TIC1001—Introduction to Computing and Programming
National University of Singapore

Lecture 4: Functional Abstraction and Pointers

Q1 Which of the following are valid function definitions? Select all that apply.

In [ ]:
```
Correct
void foo(void) {
    return;
}

Correct
int foo(int a, int b) {
    return a + a;
}

Wrong
int foo(a, b) {
    return a + b;
}

Wrong
foo(int a, int b) {
    return a + b;
}

Correct
void foo(int a, int b, int c) {
    a = b + c;
}

Correct
void foo(int *a, int b, int c) {
    *a = b + c;
}
```

A function without any parameters or return value is still valid.
A function need not use every parameter passed into it.
Void function do not need a return statement.
Pointers can be used to modify the arguments.

Q2 What is the output for the following code snippet?

In [ ]:
```
int square(int x) {
    return x*x;
}

int twice(int x) {
    return x + x;
}

printf("%d", twice(square(2)) - square(twice(2)));
```

-8

Q3 Using the square and mean functions provided, define a new function called variance that calculates the variance of two integers. Reuse the square and mean functions in your answer!

The variance formula is as below:

$Var(X) = E[X^2] - (E[X])^2$

$E[X]$ refers to the mean of numbers (2 of them for this question) and $E[X^2]$ refers to the mean of the numbers squared.

For example variance(1, 5) and variance(5, 1) should both return 4.0

In [ ]:
```
double square(double x)
{
    return x * x;
}

double mean(int a, int b)
{
    return (a+b)/2.0;
}

double variance(int a, int b)
{
    double var;
    var = mean(square(a),square(b)) - square(mean(a,b));
    return var;
}
```

Q4 The function `int area_rect(int length, int breath)` returns the area of a rectangle of given length and breadth.

Define a function `int area_square(int x)` that:

Returns the area of a square of length x. The `area_square` function should make use of the `area_rect` function. Thus, you do not need to know how the area of a rectangular is calculated. Returns 0 if the input parameter is <= 0.

Note: Do not define the area_rect function. It is already provided. You simply have to call the function.

In [ ]:
```
int area_square(int x)
{
    if ( x <= 0)
    {
        return 0;
    }
    else
    {
        return area_rect(x,x);
    }
}
```

Q5 Note: In math, 0 is considered to be neither a positive number nor negative number.

You are to implement the following three functions with the following specifications:

1. `bool is_odd(int x)` returns `true` if the input parameter x is odd, `false` otherwise.
2. `bool is_negative(int x)` returns `true` if the input parameter x is negative, `false` otherwise.
3. `bool is_even_and_positive(int x)` returns `true` if the input parameter x is even AND positive, `false` otherwise.

Bear in mind that you should try to reuse functions whenever possible!

In [ ]:
```c
#include <stdbool.h>
bool is_odd(int x) {
    if ( x == 0)
    {
        return false;
    }
    else if ((x % 2) != 0)
    {
        return true;
    }

    else
    {
        return false;
    }
}
bool is_negative(int x) {
    if ( x < 0 )
    {
        return true;
    }
    else
    {
        return false;
    }
}
bool is_even_and_positive(int x) {
    if ( x == 0)
    {
        return false;
    }
    if (is_odd(x) == false && is_negative(x) == false)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Q6 What is the output printed by the following program?

In [ ]:
```c
void square(int *x)
{
    *x = (*x + 1) * (*x);
}

int main(void)
{
    int num = 10;
    square(&num);
    printf("%d", num);
    return 0;
}
```

110.
Address of num is passed into pointer x, so 11*10 is stored back in to num.

Q7 Bump

Consider the following code snippet:

In [ ]:
```c
int i = 0;

bump(&i);
printf("i is %d\n", i);

bump(&i);
bump(&i);
printf("i is %d\n", i);
```

Implement the function bump, such that the printed output of the code is:

i is 1
i is 3

In [ ]:
```c
int bump (int *a)
{
    *a = *a + 1;
    return *a;
}
```

Q8 A safe is typically used to secure valuable objects against theft or damage. Most safes use a spinning lock to unlock the safe by requiring the dial to be spun a certain number of revolutions clockwise or anti-clockwise. The diagram below shows a dial with 100 graduations ranging from 0 to 99, and it is currently in position 15.

We can model the position of the dial as an integer. Implement a function `void spin(int *dial, int spin)` , that takes as input the dial and the amount to spin, and repositions the dial according to the amount to spin. A positive spin amount would mean spinning it anti-clockwise while a negative amount would be clockwise. You can assume the dial has 100 positions from 0 to 99 as shown above.

For example, if the current position of the dial is at position 55, spinning it by 27 would result it to be at position 82. Spinning it again by -22 would set it to position 60.

In [ ]:
```c
void spin(int *dial, int amount)
{
    *dial = *dial + amount;
    if (*dial > 99)
    {
        int mult = *dial / 100;
        *dial = *dial - (100 * mult);
    }
    else if (*dial < 0 && *dial >= -99)
    {
        *dial = *dial + 100;
    }
    else if (*dial < -99)
    {
        int mult = -(*dial) / 100;
        *dial = *dial + (100 * (mult+1));
    }
    else
    {
        *dial = *dial;
    }
}
```

-END-