

ภาคผนวก E

การทดลองที่ 5 การพัฒนาโปรแกรมภาษา C บนลินุกซ์

การทดลองนี้คาดว่าผู้อ่านผ่านหัวข้อที่ 3.2 และมีประสบการณ์การเขียนหรือพัฒนาโปรแกรมด้วยภาษา C มาแล้ว ผู้อ่านควรมีความคุ้นเคยกับ IDE (Integrated Development Environment) จากการพัฒนาโปรแกรมและการดีบั๊กโปรแกรมด้วยภาษา C/C++ ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อให้เข้าใจการพัฒนาซอฟต์แวร์ด้วย IDE ชื่อ CodeBlocks บนระบบปฏิบัติการ Raspberry Pi OS/Linux/Unix
- เพื่อให้สามารถสร้าง Makefile เพื่อพัฒนาศักยภาพการทำงานเป็นนักพัฒนาอาชีพ
- เพื่อให้เข้าใจความแตกต่างระหว่างการพัฒนาโปรแกรมภาษา C ด้วย IDE และ Makefile

E.1 การพัฒนาโดยใช้ IDE

โปรแกรมหรือแอปพลิเคชัน IDE ย่อมาจาก Integrated Development Environment ทำหน้าที่ช่วยเหลือโปรแกรมเมอร์ ทดสอบ และอาจรวมถึงควบคุมซอร์สโค้ดให้เป็นปัจจุบัน ขั้นตอนการทดลองนี้เริ่มต้นโดย

1. **ตรวจสอบ**ภายในเครื่องว่ามีโปรแกรมชื่อ CodeBlocks ติดตั้งแล้วหรือไม่ โดยพิมพ์คำสั่งเหล่านี้ลงบนโปรแกรม Terminal

```
$ codeblocks
```

2. หากติดตั้งแล้ว ให้ผู้อ่านข้ามไปข้อที่ 4 ได้ หากไม่มีโปรแกรม ผู้อ่านจะต้องติดตั้ง CodeBlocks โดยพิมพ์คำสั่งเหล่านี้ลงบนโปรแกรม Terminal

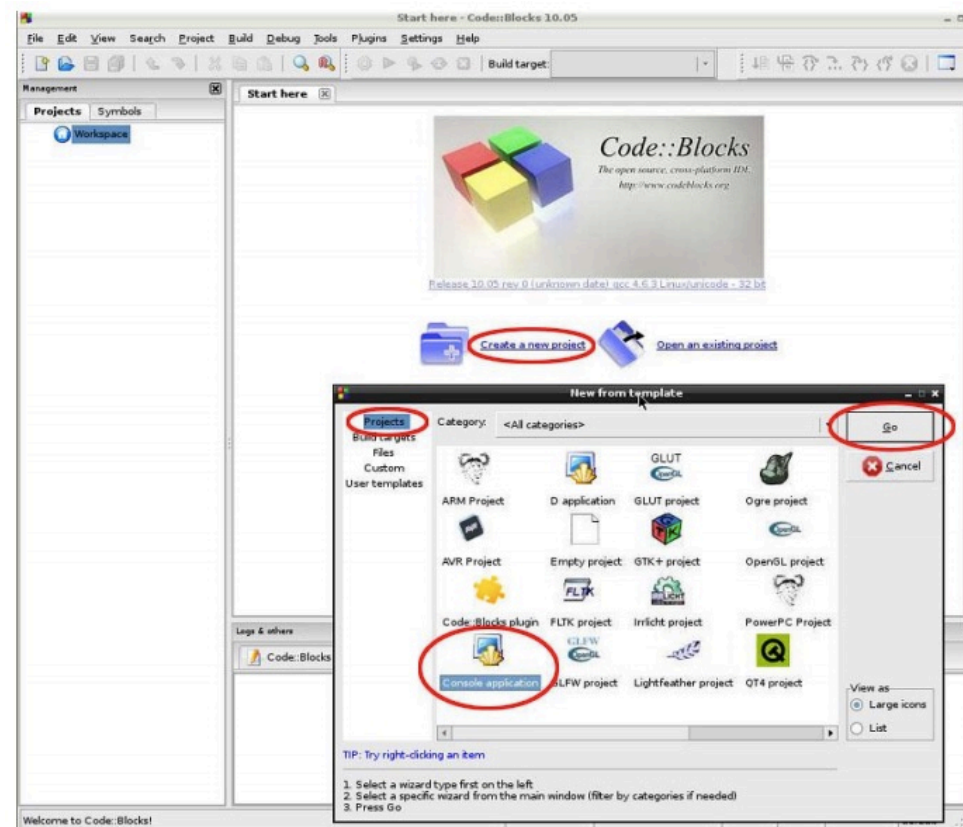
```
$ sudo apt-get install codeblocks
```

คำสั่ง `sudo` นำหน้าคำสั่งใดๆ นี่จะเป็นการเรียกใช้งานคำสั่งนั้นด้วยสิทธิ์ระดับ SuperUser การติดตั้งจะดาวน์โหลดโปรแกรมผ่านทางเครือข่ายอินเทอร์เน็ต และจำเป็นต้องใช้สิทธิ์ระดับสูงสุดนี้

