ИП-32

Прокопенко А. Р.

Принял: преподаватель

Процкая М. А.

Дата сдачи отчета: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

**Цель**: ознакомиться с техникой компиляции программ на языке программирования C

(C++) в среде ОС семейства Unix/Linux, а также получить практические навыки использования

утилиты GNU make для сборки проекта.

**Задание**

1. Ознакомиться с теоретическим материалом.
2. Воспользоваться утилитой make для автоматизации сборки проектов из лабораторной работы №4 «Инструментальные средства ращзработки Linux».
3. Создать make-файл с высоким уровнем автоматизированной обработки исходных файлов программы согласно следующим условиям:
   * имя скомпилированной программы (выполняемый или бинарный файл), флаги компиляции и имена каталогов с исходными файлами и бинарными файлами (каталоги, src, bin, и т.п.) задаются с помощью переменных в makefile;
   * зависимости исходных файлов на языке C (C++) и цели в make-файле должны формироваться динамически;
   * наличие цели clean, удаляющей временные файлы;
   * каталог проекта должен бьыть структурирован следующим образом:

src – каталог с исходными файлами;

bin – каталог с бинарными файлами (скомпилированными);

makefile.

* структура данных (STL) - forward list;
* функция обработки структурированных данных — определение максимального среди отрицательных элементов;
* начальная сортировка — обменом;
* конечная сортировка — быстрая.

Все функции размещаются в отдельных файлах.

1. Выполнить программу (скомпилировать, при необходимости отладить) для первого варианта сортировки данных.
2. Изменить тип сортировки и выполнить программу.
3. Показать, что при изменении одного исходного файла и последующем вызове make будут

исполнены только необходимые команды компиляции (неизмененные файлы перекомпилированы не будут) и изменены атрибуты и/или размер объектных файлов (файлы с расширением o).

**Выполнение**

1. sudo apt get make
2. Переместим .cpp- и .h-файлы в каталог ~/lab5\_auto\_build/src и создадим каталог bin для скомпилированных .cpp-файлов.

Создадим makefile со следующим содержимым:

**makefile**:

loki: ./bin/main.o ./bin/calculate\_volume.o ./bin/calculate\_surface\_area.o ./bin/get\_radius.o

./bin/set\_radius.o ./bin/sphere.o

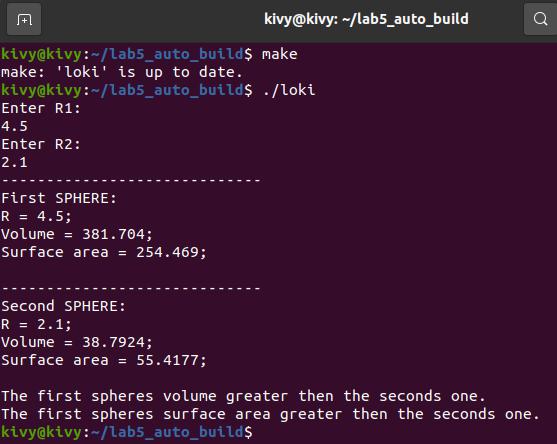
sudo g++ $^ -o $@

./bin/%.o: ./src/%.cpp

sudo g++ -c $^ -o $@

Далее просто запустим в терминале находясь в каталоге ~/lab5\_auto\_build команду make.

Получим файл loki, который можно запустить командой ./loki.



Аналогичным образом можно было бы поступить со вторым (более легким) проектом

из лабораторной работы №4.

1. **makefile**:

program\_name:= run

compiled\_files\_catalog:= ./bin/

sourced\_files\_catalog:= ./src/

flags:= -o

obj\_list:= $(compiled\_files\_catalog)main.o $(compiled\_files\_catalog)add\_item.o

$(compiled\_files\_catalog)change\_value\_by\_id.o $(compiled\_files\_catalog)get\_forwards\_lst\_size.o

$(compiled\_files\_catalog)get\_item\_by\_id.o $(compiled\_files\_catalog)hoara\_sort.o

$(compiled\_files\_catalog)my\_fun.o $(compiled\_files\_catalog)remove\_item\_by\_id.o

$(compiled\_files\_catalog)show\_flist.o $(compiled\_files\_catalog)sort\_by\_bubble.o

