TIC1001—Introduction to Computing and Programming
National University of Singapore

Tutorial 3 - Functional Abstraction

1.In tutorial 1, we performed a trace of a program containing only the main function. Extend your mental model to include traces of program execution involving more than one function using the program below.

In [ ]:
```c
#include <stdio.h>

int f(int x, int y);

int main(void)
{
    int x = 3, y = 4;
    x = f(x,y);
    y = f(x, f(y,x));
    printf("x = %d; y = %d\n", x, y); return 0;
}

int f(int x, int y)
{
    return x*10 + y;
}
```

Keep in mind the following notions while you trace.

- function call with evaluated arguments
- function activation with parameter declaration • pass-by-value
- lexical scoping
- function termination upon return

x = 34; y = 414

2.The basis representation theorem states the following: Every $n \in Z^+$ can be uniquely expressed as a sum of terms (a_i) such that

$$n = \sum_{i=1}^{k} a_i b^i, a_k != 0$$

where b is the base of the number.

For example, 130_{10} in base 10 can be expressed as 1010_5 in base 5 since

$$1 \times 10^2 + 3 \times 10^1 + 0 \times 10^0 = 1 \times 5^3 + 0 \times 5^2 + 1 \times 5^1 + 0 \times 5^0$$

(a) Implement a function void print_10_to_b(int n, int b) which takes in a number n in decimal (base 10) and prints the equivalent number in base b, for 2 ≤ b ≤ 10.
(b) Implementafunction void print_b1_to_b2(int n, int b1, int b2) thattakesin a number n in base b1 and prints the equivalent number in base b2, with 2 ≤ b1, b2 ≤ 10.

In [ ]:
```c
int print_10_to_b(int n, int b)
{
    //Convert number in base b to base 10
    //Array for converted numbers
    int convert[50];
    int i = 0;

    //Loop until n is 0
    while ( n != 0)
    {
        //Fill the ith location in array (Starts from 0) with remainder of n /
 base
        //Essentially the last digit of n
        convert[i] = n % b;
        //Removing the last digit of n in every iteration
        n = n / b;
        i ++ ;

        /*
        ----- Start of loop -----

        i = 0
        convert[0] = 130 % 5 = 0
        n = 130 / 5 = 26

        i = 1
        convert[1] = 26 % 5 = 1
        n = 26 / 5 = 5

        i = 2
        convert[2] = 5 % 5 = 0
        n = 5 / 5 = 1

        i = 3
        convert[3] = 1 % 5 = 1
        n = 1 / 5 = 0

        i = 4
        ----- End of loop -----

        Values in array --> convert[] = {0,1,0,1}

        */
    }
    //Going back one position as this position in array --> convert[4] (empty)
 is not relevant
    // i = 4 - 1, so i goes back to 3 which is convert[3]
    i--;

    //Start reverse looping from previous position of array (convert[3]), until
i is < 0
    for( i = i ; i >= 0; i--)
    {
            printf("%d",convert[i]);

        /*

        ----- Start of loop -----
        i = 3, return convert[3], prints 1
        i = 2, return convert[2], prints 0
        i = 1, return convert[1], prints 1
        i = 0, return convert[0], prints 0
        ----- End of loop -----

        Result of print_10_to_b(130, 5) --> 1010

        */
```

```
    }
    printf("\n");
    return 0;
}

int print_b1_to_b2(int n, int b1, int b2)
{
    // Convert n from base b1 to base 10,
    // then sub returned values (sum) to <print_10_to_b> function
    int i = 0, sum = 0;

    while ( n!= 0)
    {
        // Similar to <print_10_to_b> function
        int remainder = n % 10;
        n = n / 10;
        //Each digit * base^corresponding power
        sum += remainder * pow(b1,i);
        i++ ;

        /*
        Computation steps i.e.
        Convert 130 base 5 to base 10

        ----- Start of loop -----
        i = 0
        remainder = 130 % 10 = 0
        n = 130 / 10 = 13
        sum = 0 * 5^0

        i = 1
        remainder = 13 % 10 = 3
        N = 13 / 10 =1
        sum = 0 * 5^0 + 3 * 5^1

        i = 2
        remainder = 1 % 10 = 1
        n = 1 / 10 = 0
        sum = 0 * 5^0 + 3 * 5^1+ 1 * 5^2
        ----- End of loop -----

        */
    }
    print_10_to_b(sum,b2);
    return 0;
}
```

3.A function can be viewed as a black box. All you need to know are the arguments it takes as input and what its output is.