3. เมื่อติดตั้งเสร็จสิ้น พิมพ์คำสั่งนี้เพื่อเริ่มต้นใช้งาน CodeBlocks

```
$ codeblocks
```

4. การใช้งาน CodeBlocks ครั้งแรกจะเป็นการติดตั้งค่า compiler plug-ins เป็น GCC หรือ GNU C Compiler.
5. หน้าต่างหลักจะปรากฏขึ้น หลังจากนั้น ผู้อ่านควรกด "Create a new project" เพื่อสร้างโปรเจกต์ใหม่ ในหน้าต่าง "New from template"



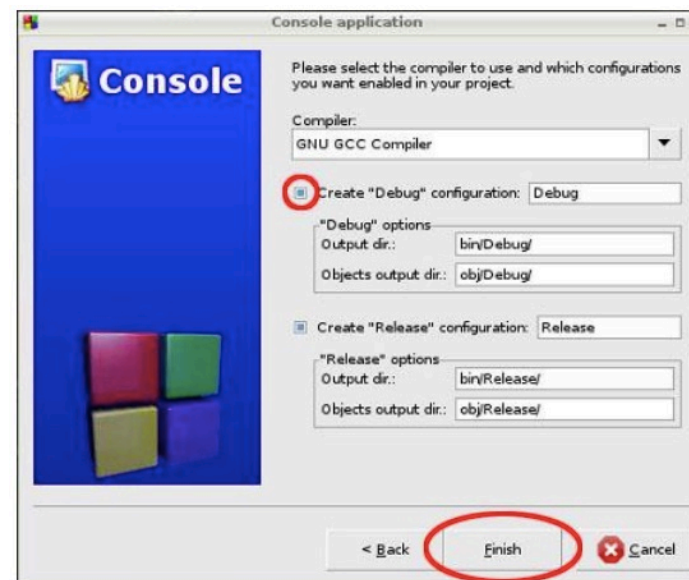
รูปที่ E.1: หน้าต่างเลือกชนิดโปรเจกต์ที่จะพัฒนาเป็นชนิด "Console application"

6. เลือก "New Projects" ในช่องด้านซ้าย แล้วเลือก "Console application" ในช่องด้านขวาเพื่อสร้างโปรแกรมในรูปแบบเท็กซ์โหมด (Text Mode) กดปุ่ม "Go" ตามรูปที่ E.1
7. กดปุ่ม Next> เพื่อดำเนินการต่อ
8. หน้าต่าง "Console application" จะปรากฏขึ้น กดเลือกภาษา "C" เพื่อพัฒนาโปรแกรมแล้วกดปุ่ม "Next>" ตามรูปที่ E.2)



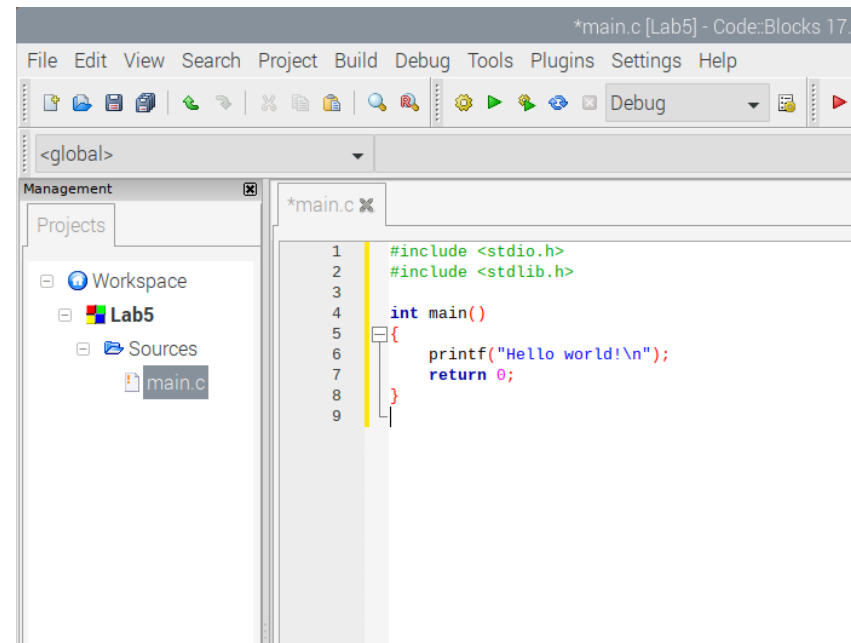
รูปที่ E.2: หน้าต่างเลือกภาษา C หรือ C++ สำหรับโปรเจกต์ที่จะพัฒนา