$(program\_name): $(obj\_list)

sudo g++ $^ $(flags) $@

$(compiled\_files\_catalog)%.o: $(sourced\_files\_catalog)%.cpp

sudo g++ -c $^ $(flags) $@

clean:

sudo rm $(program\_name)

sudo rm $(compiled\_files\_catalog)\*.o

**functionality.h**:

#pragma once

#include <forward\_list>

#include <list>

#include <algorithm>

#include <stdlib.h>

using namespace std;

void show\_flst(forward\_list<int> numbers);

int get\_forwards\_lst\_size(forward\_list<int> numbers);

int get\_item\_by\_id(forward\_list<int> numbers, int id);

forward\_list<int> change\_value\_by\_id(forward\_list<int> numbers, int id, int new\_value);

forward\_list<int> add\_item(forward\_list<int> numbers, int new\_item);

forward\_list<int> remove\_item\_by\_id(forward\_list<int> numbers, int id);

forward\_list<int> sort\_by\_bubble(forward\_list<int> numbers);

void hoara\_sort(forward\_list<int> &numbers, int left, int right);

void my\_fun(forward\_list<int> numbers);

**sort\_by\_bubble.cpp**:

#include "functionality.h"

forward\_list<int> sort\_by\_bubble(forward\_list<int> numbers)

{

int size = get\_forwards\_lst\_size(numbers);

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size - 1; j++)

{

int a = get\_item\_by\_id(numbers, j);

int b = get\_item\_by\_id(numbers, j + 1);

if (a > b)

{

numbers = change\_value\_by\_id(numbers, j, b);

numbers = change\_value\_by\_id(numbers, j + 1, a);

}

}

}

return numbers;

}

**remove\_item\_by\_id.cpp**:

#include "functionality.h"

forward\_list<int> remove\_item\_by\_id(forward\_list<int> numbers, int id)

{

int size = get\_forwards\_lst\_size(numbers);

if (id >= 0 && id < size)

{

auto current = numbers.begin();

id--;

while (id--)

current++;

numbers.erase\_after(current);

}

return numbers;

}

**add\_item.cpp**:

#include "functionality.h"

forward\_list<int> add\_item(forward\_list<int> numbers, int new\_item)

{

numbers.push\_front(new\_item);

return numbers;

}

**change\_value\_by\_id.cpp**:

#include "functionality.h"

forward\_list<int> change\_value\_by\_id(forward\_list<int> numbers, int id, int new\_value)

{

int size = get\_forwards\_lst\_size(numbers);

if (id >= 0 && id < size)

{

auto current = numbers.begin();

while (id--)

current++;

\*current = new\_value;

}

return numbers;

}

**get\_item\_by\_id.cpp**:

#include "functionality.h"

int get\_item\_by\_id(forward\_list<int> numbers, int id)

{

* it is considered that id is an integer between 0 and size - 1...

int size = get\_forwards\_lst\_size(numbers); if (id >= 0 && id < size)

{

auto current = numbers.begin();

while (id--)

current++;

return \*current;

}

return -1;

}

**get\_forwards\_lst\_size.cpp**:

#include "functionality.h"

int get\_forwards\_lst\_size(forward\_list<int> numbers)

{

return distance(numbers.begin(), numbers.end());

}

**hoara\_sort.cpp**:

#include "functionality.h"

void hoara\_sort(forward\_list<int> &numbers, int left, int right)

{

int pivot;

int l\_hold = left;

int r\_hold = right;

pivot = get\_item\_by\_id(numbers, left);

while(left < right)

{

while((get\_item\_by\_id(numbers, right) >= pivot) && (left < right))

right--;

if (left != right)

{

int a = get\_item\_by\_id(numbers, left);

int b = get\_item\_by\_id(numbers, right);

numbers = change\_value\_by\_id(numbers, left, b);

left++;

}

while((get\_item\_by\_id(numbers, left) <= pivot) && (left < right))

left++;

if (left != right)

{

int a = get\_item\_by\_id(numbers, left);

int b = get\_item\_by\_id(numbers, right);

numbers = change\_value\_by\_id(numbers, right, a);

right--;

}

}

numbers = change\_value\_by\_id(numbers, left, pivot);

pivot = left;

left = l\_hold;

right = r\_hold;

if(left < pivot)

hoara\_sort(numbers, left, pivot - 1);

if(right > pivot)

hoara\_sort(numbers, pivot + 1, right);

}

**my\_fun.cpp**:

#include "functionality.h"

#include <iostream>

void my\_fun(forward\_list<int> numbers)

{

list<int> negative\_elements;

for (int n : numbers)

if (n < 0)

negative\_elements.push\_back(n);

if (negative\_elements.empty())

cout << "No negative elements... " << endl;

else

{

std::list<int>::const\_iterator max\_negative =

max\_element(negative\_elements.begin(), negative\_elements.end());

cout << "Max negative is " << \*max\_negative << endl;

}

}

**show\_flist.cpp**:

#include "functionality.h"

#include <iostream>

void show\_flst(forward\_list<int> numbers)

{

cout << "My FORWARD LIST: " << endl;

auto current = numbers.begin();

auto end = numbers.end();

while (current != end)

{

std::cout << \*current << endl;

current++;

}

}

**main.cpp**:

#include <iostream>

#include "functionality.h"

int main()

