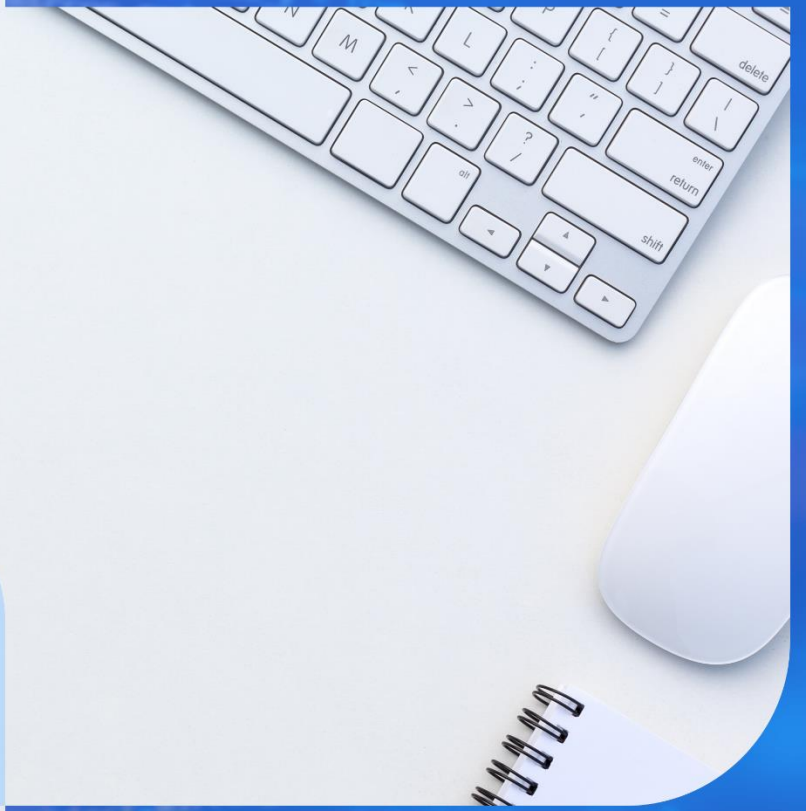
Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 6



Instructor Younghoon Kim

Random Number Generation

Rand function

SUNG KYUN KWAN UNIVERSITY(SKKU)

The rand function

- is defined in the "`stdlib.h`" header file
- generates an integer between `0` and `RAND_MAX`
  - » `RAND_MAX`: a symbolic constant defined in the <`stdlib.h`> header

Example

```
i = rand();
```

- `i` will have a random value between `0` and `RAND_MAX`
- CHECK: How can we make `i` to have a random value between 1 and 6?
  - » Hint: Use the remainder operator.

Rolling a Six-Sided Die


```
i = rand() % 100;
```

- **Generating a random number in the range of [0, 100).**
  - One typical example with the rand function
  - This technique with the remainder operator is called scaling.
    - » The number 100 is called the scaling factor.

- **Simulating a dice**
  - Use the scaling factor of 6.
  - Shift the range of numbers produced by adding 1.

```
face = rand() % 6 + 1;
```

Rolling a Six-Sided Die

DEMO

- **A simple program that prints out 20 numbers in the range of [1, 6].**
  - We can see a warning without the stdlib header.
  - Different scaling factors and shifts will be tested.

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

int main() {
    for (int i = 0; i < 20; i++)
        printf("%d ", 1 + rand() % 6);
    printf("\n");