9. กรอกชื่อโปรเจกต์ใหม่ชื่อ Lab5 ในช่อง Project title: และกรอกชื่อไดเรกทอรี `/home/pi/asm/` ในช่อง Folder to create project in: โปรดสังเกตข้อความในช่อง Project filename: ว่าตรงกับ Lab5.cbp ใช่หรือไม่
10. กดปุ่ม "Next>" เพื่อดำเนินการต่อและสุดท้ายจะเป็นขั้นตอนการเลือกคอนฟิกูเรชัน (Configuration) สำหรับคอมไพเลอร์ในรูปที่ E.3 โดย Debug เหมาะสำหรับการเริ่มต้นและแก้ไขข้อผิดพลาด แล้วจึงกดปุ่ม "Finish" เมื่อเสร็จสิ้นการดีบั๊ก



รูปที่ E.3: การเลือกคอนฟิกูเรชัน (Configuration) Debug สำหรับคอมไพเลอร์ GNU GCC ในโปรเจกต์ Lab5

11. คลิกซ้ายบนชื่อ Lab5 ในหน้าต่าง Management/Workspace ด้านซ้ายมือ เพื่อขยายไดเรกทอรี Sources แล้วจึงคลิกบนไฟล์ไอคอนชื่อ main.c



รูปที่ E.4: การเปิดอ่านไฟล์ main.c ภายใต้โปรเจกต์ Lab5 ที่สร้างขึ้น

คำสั่งเริ่มต้นที่ CodeBlocks สร้างไว้อัตโนมัติในไฟล์ main.c คือ Hello world

12. ป้อนโปรแกรมนี้นแทนที่ของเดิมในไฟล์ main.c

```
#include <stdio.h>
int main(void)
{
    int a;
    printf("Please input an integer: ");
    scanf("%d", &a);
    printf("You entered the number: %d\n", a);
    return 0;
}
```

13. Build->Compile โปรแกรม จนไม่มีข้อผิดพลาด โดยสังเกตจากหน้าต่างด้านล่างสุด

14. รันโปรแกรมเพื่อทดสอบการทำงาน

E.2 การดีบั๊ก (Debugging) โดยใช้ IDE

การดีบั๊กโปรแกรม คือ การตรวจสอบการทำงานของโปรแกรมอย่างละเอียด CodeBlocks รองรับการดีบั๊กผ่านเมนู Debug ผู้อ่านสามารถเริ่มต้นโดย

1. กด Debug บนเมนูแถบบนสุด เลือก Active Debuggers GDB/CDB Debugger เป็นค่าดีฟอลท์ (Target's Default)

2. เลื่อนเคอร์เซอร์ (Cursor) ไปยังบรรทัดที่ต้องการศึกษา กดปุ่ม F5 เพื่อตั้งเบรกพอยน์ (Break Point) ตรงบรรทัดปัจจุบันของเคอร์เซอร์ โปรแกรมจะเริ่มต้นประโยคด้านซ้ายสุดจะมีวงกลมสีแดงปรากฏขึ้น และเมื่อกด F5 อีกครั้งวงกลมสีแดงจะหายไป เรียกว่า **การท็อกเกิล (Toggle)** เบรกพอยน์ กด F5 อีกครั้งเพื่อสร้างวงกลมสีแดงตรงบรรทัดที่สนใจเพียงจุดเดียวเท่านั้น จับภาพหน้าต่างที่ได้วางไว้ได้คำสั่งนี้เพื่อให้**ตรวจสอบ**
3. กดปุ่ม F8 (Start/Continue) บนคีย์บอร์ดเพื่อรันโปรแกรมอีกรอบ โปรแกรมจะรันไปจนหยุดตรงประโยคที่มีวงกลมสีแดงนั้น โปรแกรมจะแสดงสัญลักษณ์สามเหลี่ยมสีเหลืองซ้อนทับกันอยู่ หลังจากนั้น กดปุ่ม F7 (Next line) เพื่อดำเนินการต่อทีละบรรทัด
4. เลื่อนเคอร์เซอร์ไปยังประโยคที่มีวงกลมสีแดง กดปุ่ม F5 บนคีย์บอร์ดเพื่อปลดวงกลมสีแดงออก หรือยกเลิกเบรกพอยน์
5. เริ่มต้นการดีบั๊กใหม่เพื่อศึกษาการทำงานของปุ่ม F4 (Run to cursor) โดยเลื่อนเคอร์เซอร์ไปวางบนประโยคที่สนใจ กดปุ่ม F4 และสังเกตว่าสามเหลี่ยมสีเหลืองจะปรากฏหน้าประโยค เพื่อระบุว่าเครื่องรันมาถึงประโยคนี้แล้ว
6. กดปุ่ม F8 เพื่อรันต่อไป จนถึงสิ้นสุดการทำงานของโปรแกรม
7. ใช้โปรแกรมไฟล์เมนเนเจอร์ค้นหาในไดเรกทอรี `/home/pi/asm/Lab5` ว่า ไฟล์ `main.o` ที่เป็นไฟล์อ็อบเจกต์อยู่ในไดเรกทอรีใด
8. ใช้โปรแกรมไฟล์เมนเนเจอร์ค้นหาในไดเรกทอรี `/home/pi/asm/Lab5` ว่า ไฟล์ `Lab5` ที่เป็นไฟล์โปรแกรมหรือไฟล์ Executable อยู่ในไดเรกทอรีใด

