CS2211a Lab No. 7
Introduction to C

Tuesday November 10, 2015 (sections 3 and 2),
Wednesday November 11, 2015 (sections 6 & 8), and
Thursday November 12, 2015 (sections 9 and 5)

Location: ***MC10*** lab

The objective of this lab is to practice:

- Various C data types, arrays, `rand` function, `clock` function, and *recursion* in C
- Variables scope
- Pointers

If you would like to leave, and at least 30 minutes have passed, raise your hand and wait for the TA.
Show the TA what you did. If, and only if, you did a reasonable effort during the lab, he/she will give you the lab mark.

==========

1. Write, compile, and run a program that computers the sum

$$s = \sum_{i=1}^{10000} x \qquad \text{where } x = 0.1.$$

   Compute this sum twice:
   - One time with *s* and *x* variables declared as `float`
   - One time with *s* and *x* variables declared as `double`

   Calculate the error in each sum (i.e., *1000.0 – s*).
   Did you find any difference between the two results? Try to find a reasonable justification.
   You may wish to use these declarations:

```
float  sf = 0.0f, xf = 0.1f;
double sd = 0.0, xd = 0.1;
```

2. Write, compile, and run the following program. Did you find the result reasonable? How can we fix this program?

```
#include <stdio.h>
int main(void)
{ float a = 87654321.0f, b;
  b = a + 1;
  printf("a %c= a + 1\n", a == b ? '=' : '!');
  return 0;
}
```

3. Write, compile, and run the following program. Did you find the result reasonable? How can we fix this program?

```
#include <stdio.h>
int main(void)
{ float a;
  long int i = 87654321, j;
  a = i;
  j = a;
  printf("i = %ld, j = %ld, a = %f\n", i, j, a);
  return 0;
}
```

4. Write and compile the following program. Run the program many times. Did you get the same results? How to make `rand()` function to provide various outputs each time you run it.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{ printf("%d \n", rand());
  printf("%d \n", rand());
  printf("%d \n", rand());
  return 0;
}
```

5. Predict the following values:
   - `sizeof(int)`, `sizeof(short)`, `sizeof(long)`, `sizeof(2)`, and `sizeof(87654321)`
   - `sizeof(float)`, `sizeof(double)`, `sizeof(long double)`, `sizeof(2.0f)`, and `sizeof(2.0)`
   - `sizeof(char)`, and `sizeof('a')`.

   Write a program that prints all these values. Compare the printed values with your predictions. If there is any mismatch, you should justify it.

6. Study the following program:

```
#include <stdio.h>
#define SIZE 10

int main(void)
{
  int a[SIZE] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37};
  int pass; // passes counter
  size_t i; // comparisons counter
  int hold; // temporary location used to swap array elements

  printf("Data items in original order\n");
  // output original array
  for(i = 0; i < SIZE; ++i)
    printf("%4d", a[i]);

  printf("\n");

  // bubble sort
  // loop to control number of passes
  for(pass = 1; pass < SIZE; ++pass)
     // loop to control number of comparisons per pass
     for(i = 0; i < SIZE -1; ++i)
        // compare adjacent elements
        if(a[i] > a[i+1])
        { // swap a[i] and a[i+1]
          hold   = a[i];
          a[i]   = a[i+1];
          a[i+1] = hold;
        }

  printf("Data items in ascending order\n");
  // output sorted array
  for(i = 0; i < SIZE; ++i)
    printf("%4d", a[i]);

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

   (a) After the first pass, the largest number is guaranteed to be in the highest-number element of the array; after the second pass, the two highest numbers are "in place" and so on. Instead of making nine comparisons on every pass, modify the *bubble sort* to make eight comparisons on the second pass, seven on the third pass and so.

   (b) The data in the array may already be in the proper or near-proper order, so why make nine passes if fewer will suffice? Modify the sort to check at the end of each pass whether any swaps have been made. If none has been made, then the data must already be in the proper order, so the program should terminate. If swaps have been made, then at least one more pass is needed.

