Practical 3

Introduction to C - Decimal to radix-i converter

Practical 3 has a number of sub-tasks, please complete the tasks in order.
Read and revise:
Week 1, Week 5, Week 6 on Amathuba lessons.

. You may also want to refer to the **Modern C** textbook for more detailed information.

Practical instructions

1. Read the practical instructions.
2. Download and install a suitable C compiler (and IDE) for your PC/Mac. Possible options include but are not limited to:
   (a) A compiler such as (Windows: **MinGW** (g++); MacOS: Clang (run xcode-select –install in the terminal) or GCC; Linux: GCC)
   (b) An IDE such as **Visual Studio Code** (Windows 32-bit/64-bit; Linux 32-bit/64-bit; Mac OS 32-bit/64-bit (Intel/M processors))
   (c) An IDE such as **Code::Blocks** (Windows 32-bit/64-bit; Linux 32-bit/64-bit; Mac OS X (pre Catalina))
   (d) **CLion** (Windows 32-bit/64-bit; Linux 32-bit/64-bit; Mac OS X 32-bit/64-bit (Intel/M processors)) You can register for a free educational licence with your @myuct.ac.za email address.
3. Complete the online pre-practical quiz on Amathuba
4. Complete your code. Remember to bring your laptop to your in person lab session.
5. Complete the online post-practical quiz on Amathuba
6. Attend and demonstrate your practical

3.1 Introduction

The aim of this practical is to introduce you to basic C programming by designing a decimal to radix-i converter program where $n \in \{2, 3, 4, \ldots, 16\}$.

This practical contains both online practical quizzes (pre- and post-) and physical testing. Please ensure that you submit your quizzes on Amathuba by the due dates. There will be an in person practical demonstration session during term where you will be demonstrating the operation of your code to a TA or tutor.

3.2 Background: The compilation process

C was developed in the late 1960s and early 1970s in Bell Labs by Dennis Ritchie as a cross-platform general purpose programming language that allowed low level memory access and control. This meant that it could be used to program many types of processor architecture from microcontrollers to personal computers. It has subsequently been standardised by the *American National Standards Institute* (ANSI) and later the *International Standards Organisation* (ISO) which enables portability between devices. These standards define the language syntax as well as the standard libraries such as **stdint.h** and **stdio.h**. The latest standard is C11 or more formally ISO/IEC 9899:2011.

C is a *procedural* programming language. This means that it is made up of a set of *procedures* called functions/sub-routines. Each *function* contains a set of instructions that are run sequentially. A function can make calls to other functions from within it. The instructions in these functions will then run first, before the program returns to where it left off. A logical list of instructions and functions are saved as a *source file* [1]. A source file is a plain text (**.c**) file. This file contains a series of digital 8-bit bytes each representing a single ASCII[1] character (see Appendix A.4). However a processor can only process machine code (opcodes and memory/register addresses). There therefore needs to be a process which can convert the (**.c**) file into an executable *binary file* containing a sequential set of machine instructions. This is performed in stages by four programmes:

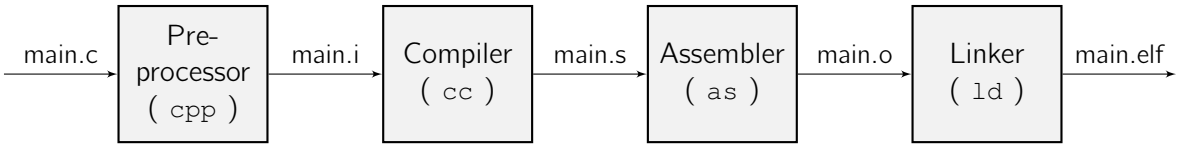

Figure 3.1: A block diagram showing the compilation process. Adapted from [1].

- **Pre-processor** runs through the original text C program and implements all pre-processor commands (prefaced by # character). It produces a text-based C file with the **.i** file extension.
- **Compiler** converts the C instructions into assembly instructions. Each assembly instruction represents one machine-code instruction but in text form, making it easier for a programmer to understand, however it still needs to be further processed before it can be read by the target processor. This produces a text-based assembly file with the **.s** file extension.
- **Assembler** converts the the text-based assembly program into a binary *relocatable object file*. Each byte in this file encodes machine-code instructions, not ASCII characters as in the previous files. This file is *relocatable* as the final addresses for each section have not been set at this stage. This produces a binary object file with the **.o** file extension.

[1] The American Standard Code for Information Interchange is a standardised encoding procedure for text characters.