E.3 การพัฒนาโดยใช้ประโยคคำสั่งทีละขั้นตอน

ผู้อ่านควรเข้าใจคำสั่งพื้นฐานในการแปลโปรแกรมภาษา C ที่สร้างขึ้นใน CodeBlocks ก่อนหน้านี้ ตามขั้นตอนต่อไปนี้

1. เปิดโปรแกรม Terminal หน้าต่างใหม่ แล้วย้ายไดเรกทอรีไปยัง `/home/pi/asm/Lab5` โดยใช้คำสั่ง `cd`
2. ทำการคอมไพล์ (Compile) ไฟล์ซอร์สโค้ดให้เป็นไฟล์อ็อบเจกต์ (.o) โดยเรียกใช้คอมไพเลอร์ชื่อ `gcc` ดังนี้

```
$ gcc -c main.c
```

ไฟล์ผลลัพธ์ ชื่อ `main.o` จะปรากฏขึ้น ผู้อ่านต้อง**ตรวจสอบ**โดยใช้คำสั่ง `ls -la` เพื่อ**ตรวจสอบ**วันที่และขนาดของไฟล์ เปรียบเทียบการใช้งานกับรูปที่ [3.10](#) จับภาพหน้าต่างที่ได้วางไว้ได้คำสั่งนี้เพื่อให้**ตรวจสอบ**

```
t63010487@raspberrypi:~/asm/Lab5 $ nano main.c
t63010487@raspberrypi:~/asm/Lab5 $ gcc -c main.c
t63010487@raspberrypi:~/asm/Lab5 $ ls -la
total 16
drwxr-xr-x 2 t63010487 t63010487 4096 Jan 31 10:50 .
drwxr-xr-x 3 t63010487 t63010487 4096 Jan 31 10:45 ..
-rw-r--r-- 1 t63010487 t63010487 150 Jan 31 10:50 main.c
-rw-r--r-- 1 t63010487 t63010487 1888 Jan 31 10:50 main.o
```

3. ทำการลิงก์ (Link) โดยใช้ gcc ทำหน้าที่เป็นลิงก์เกอร์ (Linker) และแปลงไฟล์อ็อบเจกต์เป็นไฟล์โปรแกรม (Executable file) โดย

```
$ gcc main.o -o Lab5
```

ไฟล์โปรแกรม ชื่อ Lab5 จะปรากฏขึ้น ผู้อ่านต้องตรวจสอบโดยใช้คำสั่ง ls -la เพื่อตรวจสอบวันที่และขนาดของไฟล์เพื่อเปรียบเทียบกับ main.o จับภาพหน้าต่างที่ได้วางไว้ได้คำสั่งนี้เพื่อให้ตรวจสอบ

```
t63010487@raspberrypi:~/asm/Lab5 $ gcc main.o -o Lab5
t63010487@raspberrypi:~/asm/Lab5 $ ls -la
total 28
drwxr-xr-x 2 t63010487 t63010487 4096 Jan 31 10:51 .
drwxr-xr-x 3 t63010487 t63010487 4096 Jan 31 10:45 ..
-rwxr-xr-x 1 t63010487 t63010487 9416 Jan 31 10:51 Lab5
-rw-r--r-- 1 t63010487 t63010487 150 Jan 31 10:50 main.c
-rw-r--r-- 1 t63010487 t63010487 1888 Jan 31 10:50 main.o
```

4. รัน (Run) โปรแกรม Lab5 โดยพิมพ์

```
$ ./Lab5
```

```
t63010487@raspberrypi:~/asm/Lab5 $ ./Lab5
Please input an integer: 10
You entered the number; 10
```

5. เปรียบเทียบผลลัพธ์ที่ปรากฏขึ้นว่าตรงกับผลการรันใน CodeBlocks หรือไม่
อย่างไร
.....

E.4 โครงสร้างของ Makefile

นอกเหนือจากการพัฒนาโปรแกรมด้วย IDE แล้ว การพัฒนาด้วย Makefile จะช่วยให้นักพัฒนาเมื่อสมัครเล่นและมีอาชีพดำเนินการได้ถูกต้องและรวดเร็ว ไฟล์ชื่อ Makefile เป็นไฟล์อักษรหรือเท็กซ์ไฟล์ (text file) ง่าย ๆ ที่อธิบายความสัมพันธ์ระหว่าง ไฟล์ซอร์สโค้ดต่างๆ ไฟล์อ็อบเจกต์ และไฟล์โปรแกรม แต่ละบรรทัดจะมีโครงสร้างดังนี้

```
target : prerequisites ...
<tab>recipe
<tab>      ...
<tab>...
```

