ENGG1410: Introductory Programming for Engineers

Lab #5 | "Introduction to C Programming": Creating Functions in C

Eric Cao, Jeremiah George

October 22, 2024

Contents

Question 1 – Calculations in Different Bases

Objective

Create a C program that repeatedly takes two numbers in a specific base, performs arithmetic (i.e. add "+", subtract "-", multiply "*", divide "/") on both numbers, and outputs the result to the terminal in another base. Functions are implemented for:

1. **Input validation** – the first function checks that the user's input is in the correct format.
2. **Base conversion** – a function can handle converting between bases from 2 to 9, inclusive.
3. **Arithmetic operations** – each of the four (4) arithmetic operators have their own specific function.

```
Give input ($ to stop): / 10 743 18 4
743 / 18 (base 10) = 743 / 18 = 221 (base 4)
Give input ($ to stop): # 8 27 15 6
Invalid operator
Give input ($ to stop): - 4 23 11 15
Invalid base
```

Figure 1: example of terminal including the program responding to invalid inputs

Constraints

- The user can submit inputs to perform arithmetic on different operators continuously.
- When asked for the operator the user inputs "$", the program is terminated.
- Any invalid character in an input sequence results in the program prompting the user to reenter the entire sequence.
- The input or output bases can be from 2 to 9, inclusive; all other bases are invalid.
- If the input number includes digits invalid for its particular base

Pseudocode

List of functions:

- *Input validation* "VaildInput" – **receives an input sequence and validates** each portion of the sequence within the constraints.
- *Base-10 converter* "ctofb10" – converts a number either **from base-b to base-10** or vice versa.
- *Operation selector* "operate" – **chooses** the **operation** ("add", "sub", "mult", "div") performed on the two numbers **in a starting base** and **returns** the result **in the target base**.
- *Addition* "add" – **adds** two numbers in base 10.
- *Subtraction* "sub" – **subtracts** two numbers in base 10.
- *Multiplication* "mult" – **multiplies** two numbers in base 10.
- *Division* "div" – **divides** two numbers in base 10.
- "main" – repeatedly **calls VaildInput**. The function tells the user if the sequence is invalid or **calls operate** and displays the resultant value to the terminal.

Function 1: VaildInput

Return values: 1 (valid input), 2 (exit program), 0 (invalid input)

Character parameters (by reference): operator (o)
Integer parameters (by reference): initial base (b1), *first number* (n1), *second number* (n2), *final base* (b2)

Boolean variable: isInvalid = **false**

```
o <- received input (ignore spaces)
switch (go to first case where condition is true) {
      o == '+' or o == '-' or o == '*' or == '/': break out of switch
      o == '$':
            return 2
            break
      default case:
            output -> "invalid operator"
            isInvalid = true
            break
}

b1, n1, n2, b2 <- received input (ignore spaces)

if (isInvalid == true) return 0

for every digit in n1, n2 if digit < 0 or digit > b1 {
      output -> "invalid operand"
      return 0
}

return 1
```

Function 2: ctofb10

Return value: converted number

```
Integer parameters: number (n), base (b),
startingBase, targetBase
Boolean parameter: isToTen

if isToTen == true {
      startingBase = b
      targetBase = 10
} else {
      startingBase = 10
      targetBase = b
}

Integer variable: result = 0, power = 0

repeat while number > 0 {
      Integer variable: remain = remainder of (number / targetBase)
      number = number / targetBase
      result = result + (remain * startingBase ^ power)

      power = power + 1
}
```

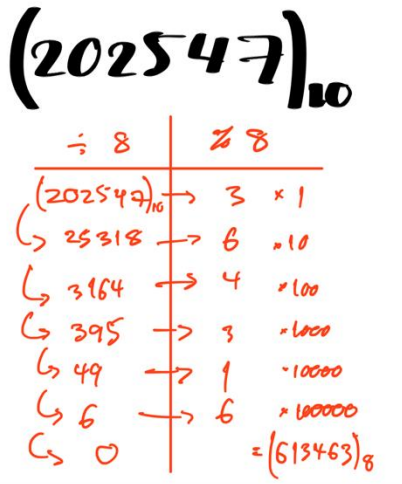

Figure 2: visual representation of converting from base 10 to base 8 by repeatedly dividing 8 and multiplying each remainder by increasing powers of 10, starting from 0.

Function 3: operate

Return value: result of operation in new base

```
Integer parameters: startingBase, numbers (n1) and (n2), targetBase
Character parameter: operator (o)

Integer variables: numbers in base 10 (n1Base10) and (n2Base10)
n1Base10 = ctofb10(n: n1, b: startingBase, isToTen: false)
n2Base10 = ctofb10(n: n2, b: startingBase, isToTen: false)

Integer variable: result
if (o == '+') result = add(n1Base10, n2Base10)
else if (o == '-') result = sub(n1Base10, n2Base10)
else if (o == '*') result = mult(n1Base10, n2Base10)
else result = div(n1Base10, n2Base10)
```

Function 4, 5, 6, 7: add, sub, mult, div

Return value: result after operation

Integer parameters: numbers (n1) *and* (n2)

The C operator for integer addition, subtraction, multiplication and division are wrapped in their respective functions to handle operations in base 10.

Function 8: main

```
Integer variables: starting base (b1), target base (b2), numbers (n1) and
(n2)
Character variable: output (o)

repeat forever {
      Integer variable: condition
                        = VaildInput(pass by reference: o, b1, n1, n2, b2)
      if condition == 0 skip to next iteration
      else if condition == 2 break out of repeat
      else {
      output -> (
         • User input – n1, o, n2, b1, b2
         • Numbers converted to base 10 –
                ctofb10(n1, b1, true), ctofb10(n2, b1, true)
         • Final result after operations and base conversion –
                operate(o, b1, n1, n2, b2))
      }
}
```

Conclusion

In this lab, Jeremiah and I use functions to modularize repeated sections of code. Based on the lab guidelines, we settled on creating functions for a number converter between base 10 and bases 2-9, as well as choosing what operation to perform and validating a user's input. The result is a main function with less complexity and is easier to read by other programmers. Along the way, we would debug functions independently and then we would also discuss how our different functions would interact with each other.