    return 0;
}
```

Randomizing the Random Number Generator

Repeatability of rand function

- You can observe that the execution results are always the same in the previous demo.
  - The same sequence of random numbers
  - Calling rand repeatedly produces a sequence of numbers that appears to be random. (Pseudorandom numbers)

- When debugging a program, this repeatability is essential for proving that corrections to a program work properly.
  - With different sequences of random numbers, every execution will have a different result which make it very hard to correct errors.

Randomizing the sequences

- Can be accomplished with the standard library function `srand`.

srand in stdlib.h

- **srand takes a seed.**
  - Seed: an unsigned integer argument
  - A sequence of random numbers are generated based on the value of the seed.
  - With the same seed values, the random sequences will be the same.
    - » Without calling `srand`, a default value will be used.

- **`srand(time(NULL));`**
  - To randomize without entering a seed each time
  - This make the program to use the current time, which will be differed on every execution.
  - The function `time` is defined in `time.h`.

Rolling a Six-Sided Die 2

DEMO

- **A simple program that prints out 20 numbers in the range of [1, 6].**
  - Random sequence on each execution.
  - We can set a specific seed for srand function for debugging purposes.

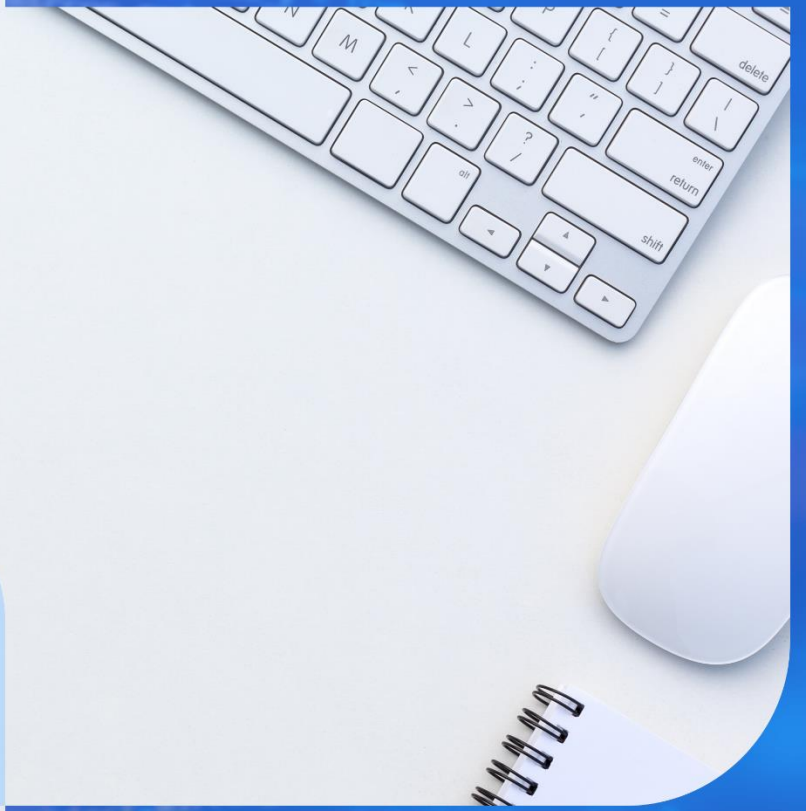
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    srand(time(NULL));
    for (int i = 0; i < 20; i++)
        printf("%d ", 1 + rand() % 6);
    printf("\n");
    return 0;
}
```

Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 6



Instructor Younghoon Kim

getchar and puts

- **`char` data type**
  - The data type for storing one character
    - » Use single quotation marks for a character
  - Occupies one byte only
    - » 8 bits. We have 256 items with the `char` data type including special characters.
  - conversion specifier: %c

```
char c = 'a';
printf("%c %c\n", c, 'a');
```

```
a a
```

Characters

- Characters can be stored in any integer data type because they're usually represented as one-byte integers in the computer.
  - Although they are normally stored in variables of type char, ...
- Thus, we can treat a character as either an integer or a character, depending on its use.
- When assigning a character literal to an integer variable, 'promotion' happens regarding the data type.
  - From 1 byte to 4 bytes

```
int c = 'a';            // OK!!!
printf("%c\n", c);      // Ok for this case because we know the value of c
                        // Some concerns about loosing data still exists.
return 0;
```

```
a
```