- target หมายถึง ชื่อไฟล์ที่จะถูกสร้างขึ้น โดยอาศัยไฟล์ต่างๆ จากส่วนที่เรียกว่า prerequisites นอกจากชื่อไฟล์แล้ว คำสั่ง 'clean' สามารถใช้เป็น target ได้ จึงนิยมใช้สำหรับลบไฟล์ต่างๆ ที่ไม่ต้องการ
- recipe หมายถึง คำสั่งหรือการกระทำที่จะใช้รายชื่อไฟล์ใน prerequisites นั้นมาสร้างไฟล์ target ได้สำเร็จ โดยแต่ละบรรทัดจะต้องเริ่มต้นด้วยปุ่ม tab เสมอ

E.5 การพัฒนาโดยใช้ Makefile

ตัวอย่างนี้เป็นการสร้าง Makefile เพื่อใช้คอมไพล์และลิงก์โปรแกรมเดิมที่เรามีอยู่ ผู้อ่านจะได้เข้าใจกลไกการทำงานที่ง่ายที่สุดก่อน หลังจากนั้นผู้อ่านสามารถศึกษาเพิ่มเติมด้วยตนเองได้จากเว็บไซต์หรือตัวอย่างโปรแกรมโอเพนซอร์สที่ซับซ้อนขึ้นเรื่อยๆ ต่อไป

1. ในโปรแกรม Terminal ย้ายไดเรกทอรีปัจจุบันไปที่ `/home/pi/asm/Lab5`
2. เรียกใช้โปรแกรม nano ในหน้าต่าง Terminal

```
$ nano
```

กรอกข้อความเหล่านี้ในไฟล์เปล่าโดยใช้ nano

```
Lab5: main.o
        gcc main.o -o Lab5
main.o: main.c
        gcc -c main.c
clean:
        rm *.o
```

3. เมื่อกรอกเสร็จแล้ว ให้ทำการบันทึก หรือ save โดยตั้งชื่อไฟล์ว่า Makefile หรือ makefile ใดๆอย่างหนึ่งโดยไม่มีนามสกุล หลังจากนั้น และบันทึกในไดเรกทอรี `/home/pi/asm/Lab5` แล้วปิดโปรแกรม nano
4. พิมพ์คำสั่งนี้ใน Terminal

```
$ make clean
```

เพื่อเรียกใช้คำสั่ง `rm *.o` ผ่านทาง Makefile เพื่อลบ (Remove) ไฟล์ที่มีนามสกุล `.o` ทั้งหมด

5. พิมพ์คำสั่งนี้ใน Terminal

```
$ make Lab5
```

เพื่อเรียกใช้คำสั่ง `gcc -c main.c` และ `gcc -g main.c -o Lab5` เพื่อสร้างไฟล์คำสั่ง Lab5 ที่จะทำงานตามซอร์สโค้ด `main.c` ที่กรอกไป โดยไฟล์ Lab5 ที่เกิดขึ้นใหม่จะมีโครงสร้างรูปแบบ ELF

6. พิมพ์คำสั่งนี้ใน Terminal

```
$ ls -la
```

เพื่ออ่านค่าเวลาที่ไฟล์ Lab5 ที่เพิ่งถูกสร้าง โปรดสังเกตสีของชื่อไฟล์ต่างๆ ว่ามีสีอะไรบ้าง และบ่งบอกอะไรตามสีนั้นๆ

```
t63010487@raspberrypi:~/asm/Lab5 $ nano
t63010487@raspberrypi:~/asm/Lab5 $ make clean
rm *.o
t63010487@raspberrypi:~/asm/Lab5 $ make Lab5
gcc -c main.c
gcc main.o -o Lab5
t63010487@raspberrypi:~/asm/Lab5 $ ls -la
total 32
drwxr-xr-x 2 t63010487 t63010487 4096 Jan 31 11:03 .
drwxr-xr-x 3 t63010487 t63010487 4096 Jan 31 10:45 ..
-rwxr-xr-x 1 t63010487 t63010487 9416 Jan 31 11:03 Lab5
-rw-r--r-- 1 t63010487 t63010487 150 Jan 31 10:50 main.c
-rw-r--r-- 1 t63010487 t63010487 1888 Jan 31 11:03 main.o
-rw-r--r-- 1 t63010487 t63010487 79 Jan 31 11:02 makefile
```

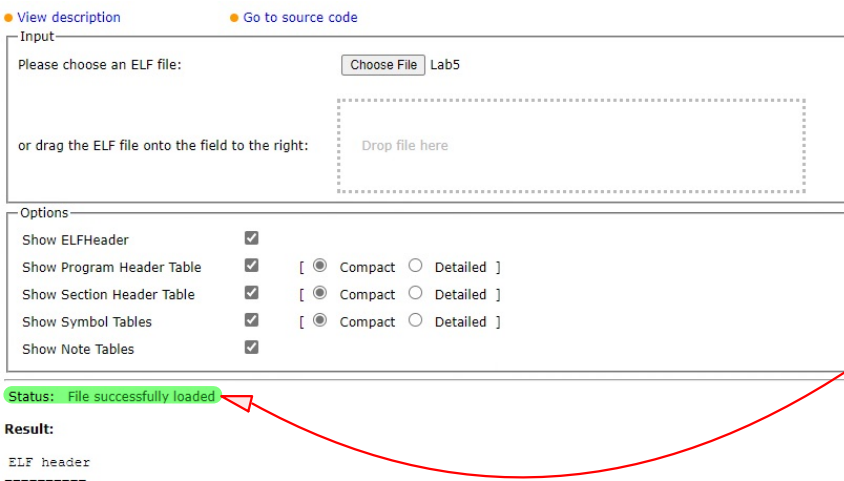
7. พิมพ์คำสั่งนี้ใน Terminal

```
$ ./Lab5
```