- **Linker** links all object files together and combines them into a single executable binary file, where the final absolute address for each section is assigned. This file can be run on the target. The function of the linker is to tie all of the object code files together to create one linked set of code. All cross references within the linker are checked and tied together. All of the symbolic addresses (labels, variable names) within your code are converted to absolute memory addresses and all C library references are tied into your code where they have been called. This requires that the linker knows several basic facts about your system. The linker needs to know where to place code in memory. The linker also needs to know where to place variables in memory. Finally the linker needs to know which files to link together and which libraries are to be used. It then produces a binary executable with the **.elf** or **.exe** file extension.

A *toolchain* is a collection of software "tools" and programs which are used to develop source code, convert the code into the appropriate form and then debug, program and run the code on the target processor.

3.3 Programming Task

In this practical you will learn how to display values to the screen, read in values from the keyboard, create preprocessor constants, loops and functions using C. You will then use these skills to write a decimal to radix-i conversion program.

3.3.1 The `printf` function

The **`printf`** function writes a 'string'[2] to the standard output stream (the screen in this case). This C function is part of the **stdio.h** library. A brief description of the function and how to use this it is given below.

```
int printf (const char *format, ... );
```

It can be used as follows (Modified from [3]):

```
/*===================================================================
// printf format specifiers
//-------------------------------------------------------------------
printf("Hello world!\n");
printf("Characters: %c %c \n",'a', 65);
printf("Decimals: %d %ld \n",1977, 650000L);
printf("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
printf("Floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, -3.1416);
printf("%s \n","Welcome to EEE2046F");
```

Outputs the following to the screen

[2] Note C does not have a 'string' datatype. Stings are represented by an array of characters. If we want this to be displayed correctly, each character in the array must be the ASCII value for the character. The **`printf`** function converts all numerical values to their correct ASCII characters before incorporating them into the array, based on the value of the format specifier.

```
Hello world!
Characters: a A
Decimals: 1977 650000
Some different radices: 100 64 144 0x64 0144
Floats: 3.14 +3e+000 3.141600E+000
Welcome to EEE2046F
```

For more detailed information on the function and all the format specifiers see Cplusplus reference site.

3.3.2 The `scanf` function

The **`scanf`** function stores input from the standard input stream, in a specified format, to a location pointed to by a reference to a variable. This C function is part of the **stdio.h** library.

```
int scanf (const char *format, ... );
```

It can be used as follows[3]:

```
//=====================================================================
// scanf
//---------------------------------------------------------------------
int age;
char name[80];

printf("Enter your name: ");
scanf("%79s",name);
printf("Enter your age: ");
scanf("%d",&age);
```

Outputs the following to the screen

```
Enter your name: Robyn
Enter your age: 20
```

For more detailed information on the function and all the format specifiers see Cplusplus reference site.

3.4 Task 1: Online pre-practical quiz

Please answer the following questions in your pre-practical Q&A quiz **before the start of the practical**. This will be used as a reference document during the practical.

(a) Read through the questions in Section 3.5 and draw a flow chart which describes the **FULL** radix-i program operation. Upload a pdf version of this to the pre-practical quiz on Amathuba.

[3] Note the `scanf` function, expects an address (or reference) of where the data are to be written in memory, as an argument after the format specifier. To access the address of a variable we use the unary `&` operator in front of the variable name. The name of an array is already a reference or address to the first value in the array and therefore does not need the `&` operator to extract the address.

(b) Write out the step by step algorithm using pseudocode to convert a positive decimal integer into any radix-i value (where $i \in \{2,3,4,\ldots,16\}$). Your algorithm must display the resulting number using the correct symbolic set for that radix. For example:

- $(254)_{10} = (11111110)_2$
- $(254)_{10} = (376)_8$
- $(254)_{10} = (FE)_{16}$

This algorithm will form the basis of your conversion function in the practical. Upload a pdf version of this to the pre-practical quiz on Amathuba.

We will now start our program design, which we are going to complete step by step.

3.5 Questions and coding

In your **.c** file start to write the following program step by step. Compile and run your code after each question and make sure it behaves as expected, troubleshooting as you progress. Ensure that you provide detailed comments in each section. Use your flow chart and pseudocode written in your pre-practical write up to help with the coding in this practical.

(a) Write a basic program, using the template, which displays the following on the screen. Copy this code segment and upload it to the post-practical quiz on Amathuba.

```
*****************************
 DECIMAL TO RADIX-i converter
 Written by: Name
 Date: 2023
*****************************
```

(b) Create three constants called `TITLE`, `AUTHOR` and `YEAR` using the preprocessor command `#define`. Update your `printf` code from step one to use the information from these constant values. Copy this code segment and upload it to the post-practical quiz on Amathuba.