(a) One way of calculating the area of a triangle is using the formula area $= \frac{1}{2} \times base \times height$.

Implement a function `area1` that calculates and returns the area of any given triangle using this formula.
Decide what arguments the function requires as input.

In [ ]:
```
double area1(double a, double b)
{
    double area1 = 0.5 * a * b;
    return area1;
}
```

(b) Another way of calculating the area of a triangle with sides A, B, C is using the trigonometric ratio sine to get area $= \frac{1}{2} \times A \times B \times sin(AB)$., where AB is the included angle between sides A and B.

The sin function is provided by the `math.h` library. You can call it by using sin after including the line `#include < math. h>` at the top of your source file.

Implement a function `area2` that calculates and returns the area of any given triangle using this formula.
Decide what arguments the function requires as input.

In [ ]:
```
double area2(double a, double b,double c)
{
    double area2 = 0.5 * a * b * sin(c);
    return area2;
}
```

(c) We can also calculate the area of triangle using Heron's Formula,

$area = \sqrt{s(s-a)(s-b)(s-c))}$ where $s = \frac{a+b+c}{2}$

Implement a function `area3` that calculates and returns the area of any given triangle using Heron's Formula.

Decide what arguments the function requires as input.

In [ ]:
```
double area3(double a, double b, double c)
{
    double s = (a + b + c) / 2;
    double area3 = sqrt(s * (s -a)*(s - b)*(s - c));
    return area3;
}
```

4.Implement a function that takes in three integers, and rearranges them in ascending order. For example, after the following code snippet is executed:

In [ ]:
```
int a = 5, b = 7, c = 2;
sort(&a, &b, &c);
printf("%d, %d, %d", a, b, c);
```

the output will be 2, 5, 7 .

Assume that an in-place `swap` function that takes in two variables via pointers and swap their values is available (e.g. as shown in lecture). Try implementing your sort function without declaring any new variables.

In [ ]:
```
void sort(int *a, int *b, int *c)
{
   //Assign a, b, c into number[] as pointers
   //E.g. *number[0] = {a}, *number[1] = {b}, *number[2] = {c}
   int *number[] = {a, b, c};

   //Loop 1: to loop through all numbers
   //If there are 3 numbers, loop 3 times
   for (int i = 0; i < 3; i++)
   {
       //Loop 2: to swap numbers
       //If there are 3 numbers, loop 2 times
       //Since we only have to compare and swap 2 times
       //e.g. compare number[0] & number[1],
       //then compare number[1] & number[2]
       for (int j = 1; j < 3; j++)
       {
           //If *number[0] > *number[1],
           //swap in order to move smaller number to the front
           if (*number[j - 1] > *number[j])
           {
               swap(number[j - 1],number[j]);
           }
           //Else, proceed to j + 1,
           //compare *number[1] & *number[2]
       }
       //Next, proceed to i + 1 to check again
   }
}
```

5.For this question, we will use an integer to represent the amount of money in a bank account. Implement the following functions:

- `void deposit(int *account, int amount)` that takes in the bank account and an amount of money, and increments the account by the amount.
- `bool withdrawl(int *account, int amount)` that takes in the bank account and an amount of money, and decrements the account by the amount if there is sufficient money in the account. It returns true if the account was successfully decremented and false if not.
- `bool transfer(int *from, int *to, int amount)` that takes in two bank accounts and an amount of money, and transfers the amount from the first account to the second, if there is sufficient money in the first account. It returns true if the ac- count was successfully decremented and false if not.

The above functions use pass-by-pointer. You can also use pass-by-reference in C++ to achieve the same result. How will your code differ between these two methods?

As a practice on functional abstraction, you should reuse functions as much as possible.

In [ ]:
```
void deposit(int *account, int amount)
{
    *account += amount;
}

bool withdrawl(int *account, int amount)
{
    if (*account < amount)
    {
        return false;
    }
    else
    {
        *account -= amount;
        return true;
    }
}

bool transfer(int *from, int *to, int amount)
{
    if (withdrawl(from,amount))
    {
        deposit(to,amount);
        return true;
    }
    return false;
}
```

- END -