เพื่อรันโปรแกรม Lab5 ให้ซีพียูปฏิบัติตาม โดย . หมายถึง `current directory` / หมายถึง `home directory` ว่างภาพหน้าต่างที่ได้วางไว้ได้คำสั่งนี้เพื่อให้ตรวจสอบ

```
t63010487@raspberrypi:~/asm/Lab5 $ ./Lab5
Please input an integer: 6
You entered the number; 6
current คือ Lab5 เข้า dir คือ Lab5
```

- คลิกบนลิงก์ต่อไปนี้ sunshine2k.de เพื่อเปิดเบร่าส์เซอร์และอัปโหลดไฟล์ Lab5 ที่ได้จากการคอมไพล์ และลิงก์ก่อนหน้านี้ ตรวจสอบค่า Status: File successfully loaded หรือไม่
- เปิดเบร่าส์เซอร์ให้เต็มจอแล้วเลื่อนหน้าจอแสดงผลขึ้นเพื่อเปรียบเทียบกับโครงสร้างของไฟล์ ELF ในรูปที่ 3.14 แคปเจอร์หน้าจอบริเวณเท็กซ์เช็กเมนต์และดาด้าเช็กเมนต์ของ Lab5 วางภาพหน้าต่างที่ได้วางไว้ได้คำสั่งนี้เพื่อให้ตรวจสอบ



E.6 การตรวจจับ Overflow คณิตศาสตร์เลขจำนวนเต็มฐานสอง

E.6.1 เลขจำนวนเต็มฐานสองไม่มีเครื่องหมาย

หัวข้อที่ 2.3.1 กล่าวถึงการบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะไม่มีเครื่องหมายด้วยเช่นกัน แต่การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า **การเกิดโอเวอร์โฟลว์ (Overflow)** ในสมการที่ (2.43) ซึ่งเป็นผลสืบเนื่องจากวงจรดิจิทัลที่สามารถประมวลผลได้จำกัด ตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ในตัวอย่างการแปลงเลขฐานสองเป็นฐานสิบที่ได้แสดงไปแล้ว ยกตัวอย่างเช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยคในภาษา C/C++ ตามการทดลองต่อไปนี้

1. ทดสอบโปรแกรมภาษา C ต่อไปนี้

```
#include <stdio.h>

int main()
{
    unsigned int i=1;
    while (i>0) {
```