Characters

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
printf("The character (%c) has the value %d.\n", 'a', 'a');
```

```
The character (a) has the value 97.
```

- The integer 97 is the character's numerical representation in the computer.
- Many computers today use the **ASCII (American Standard Code for Information Interchange) character set** in which 97 represents the lowercase letter 'a'.

getchar

- The getchar function (from <stdio.h>) reads one character from the keyboard and returns as an int the character that the user entered.
  - Syntax: `int getchar ( void );`
  - `getchar()` return the character read as an unsigned char cast to an int or EOF on end of file or error.
  - Considering the promotion issue, it is recommended to assign the returned values of this function to an integer type variable.

```
int c;                  // or, 'char c;' to use the char data type.
c = getchar();          // When you use 'char', implicit type casting happens.
printf("%c\n", c);
```

Characters

DEMO

- **A simple example for characters**
  - Variations on data types will be shown.

```
#include <stdio.h>
int main() {
  int c = 'a';
  printf("%c\n", c);

  c = getchar();
  printf("The character (%c) has the value %d.\n", c, c);
  return 0;
}
```

Reading Character Input

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
#include <stdio.h>

int main()
{
  int c;
  int num_of_a = 0;

  while( (c = getchar()) != EOF ) {
    if(c == 'a') {
      num_of_a++;
    }
  }
  printf("The number of 'a' is %d.\n",
          num_of_a);

  return 0;
}
```

- This program counts the number of 'a' from the user input.
  - The parenthesized assignment (`c = getchar()`) executes first.
  - In the program, the value of the assignment `c = getchar()` is compared with the value of EOF (a symbol whose acronym stands for “end of file”).
  - We use EOF (which normally has the value -1) as the sentinel value.

Reading Character Input

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
#include <stdio.h>

int main()
{
  int c;
  int num_of_a = 0;

  while( (c = getchar()) != EOF ) {
    if(c == 'a') {
      num_of_a++;
    }
  }
  printf("The number of 'a' is %d.\n",
          num_of_a);

  return 0;
}
```

- This program counts the number of 'a' from the user input.
  - End-of-file (EOF): A system-dependent keystroke combination to mean "end of file"—i.e., “I have no more data to enter.”
  - If the value entered by the user is equal to EOF, the while loop terminates.
  - The variable c is declared as `int` because EOF has an integer value (normally -1).

Entering the EOF Indicator

- On Linux/UNIX/Mac OS X systems, the EOF indicator is entered by typing

  <Ctrl> d

  - This notation <Ctrl> d means to press the Enter key then simultaneously press both Ctrl and d.

- On other systems, such as Microsoft Windows, the EOF indicator can be entered by typing

  <Ctrl> z

- You may also need to press *Enter* on Windows.

Reading Character Input

DEMO

- **This program counts the number of 'a' from the user input.**
  - What happens when we use 'char' data type for the variable c? Is it OK?
  - Check the difference when we use '\n' Instead of EOF. (Ex: Counting the number of '\n'.)

```
#include <stdio.h>
int main() {
  int c;
  int num_of_a = 0;
  while( (c = getchar()) != EOF ) {
    if(c == 'a') { num_of_a++; }
  }
  printf("The number of 'a' is %d.\n", num_of_a);
  return 0;
}
```

puts

- Function puts takes a string as an argument and displays the string followed by a **newline character**.
  - Syntax: `int puts ( const char * str );`
  - Similar to `printf` but only for C string

```
printf("Hi, this statement is using printf\n");
puts("Hi, I'm using puts instead of printf");    // Check that puts is without \n.
```

printf("%s", "Hello\n");

- C string can be printed out using printf function with a conversion specifier '%s'.
  - %s: a conversion specifier for string data
  - You can use '%s' with C strings or variables with string data.

- String variables in C are with 'char array' data type.
  - Use squared brackets after the variable name to specify the string length.
    - They are called 'arrays' which will be dealt later.
  - The size of the string MUST be greater than the actual string length.