{

forward\_list<int> numbers({1,100,1,6,5,4,3,2,1});

bool flag = true;

while (flag)

{

printf("\033c");

cout << "1. Show elements." << endl;

cout << "2. Add an element in front." << endl;

cout << "3. Remove an element by id." << endl;

cout << "4. Sort elements." << endl;

cout << "5. Use my finction." << endl;

cout << "6. Exit." << endl;

int choice;

cin >> choice;

switch (choice)

{

case 1:

show\_flst(numbers);

break;

case 2:

int elem;

cout << "Enter new value: ";

cin >> elem;

numbers = add\_item(numbers, elem);

break;

case 3:

int id;

cout << "Enter id: ";

cin >> id;

numbers = remove\_item\_by\_id(numbers, id);

break;

case 4:

//cout << "Sort by bubble ----> " << endl;

//numbers = sort\_by\_bubble(numbers);

cout << "Quick sort (Hoara) ----> " << endl;

hoara\_sort(numbers, 0, get\_forwards\_lst\_size(numbers) - 1);

break;

case 5:

my\_fun(numbers);

break;

case 6:

flag = false;

break;

default:

cout << "Wrong data. Try again." << endl;

break;

}

cout << "Press any key to continue..." << endl;

char key;

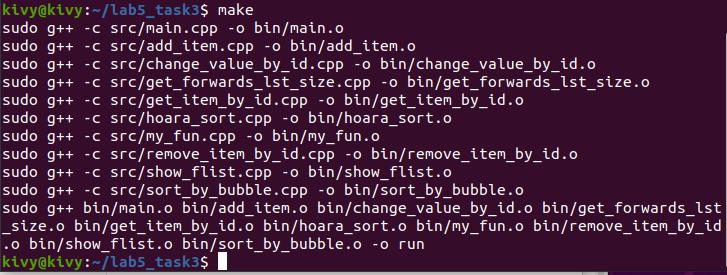
cin >> key;

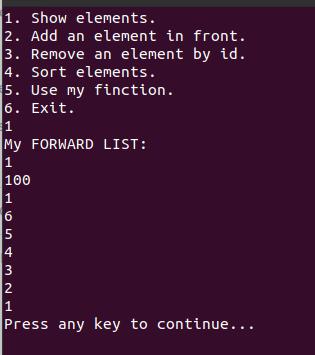
}

return 0;

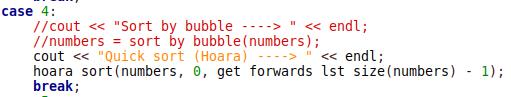
}

4. make

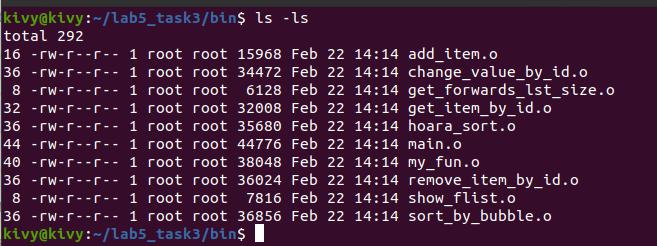




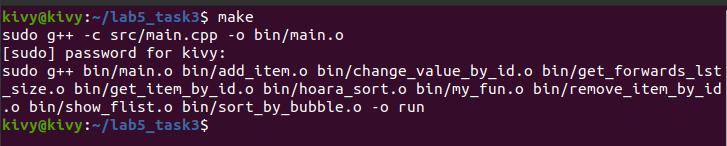
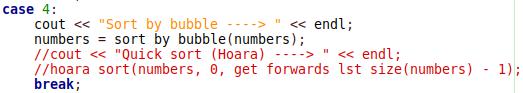
5, 6. Зафиксируем состояние main.cpp файла.

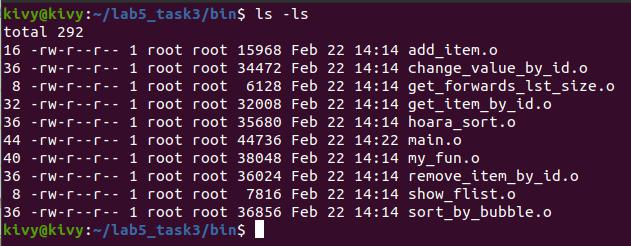


Зафиксируем размеры объектных файлов до пересборки:



Заменим вид сортировки в main.cpp файле и пересоберем проект.





Файл main.o в двух случая весит по-разному (44776 и 44736), что говорит об его изменении. Также факт неизменности остальных файлов подтверждается записями во время сборки и неизменным весом.

**Вывод**: в процессе выполнения лабораторной работы я ознакомился с техникой компиляции программ на языке программирования C (C++) в среде ОС семейства Unix/Linux, а также получил практические навыки использования утилиты GNU make для сборки проекта.