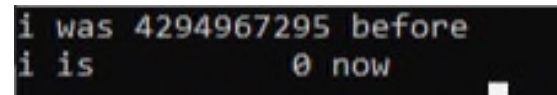
| Section header tables | | | | | | | | | | |
|-----------------------|--------------------|---------------|---------------------|---------------------|---------------------|------------|------------|--------------------|--------------------|----------------|
| Nr | Name | Type | Address | Offset | Size | Link | Info | AddrAlign | EntSize | Flags |
| 0 | | SH_NULL | 0x0000000000000000 | 0x0000000000000000 | 0x0000000000000000 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000000 | |
| 1 | .interp | SH_PROGBITS | 0x0000000000000238 | 0x0000000000000238 | 0x0000000000000018 | 0x00000000 | 0x00000000 | 0x0000000000000001 | 0x0000000000000000 | Alloc |
| 2 | .note.gnu.build-id | SH_NOTE | 0x0000000000000254 | 0x0000000000000254 | 0x0000000000000024 | 0x00000000 | 0x00000000 | 0x0000000000000004 | 0x0000000000000000 | Alloc |
| 3 | .note.ABI-tag | SH_NOTE | 0x0000000000000278 | 0x0000000000000278 | 0x0000000000000020 | 0x00000000 | 0x00000000 | 0x0000000000000004 | 0x0000000000000000 | Alloc |
| 4 | .gnu.hash | Unknown | 0x0000000000000298 | 0x0000000000000298 | 0x000000000000001C | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000000 | Alloc |
| 5 | .dynsym | SH_DYNSYM | 0x00000000000002B8 | 0x00000000000002B8 | 0x00000000000000F0 | 0x00000006 | 0x00000003 | 0x0000000000000000 | 0x0000000000000018 | Alloc |
| 6 | .dynstr | SH_STRTAB | 0x00000000000003A8 | 0x00000000000003A8 | 0x0000000000000089 | 0x00000000 | 0x00000000 | 0x0000000000000001 | 0x0000000000000000 | Alloc |
| 7 | .gnu.version | Unknown | 0x0000000000000432 | 0x0000000000000432 | 0x0000000000000014 | 0x00000000 | 0x00000000 | 0x0000000000000002 | 0x0000000000000002 | Alloc |
| 8 | .gnu.version_r | Unknown | 0x0000000000000448 | 0x0000000000000448 | 0x0000000000000020 | 0x00000006 | 0x00000001 | 0x0000000000000000 | 0x0000000000000000 | Alloc |
| 9 | .rela.dyn | SH_RELA | 0x0000000000000468 | 0x0000000000000468 | 0x00000000000000F0 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000018 | Alloc |
| 10 | .rela.plt | SH_RELA | 0x0000000000000558 | 0x0000000000000558 | 0x0000000000000078 | 0x00000000 | 0x00000016 | 0x0000000000000000 | 0x0000000000000018 | Alloc InfoLink |
| 11 | .init | SH_PROGBITS | 0x00000000000005D0 | 0x00000000000005D0 | 0x0000000000000000 | 0x00000000 | 0x00000000 | 0x0000000000000004 | 0x0000000000000000 | Alloc Exec |
| 12 | .plt | SH_PROGBITS | 0x00000000000005F0 | 0x00000000000005F0 | 0x0000000000000070 | 0x00000000 | 0x00000000 | 0x0000000000000010 | 0x0000000000000000 | Alloc Exec |
| 13 | .text | SH_PROGBITS | 0x0000000000000660 | 0x0000000000000660 | 0x0000000000000014 | 0x00000000 | 0x00000000 | 0x0000000000000010 | 0x0000000000000000 | Alloc Exec |
| 14 | .fini | SH_PROGBITS | 0x0000000000000874 | 0x0000000000000874 | 0x0000000000000010 | 0x00000000 | 0x00000000 | 0x0000000000000004 | 0x0000000000000000 | Alloc Exec |
| 15 | .rodata | SH_PROGBITS | 0x00000000000008B8 | 0x00000000000008B8 | 0x000000000000002F | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000000 | Alloc |
| 16 | .eh_frame_hdr | SH_PROGBITS | 0x00000000000008B8 | 0x00000000000008B8 | 0x0000000000000044 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000000 | Alloc |
| 17 | .eh_frame | SH_PROGBITS | 0x0000000000000900 | 0x0000000000000900 | 0x00000000000000E4 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000000 | Alloc |
| 18 | .init_array | SH_INIT_ARRAY | 0x00000000000010B8 | 0x00000000000010B8 | 0x0000000000000008 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000008 | Write Alloc |
| 19 | .fini_array | SH_FINI_ARRAY | 0x00000000000010DC0 | 0x00000000000010DC0 | 0x0000000000000008 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000008 | Write Alloc |
| 20 | .dynamic | SH_DYNAMIC | 0x00000000000010DC8 | 0x00000000000010DC8 | 0x00000000000001E0 | 0x00000006 | 0x00000000 | 0x0000000000000000 | 0x0000000000000010 | Write Alloc |
| 21 | .got | SH_PROGBITS | 0x00000000000010FA8 | 0x00000000000010FA8 | 0x0000000000000040 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000008 | Write Alloc |
| 22 | .got.plt | SH_PROGBITS | 0x00000000000010FAB | 0x00000000000010FAB | 0x0000000000000040 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000008 | Write Alloc |
| 23 | .data | SH_PROGBITS | 0x00000000000011028 | 0x00000000000011028 | 0x0000000000000010 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000000 | Write Alloc |
| 24 | .bss | SH_NOBITS | 0x00000000000011038 | 0x00000000000011038 | 0x0000000000000008 | 0x00000000 | 0x00000000 | 0x0000000000000001 | 0x0000000000000000 | Write Alloc |
| 25 | .comment | SH_PROGBITS | 0x0000000000000000 | 0x0000000000000138 | 0x0000000000000027 | 0x00000000 | 0x00000000 | 0x0000000000000000 | 0x0000000000000001 | Merge Strings |
| 26 | .symtab | SH_SYMTAB | 0x0000000000000000 | 0x0000000000000180 | 0x00000000000000B0 | 0x00000000 | 0x00000046 | 0x0000000000000000 | 0x0000000000000018 | |
| 27 | .strtab | SH_STRTAB | 0x0000000000000000 | 0x0000000000000180 | 0x0000000000000031A | 0x00000000 | 0x00000000 | 0x0000000000000001 | 0x0000000000000000 | |
| 28 | .shstrtab | SH_STRTAB | 0x0000000000000000 | 0x0000000000001C4A | 0x00000000000000103 | 0x00000000 | 0x00000000 | 0x0000000000000001 | 0x0000000000000000 | |


```

        i=i+1;
    if (i==0) {
        printf("i was %10u before\n",i-1);
        printf("i is %10u now\n",i);
    }
}
return 0;
}