(c) Now add code to your program that will prompt an external user to enter a positive decimal integer and display it on the screen, as follows:

```
*****************************
 DECIMAL TO RADIX-i converter
 Written by: Name
 Date: 2023
*****************************
Enter a decimal number: 23
The number you have entered is 23
```

Copy this code segment and upload it to the post-practical quiz on Amathuba.

(d) Add code to your program that will prompt an external user to enter an integer between 2 and 16, which will be the radix of the number system that we will convert our decimal value to, and display it on the screen as follows:

```
*****************************
 DECIMAL TO RADIX-i converter
 Written by: Name
 Date: 2023
*****************************
Enter a decimal number: 23
The decimal number you have entered is 23
Enter a radix for the converter between 2 and 16: 3
The radix you have entered is 3
```

Copy this code segment and upload it to the post-practical quiz on Amathuba.

(e) Modify your program so that it, continuously prompts the user to enter a decimal number and radix and display it on the screen, until the user types in a number less than 0. After which the program should terminate and display the message `EXIT` on the screen.

```
*****************************
 DECIMAL TO RADIX-i converter
 Written by: Name
 Date: 2023
*****************************
Enter a decimal number: 23
The number you have entered is 23
Enter a decimal number: 1
Enter a radix for the converter between 2 and 16: 3
The radix you have entered is 3
Enter a decimal number: -1
EXIT
```

Copy this code segment and upload it to the post-practical quiz on Amathuba.

(f) Now find the $\log_2 n$ of the entered decimal number and display the result to the screen[4]. Copy this code segment and upload it to the post-practical quiz on Amathuba.

(g) Now display the entered decimal number divided by the radix and display the integer result to the screen. Copy this code segment and upload it to the post-practical quiz on Amathuba.

(h) Now display the remainder of the decimal number divided by radix and display the result to the screen. Copy this code segment and upload it to the post-practical quiz on Amathuba.

The output from your code should now look like this:

```
*****************************
 DECIMAL TO RADIX-i converter
 Written by: Name
 Date: 2023
*****************************
Enter a decimal number: 16
The number you have entered is 16
Enter a radix for the converter between 2 and 16: 2
The radix you have entered is 2
The log2 of the number is 4.00
The integer result of the number divided by 2 is 8
The remainder is 0
Enter a decimal number: -1
EXIT
```

We are now going to write a C program to convert an entered decimal number and display it as a radix-i value on the screen.

(i) Create a function called `char* Dec2RadixI(int decValue, int radValue)` which takes in the decimal number and the radix value entered by the user and displays the equivalent radix-i symbol[5]. The output from your code should now look like this:

[4] HINT: Include the **math.h** library and look up the function `log2()`

[5] Note that we follow the same single symbol convention as Hexadecimal for values above 9_{10}.

```
****************************
 DECIMAL TO RADIX-i converter
 Written by: Name
 Date: 2023
****************************
Enter a decimal number: 16
The number you have entered is 16
Enter a radix for the converter between 2 and 16: 2
The radix you have entered is 2
The log2 of the number is 4.00
The integer result of the number divided by 2 is 8
The remainder is 0
The radix-2 value is 10000
Enter a decimal number: -1
EXIT
```

Copy this code segment and upload it to the post-practical quiz on Amathuba.

Ensure that your main.c file is well commented and upload it to Amathuba. It will be compiled, run and evaluated by the tutors.

3.6 Task 2.2: Online post-practical quiz

Test your code for each of the following conditions and enter the outputs in the post-practical quiz on Amathuba.

1. Decimal number: 0; Radix: 2
2. Decimal number: 15; Radix: 2
3. Decimal number: 24; Radix: 3
4. Decimal number: 47; Radix: 4
5. Decimal number: 61; Radix: 5
6. Decimal number: 145; Radix: 6
7. Decimal number: 512; Radix: 7
8. Decimal number: 1325; Radix: 8
9. Decimal number: 1325; Radix: 9
10. Decimal number: 6940; Radix: 10
11. Decimal number: 7400; Radix: 11
12. Decimal number: 8000; Radix: 12
13. Decimal number: 9573; Radix: 13
14. Decimal number: 10492; Radix: 14
15. Decimal number: 12243; Radix: 15
16. Decimal number: 12345; Radix: 16
17. Decimal number: 32; Radix: 2
18. Decimal number: 128; Radix: 2
19. Decimal number: 512; Radix: 2
20. Decimal number: -1