7. A recursive function is a function that calls itself. The speed of a recursive program is usually slower than non-recursive programs because of the stack overheads. The following program recursively calculates one number in the Fibonacci sequence. The *first* number of the Fibonacci sequence is 0, the *second* number is 1, and each subsequent number is equal to the sum of the previous two numbers of the sequence, yielding the sequence 0, 1, 1, 2, 3, 5, 8, etc.
   - Type, compile, and run the following program. The program repetitively calculates one number in the Fibonacci sequence for `counter` times. At the end, it prints the calculated Fibonacci number and the execution time.
   - **Read "`man -s 3C clock`" to learn more about the `clock()` function.**
   - Re-write `Fibonacci` function in a non-recursive fashion. Re-compile and re-run your program.
   - Compare the results and the execution time in the two programs.
   - Test your program for `Fibonacci(44), Fibonacci(45), Fibonacci(46)` and `Fibonacci(47)`. Explain why the `Fibonacci(47)` results is not consistence with the sequence.

```
#include <stdio.h>
#include <time.h>

long fibonacci(long n)
{  if (n == 0 || n == 1)
      return n;
   else
      return fibonacci(n - 1) + fibonacci(n - 2);
}

int main()
{  long result, number;
   long counter;
   clock_t start, end;

   printf("Enter an integer number: ");
   scanf( "%ld", &number );

   printf("Enter repeatitions: ");
   scanf( "%ld", &counter );

   start = clock();

   while(counter-- > 0)
     result = fibonacci(number);

   end = clock();

   printf("Fibonacci(%ld) = %ld\n", number, result);
   printf("Calculation time %f\n", ((double) (end - start))
                                  / CLOCKS_PER_SEC);

   return 0;
}
```

8. Trace the execution of the following function by hand.
   What does the function do?
   Write a program that calls the function, passing it a number entered by the user.

```
void pb(int n)
{
  if (n != 0)
  { pb(n / 2);
    putchar('0' + n%2);
  }
}
```

9. What is the output of the following program? ***Hint: You should consider the scope of each variable first. You should decide on the correct responses before running the code to test the real result.***

```
#include <stdio.h>

/*---------------------------------*/
void print_variable(int i);

void print_variable_2(void);

int f1(int i);

void f2(void);
/*---------------------------------*/

int i = 0;

/*---------------------------------*/

int main(void)
{
  print_variable_2();

  f2();

  i=f1(i);
  i=f1(50);

  print_variable(i);

  return 0;
}

/*---------------------------------*/

void print_variable(int i)
{
  printf("%d\n", i++);
}

/*---------------------------------*/


void print_variable_2(void)
{
  printf("%d\n", i++);
}

/*---------------------------------*/

int f1(int i)
{
  printf("Begining of f1\n");
  print_variable_2();
  {
    int i = 100;
```

```
    print_variable(i);
  }

  print_variable(i);

  printf("Ending of f1\n\n");

  return ++i;
}

/*---------------------------------*/

void f2(void)
{
  printf("Begining of f2\n");
  print_variable_2();
  {
    int i = 200;
    print_variable(i);
  }

  print_variable(i);

  printf("Ending of f2\n\n");
}

/*---------------------------------*/
```

In the following questions, you should decide on the correct responses before running the code to test the real result.

10.
```
int a, *p;
a = 2;
p = &a;
a = a + 1;
printf("%d\n", *p);
```
a) 2 b) 3 c) Won't Run d) Random Garbage

11.
```
int a, *p;
a = 2;
p = a;
a = a + 2;
printf("%d\n", *p);
```
a) 2 b) 4 c) Won't Run d) Random Garbage

12.
```
int a, b, *p;
p = &a;
*p = 4;
p = &b;
*p = 3;
printf("%d %d\n", a, b);
```
a) 4 3 b) 3 3 c) Won't Run d) Random Garbage

13.
```
int a, b, *p, *q;
a = 3;
p = &a;
q = p;
*q = *q + 5;
printf("%d\n", *p);
```
a) 8 b) 3 c) Won't Run d) Random Garbage

14.
```
int a;
int *p;
a = 4;
p = &a;
printf("%d\n", (*p)/a);
```
a) 1 b) 4 c) Won't Run d) Random Garbage

15.
```
void f1(int *p) {(*p) = (*p) * 2;}
int main()
{
int a = 5;
f1(&a);
printf("%d\n", a);
}
```
a) 5 b) 10 c) Won't Run d) Random Garbage

16.
```
int a = 3, b = 4;
int *p, *q;
p = &a;
q = p;
*q = b;
printf("%d %d\n", *p, a);
```
a) 4 3 b) 3 4 c) 4 4 d) 3 3

17.
```
int a = 4, *p;
p = &a;
printf("%d\n", *p+1);
```
a) 4 b) 5 c) Won't Run d) Random Garbage

18.
```
int a = 4, *p;
p = &a;
printf("%d\n", *(p+1));
```
a) 4 b) 5 c) Won't Run d) Random Garbage