```

- อธิบายว่า ประกาศตัวแปรจึงตั้งค่าเริ่มต้น unsigned int i=1; ~~ในคณณ int โดยเริ่ม + 1 - โดยค่าเริ่ม 1~~ ^{ในคณณ int โดยเริ่ม + 1 - โดยค่าเริ่ม 1}
- การวนลูปเพิ่มค่า i=i+1 จน i มีค่าเป็นศูนย์แล้วแสดงผลค่าของ i มาทางหน้าจอ เมื่อเกิดเหตุการณ์อะไร เพราะเหตุใด Overflow
- จับภาพหน้าต่างที่ได้วางไว้ได้คำสั่งนี้เพื่อให้ตรวจสอบ



```

i was 4294967295 before
i is 0 now

```

E.6.2 เลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement

หัวข้อที่ ?? กล่าวถึงการบวกเลขจำนวนเต็มชนิดมีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะมีเครื่องหมายด้วยเช่นกัน แต่การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า **การเกิดโอเวอร์โฟลว์ (Overflow)** ในสมการที่ (2.47) ซึ่งเป็นผลสืบเนื่องจากวงจรดิจิทัลที่สามารถประมวลผลได้จำกัด ตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ในตัวอย่างการแปลงเลขฐานสองเป็นฐานสิบที่ได้แสดงไปแล้ว ยกตัวอย่างเช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยคในภาษา C/C++ ตามการทดลองต่อไปนี้

- ทดสอบโปรแกรมภาษา C ต่อไปนี้

```

#include <stdio.h>
int main()
{
    int i=1;
    while (i>0) {
        i=i+1;
        if (i<0) {
            printf("i was %10d before\n",i-1);

```

```
        printf("i is %10d now\n", i);
        break;
    }
}
return 0;
}
```

- อธิบายว่า ประกาศตัวแปรจึงตั้งค่าเริ่มต้น int i=1; ...ไปจนกว่าจะถึงผลป.ป.ต. โดยค่าของ i
- การวนลูปเพิ่มค่า i=i+1 จน i มีค่าน้อยกว่าศูนย์แล้วแสดงผลค่าของ i มาทางหน้าจอ เมื่อเกิดเหตุการณ์อะไร เพราะเหตุใด Overflow.....
- จับภาพหน้าต่างที่ได้วางไว้ได้คำสั่งนี้เพื่อให้ตรวจสอบ

```
i was 2147483647 before
i is -2147483648 now
```

E.7 กิจกรรมท้ายการทดลอง

- จงเปรียบเทียบไฟล์การพัฒนาโปรแกรมในภาคผนวกนี้กับรูปที่ 3.9
- ใน Terminal จงย้ายไดเรกทอรีปัจจุบันไปที่ /home/pi/asm/Lab5

\$ ls -la

เพื่ออ่านรหัสสีของชื่อไฟล์ต่างๆ
- จงพัฒนาโปรแกรมภาษา C โดยประกาศตัวแปรและตั้งค่าเริ่มต้น int i=-1 และให้วนลูปลดค่า i=i-1 จน i มีค่าเป็นบวกแล้วแสดงผลค่าของ i ออกมาทางหน้าจอ โดยใช้โปรแกรมในหัวข้อที่ E.6.2 เป็นต้นแบบ
- จงพัฒนาโปรแกรมภาษา C ให้สามารถอ่านไฟล์ Makefile เพื่อแสดงตัวอักษรในไฟล์ที่ละตัวและค่ารหัส ASCII ฐานสิบหกของตัวอักษรนั้นบนหน้าจอ แล้วปิดไฟล์เมื่อเสร็จสิ้น
- จงพัฒนาโปรแกรมภาษา C เพื่อสั่งพิมพ์เลขอนุกรม Fibonacci โดยรับค่าเลขเป้าหมาย n ซึ่งเกิดจาก n = (n-1) + (n-2) และรายละเอียดเพิ่มเติมได้จาก [wikipedia](#) ตัวอย่างต่อไปนี้ n=5 และพิมพ์ผลลัพธ์ดังนี้
1 1 2 3 5

ଉଦା ୩

test.c

```
1 #include<stdio.h>
2 int main()
3 {
4     int i=-1;
5     while (i<0) {
6         i=i-1;
7         if (i>0) {
8             printf("i was %10d before\n",i+1);
9             printf("i is %10d now\n",i);
10            break;
11        }
12    }
13    return 0;
14 }
```

```
i was -2147483648 before
i is 2147483647 now
```

```
-----
Process exited after 4.668 seconds with return value 0
Press any key to continue . . .
```

ଉଦା ୫

test.c

```
1 #include<stdio.h>
2 int main()
3 {
4     int f1=1,f2=1;
5     int n,temp;
6     printf("n=");
7     scanf("%d",&n);
8     printf("%d %d ",f1,f2);
9     while (f1+f2<n+f1) {
10        temp = f2;
11        f2 = f1+f2;
12        f1 = temp;
13        printf("%d ",f2);
14    }
15    return 0;
16 }
```

n=5

1 1 2 3 5

```
-----
Process exited after 0.7199 seconds with return value 0
Press any key to continue . . .
```