National University of Singapore
School of Continuing and Lifelong Education

TIC1001: Introduction to Computing and Programming I
Semester I, 2019/2020

Lab 1
Mental State of Computation

This lab exercise will teach you how to set up the lab PC and familiarise you with the edit/compile/run cycle of C/C++ programming. You will also try writing a simple program and submitting it on Coursemology.

Understanding the State of Computation

When a program executes, state of computation is simply the contents of the computer memory at that point in time. For C/C++ programs, that would be organised as symbols and their respective values.

A symbol is just another fancy name for variables and functions. In Q1 of tutorial 1, you were required to trace through the execution of the following program:

```c
#include <stdio.h>

int main(void) {
    int cur, prev1=1, prev2=1;

    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    printf("cur is %d; prev1 is %d; prev2 is %d\n", cur, prev1, prev2);

    return 0;
}
```

1. What are the symbols in this program?

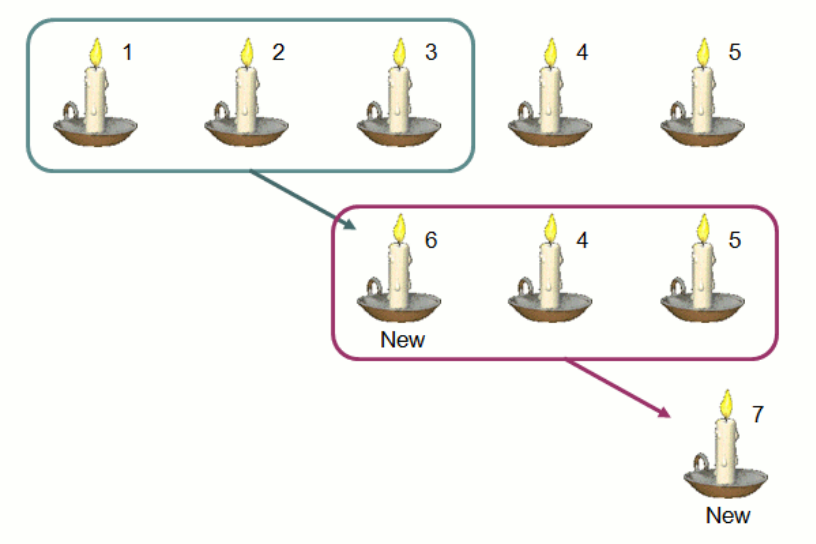
Solving a Problem

Let us now see how having a mental picture of the computation state can help us solve a problem.

Task 1

Alexandra has n candles. He burns them one at a time and carefully collects all unburnt residual wax. Out of the residual wax of exactly k (where $k > 1$) candles, he can roll out a new candle.

The diagram below illustrates the case of n = 5 and k = 3.



After burning the first 3 candles, Alexandra has enough residual wax to roll out the 6th candle. After burning this new candle with candles 4 and 5, he has enough residual wax to roll out the 7th candle. Burning the 7th candle would not result in enough residual wax to roll out anymore new candle. Therefore, in total he can burn 7 candles.

Implement a function `int candles(int n, int k)` that takes as input two positive integers n and k, and returns the total number of candles that can be burn.

Before starting to write any code, you first need to come up with the algorithm, or strategy to solve the problem on paper. One way to start is to try working out the solution with some sample inputs. The solution for $n = 5, k = 3$ is shown above. Can you solve $n = 6, k = 4$?

Seems pretty straightforward right? What about $n = 6, k = 2$?

When you are done, ask yourself:

1. When solving this problem on paper, what state did you maintain?
2. What was your strategy to get the answer?
3. How did the state transform with every step in your process?

To solve problems computationally, it is much simpler if you are consistent in how the state is manipulated, e.g., instead of randomly selecting candles to burn, be consistent in the number of candles to always burn.

4. Does your strategy employ a consistent state transformation?

Now express your transformations using the control structures that we learnt.

5. If your transformation is dependent on some condition, you will use _____.
6. If you repeat the transformation, you can use _____.

Our Solution

We strongly recommend you try and solve this simple problem on your own before reading our answer.

Here is an example of a possible solution. Note that in programming, there can be several ways to solve a problem. This is just one example.

Our State

Here are the symbols, or variables that we maintain in our state.

- n: the number of currently unburnt candles.
- k: the number of burnt candles that can be rolled into one new candle.
- $count$: maintain the total number of candles we have burnt.

Did you have these variables in your strategy too?

Our Algorithm

1. Initialize count = 0
2. If $n \geq k$, we can burn k candles to get a new one
   (a) $n = n - k$; Burn k candles
   (b) $n = n + 1$; Roll a new candle
   (c) $count = count + k$; Count number of candles burned
3. Repeat step 2 until $n < k$.
4. Burn the remaining n candles.
   (a) $count = count + n$
   (b) $n = n - n$
5. The number of candles burned is the value in $count$.

