COMP2006 Computer Organization
Console Input & Decision Making

Console Input

The standard library <stdio.h> includes another import function **scanf** for obtaining a value from the user. The **scanf** function takes input from the standard input which is the keyboard by default.

```
scanf("%d", &int1);
```

In the statement above, the **scanf** has two arguments, **"%d"** and **&int1**. The first argument is a format control string that indicates the type of data that should be input by the user. The **%d** conversion specifier indicates that the data should be an integer. The **&int1** indicates the location in the system memory that will be used for storing the data. The ampersand (&) is the address operator in C, we will learn it in the later labs.

Compile and run the following C program:

```c
#include <stdio.h>

int main() {

    int w, h;

    printf("Please input the width and height of a rectangle: ");

    scanf("%d", &w);
    scanf("%d", &h);

    printf("The area of the rectangle is %d\n", w * h);

    return 0;
}
```

The **scanf** statements block the program and wait for the user inputs. The first integer will be stored in the variable **w**. The second integer will be stored in the variable **h**.

```
Please input the width and height of a rectangle: 15 10
The area of the rectangle is 150
```

Also, consider the following C program, we use the scanf to obtain a string and float.

```c
#include <stdio.h>

int main() {

    char product[10];
    float price;

    printf("Product name: ");
    scanf("%s", product);
    printf("Price: ");
    scanf("%f", &price);

    printf("The price of %s is $%.2f.\n", product, price);

    return 0;
}
```

Following is the sample output:

```
Product name: Apple
Price: 5.5
The price of Apple is $5.50.
```

Note that the string may include rubbish characters if the length of the character array is too short.

Decision Making

If…Then…Else

Usually, we make decisions in the program, for example, to determine if a student's grade on an exam is greater than or equal to 85, the mark "Distinction" will be printed next to the score in his/her transcript. An if statement with the condition will be used to make the decision.

```c
if (condition) {
    ...
} else {
    ...
}
```

The conditions in the if structures are formed by using the equality operators and relational operators. The relational operators.

The equality operators include equal to (==) and not equal to (!=). The relational operators include greater than (>), greater than or equal to (>=), less than (<), and less than or equal to (<=).

The following example program uses the if statements to make the decision for printing different messages:

```c
#include <stdio.h>

int main() {
    int r = 50;
    int g;

    printf("Please input your guess! ");
    scanf("%d", &g);

    if (g > r) {
        printf("Oh~ Too large!\n");
    } else if (g < r) {
        printf("Hmm~ Too small!\n");
    } else {
        printf("Wow... You win!\n");
    }

    return 0;
}
```

The following program uses the `if` structure to validate the user input:

```c
#include <stdio.h>

int main() {
    int a, b;
    int status1, status2;

    printf("Please input two integers:\n");
    status1 = scanf("%d", &a);
    status2 = scanf("%d", &b);

    if (status1 != 1 || status2 != 1)
        printf("Invalid input!\n");
    else
        printf("a + b = %d\n", a + b);

    return 0;
}
```

While Loop

A repetition structure allows the programmer to specify that an action is to be repeated while some condition remains true. Consider the following statement:

```
While the GPS is on
       show the current position
```

The statement describes the repetition that occurs while the GPS is on. The condition, “the GPS is on” may be true or false. If it is true, then the action, “show the current position” is performed. This action will be performed repeatedly while the condition remains true.

Compile and run the following C program:

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

int main() {
    srand(time(NULL));
    int random = rand() % 100;
    int guess;

    printf("Please input your guess (0 - 99)!\n");

    while(guess != random) {
        printf("> ");
        scanf("%d", &guess);

        if (guess > random) {
            printf("Oh~ Too large!\n");
        } else if (guess < random){
            printf("Hmm~ Too small!\n");
        }
    }

    printf("Wow... You win!\n");

    return 0;
}
```

The program is a number guessing game. The user needs to guess an integer in the range from 0 to 99 repeatedly while the guess is not the answer.

We use the **rand** function to generate a random integer. Avoiding to obtain the same integer in each execution, the **srand** function with the current time (**time(NULL)**) is used for initializing the random number generator.

Compile and run the following program:

```c
#include <stdio.h>

int main() {
    int n;
    int max = 0;
    int status;

    printf("FIND MAX\n");
    printf("Please input integers (non-integer to stop):\n");

    while(1) {
        status = scanf("%d", &n);

        if (status != 1)
            break;

        if (max < n)
            max = n;
    }

    printf("The maximum number is %d.\n", max);

    return 0;
}
```

The following is the sample output:

```
FIND MAX
Please input integers (non-integer to stop):
5
8
1
6
5
2
x
The maximum number is 8.
```

Break and Continue

The **break** and **continue** statements are used to alter the flow of control.

The **break** statement, when executed in a while or for structure, causes an immediate exit from that structure.

The **continue** statement, when executed in a while structure, the loop-continuation test is evaluated immediately after the continue statement is executed. When it executed in a for structure, the increment expression is executed, then the loop-continuation test is evaluated.

Exercise 1

Write a C program that prompts the user to input integers. Every time the user inputs an integer, the program prints the average of the inputted integers. If the user inputs other than integer, the program will stop.

The following is the sample output:

```
AVERAGE
Please input ten integers:
25
Average: 25.00
30
Average: 27.50
44
Average: 33.00
78
Average: 44.25
92
Average: 53.80
51
Average: 53.33
6
Average: 46.57
27
Average: 44.13
62
Average: 46.11
81
Average: 49.60
```

For Loop

The `for` repetition structure handles all the details of counter-controlled repetition automatically. The syntax of the `for` structure is as follows:

```
for (initialization; loop-continuation condition; increment expression) {
    ...
}
```

Consider the following program:

```c
#include <stdio.h>

int main() {
    int counter;

    for (counter = 1; counter <= 10; counter++) {
        printf("%d\n", counter);
    }

    return 0;
}
```

The program above operates as follows. When the for structure begins executing, the control variable **counter** is initialized to **1**. Then, the loop-continuation condition **counter <= 10** is checked. Because the initial value of **counter** is 1, the condition is satisfied, so the **printf** statement prints the value of **counter**, the control variable **counter** is then incremented by the expression **counter++**, and the loop begins again with the loop-continuation test.

Two-level Loop (a.k.a. Nested Loop)

In the previous section, we learned to iterate set of statements using the while and for loops. Here we learn to write one for loop inside other.

C language supports nesting of one loop inside another. We can define any number of loop inside another loop. We can also have any number of nesting level and put any type of loop in another type. For example, you can write a for loop inside while loop, while inside another while etc.

The following is the example of defining two-level FOR loop is:

```c
#include <stdio.h>

int main()
{
    int i, j;

    for (i = 1; i <= 10; i++)
    {

        for (j = 1; j <= 5; j++)
        {
            printf("%d\t", (i * j));
        }

        printf("\n");
    }

    return 0;
}
```

Outer loop

Inner loop

After compiling and executing the example code above, you will get the result as follows:

```
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
5       10      15      20      25
6       12      18      24      30
7       14      21      28      35
8       16      24      32      40
9       18      27      36      45
10      20      30      40      50
```

The outer loop executes the inner loop and print the new lines.

The inner loop print columns of each row.

Consider the following example code, the value of *j* is affected by *i*:

```c
#include <stdio.h>

int main()
{
    int i, j;

    for (i = 1; i <= 10; i++)
    {
        for (j = i; j <= i+5; j++)
        {
            printf("%d\t", (i * j));
        }

        printf("\n");
    }

    return 0;
}
```

The following is the output of the code above:

```
1       2       3       4       5       6
4       6       8       10      12      14
9       12      15      18      21      24
16      20      24      28      32      36
25      30      35      40      45      50
36      42      48      54      60      66
49      56      63      70      77      84
64      72      80      88      96      104
81      90      99      108     117     126
100     110     120     130     140     150
```

Exercise 2

Write a C program that prompts the user to input the principal and interest rate. Then, the program prints the amount on deposit of the years.

The amount on deposit can be computed using the following formula:

$$Amount = principal \times (1 + interestRate)^{years}$$

For example, a person invests $1000.00 in a savings account yielding 0.5 percent interest. Assume that all interest is left on deposit in the account. The amount of money in the account at the end of two years will be:

$$Amount = 1000 \times (1 + 0.005)^2 = 1010.03$$

The following is the sample output:

```
AMOUNT ON DEPOSIT
Please input the principal, rate, and no. of years:
1000
.005
10

year    Amount on deposit
   1              1005.00
   2              1010.03
   3              1015.08
   4              1020.15
   5              1025.25
   6              1030.38
   7              1035.53
   8              1040.71
   9              1045.91
  10              1051.14
```

Hint: use the standard library *<math.h>* function **pow** to calculate the power. The function **pow(x, y)** calculates the value of **x** raised to the **y**th power. For example, **4^3 → pow(4, 3)**.

Exercise 3

Write C programs using While loop or For loop to output the followings respectively:

```
*
**
***
****
*****
******
*******
```

```
*######
**#####
***####
****###
*****##
******#
*******
```

```
   *
  ***
 *****
*******
  ***
  ***
```