Loop Invariant

An invariant is a point in the program where the state is meaningful. The values in the variables in our state should reflect the meaning as stated above.

When transforming the state, there is usually a stage where the state is "in transition". In our algorithm, that would be steps 2(a) to 2(c), and steps 4(a) and 4(b).

At the main steps, 2 through 5, we can check that the invariant holds true, i.e., n reflects the number of unburned candles, and $count$ records the total number of candles burned. If this is true, then we can be certain that our algorithm will always produce the correct result.

Our Code

Now it is time to write the C/C++ code. With the algorithm formerly described, it is a simple task to translate it to C/C++ code.

```
int candles(int n, int k) {
    int count = 0;          // 1. Initialise count
    while (n >= k) {        // 2. Loop if n >= k
```

```
        n = n - k;          //   a. Burn k candles
        n = n + 1;          //   b. Roll a new candle
        count = count + k;  // c. Increase number of candles burned
    }
    count = count + n;      // 4a.
    n = n - n;              // 4b.
    return count;           // 5
}
```

As you can see, a lot of thinking, planning and strategizing is done before we even think about writing code. This is true especially when dealing with large and complex problems.

For small problems like this, experienced programmers often do all these mentally in their minds and can be seen just typing out the code. They are not skipping these steps but because they are so experienced, they will do it subconsciously in an instant and just write the code.

Since you are not at that level yet, you should take time to properly plan and structure your approach to solve the problem. Having a vague idea before writing code is not enough. You will end up not knowing why your code fails, or if you are lucky why it even works. And even if it works, how can you be sure it works for all inputs?

> A woodsman was once asked, "What would you do if you had just five minutes to chop down a tree?"
>
> He answered, "I would spend the first three minutes sharpening my axe."

Few minutes of sharpening your axe will save your hours on the job. Few minutes spent planning and careful thought will save you hours on coding.

Hints for Problem Set 1

Task 3: Time Elapsed

How would you solve the following math equation?

$$\frac{2}{3} + \frac{3}{4} + \frac{6}{7} =$$

The strategy taught in high school is to find the common denominator and convert each fraction to that denominator, after which, the addition becomes trivial.

Task 3 could be solved without using any control statements. Just like with fractions, there is a common denominator that we can covert to and do elementary addition.

Testing

You might have submitted code on Coursemology that passed all the public test cases, but fails one or more private tests. That is because your code is supposed to work for all valid inputs. So there is one particular input that your code gives the wrong result.

Since there are 24 possible values for hour, 60 possible values for minutes and 60 for seconds, there is a total of $24^2 \times 60^2 \times 60^2 = 7,464,960,000$ possible inputs.

$$\begin{array}{ccccc} h0\ (24) & : & m0\ (60) & : & s0\ (60) \\ h1\ (24) & : & m1\ (60) & : & s1\ (60) \\ \hline (24 \times 24) & \times & (60 \times 60) & \times & (60 \times 60) \end{array}$$

Even though some combinations are invalid (since the first time is no later than the second), there are still easily over a trillion combinations. It is definitely not possible to run your program through a few trillion inputs.

Equivalence Class Testing

We can make use of this technique called Equivalence Class Testing to reduce the size of the test data. In summary, it is identifying and partitioning the different combination of test inputs into different classes, and then just testing one input from each class. You can more in detail on Wikipedia or search on the Internet.

For this task, we can reduce the test space by just examining the relation between the inputs $h0$ and $h1$, $m0$ and $m1$, and $s0$ and $s1$.

$$\begin{array}{ccccc} h0 < h1 & : & m0 < m1 & : & s0 < s1 \\ h0 = h1 & : & m0 = m1 & : & s0 = s1 \\ h0 > h1 & : & m0 > m1 & : & s0 > s1 \\ \hline 3 & \times & 3 & \times & 3 \end{array}$$

Since there are only 3 different relation classes for each parameter hour, min and sec, there are now only $3 \times 3 \times 3 = 27$ combinations. Some combination are invalid inputs so the actual number of test combination is 14.

1. Can you list down all possible classes of test inputs for this task?
2. Use these cases to test your program.

Task 4: IP Address

For this problem set, there is no need to use control statements to solve the problem, though you are free to try.

Examine the question closely. Note that the input will be an integer of at most 8 digits (without the leading 0). The formula is fixed and operates on each digit. So what remains is determine what is the value of each digit.

Using an example, 1010,

1. how can we get the value of the right most digit? In elementary school we call that the ones place.
2. how about the second digit from the right? The tens place.
3. how about the third (hundreds)? the fourth (thousands)?

As with every task in this problem set, your solution can simply be a single formula.