```
char hello[10] = "Hello";                  // A string variable
char name[10] = "Mike";
printf("%s, %s!\n", hello, name);          // Printing string variables with %s
printf("%s", "Hello\n");                   // Weird, but ok.
```

Print strings

DEMO

- A simple program that prints strings.

```
#include <stdio.h>

int main() {
  printf("Hi, this statement is using printf\n");
  puts("Hi, I'm using puts instead of printf");   // Check that puts is without \n.

  char hello[10] = "Hello";   // A string variable
  char name[10] = "Mike";
  printf("%s, %s!\n", hello, name); // Printing strings with %s
  printf("%s", "Hello\n");      // Weird, but ok.

  return 0;
}
```

Exercise: ASCII Printer for alphabets

DEMO

- **Make a program that prints ASCII values for user-entered characters.**
  - You must print ASCII values only for alphabets.
  - The program terminates when EOF is entered.

```
#include <stdio.h>

int main() {
  int c;

  /* Fill in here */

  return 0;
}
```

Exercise: ASCII Printer for characters

DEMO

- **Make a program that prints ASCII values for user-entered characters.**
  - You must print ASCII values only for characters.
  - The program terminates when EOF is entered.

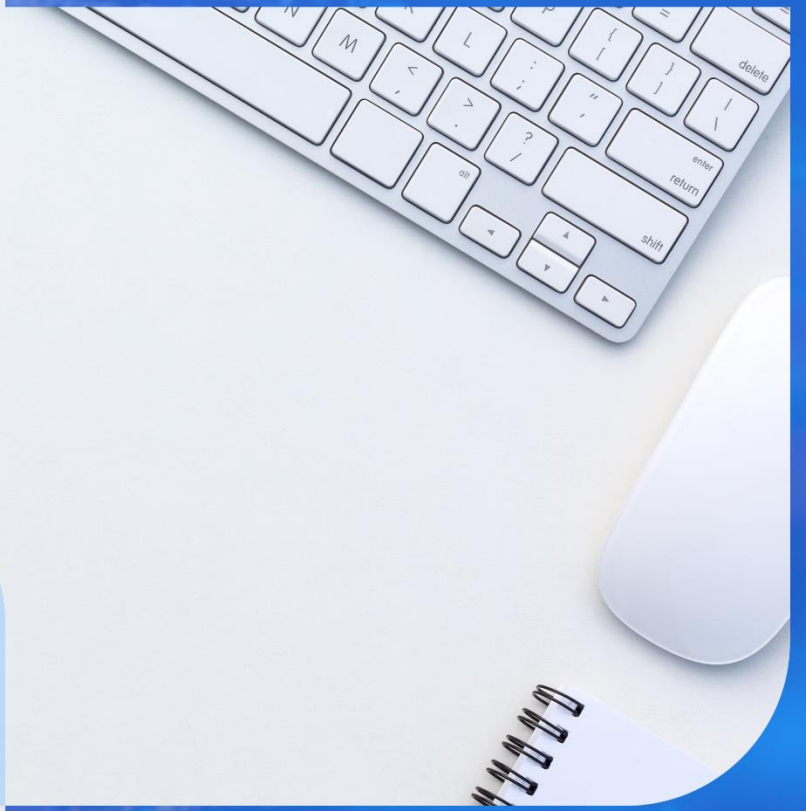
```
#include <stdio.h>


int main() {
  int c;
  while( (c = getchar()) != EOF) {
    if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
      printf("Character: %c, ASCII value: %d\n", c, c);
    }
  }
  return 0;
}
```

Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 6



Instructor Younghoon Kim

Conditional Operator

Conditional Operator

SUNG KYUN KWAN UNIVERSITY(SKKU)

- C provides the **conditional operator** (**?:**) which is closely related to the `if…else` statement.
- The conditional operator is C's only ternary operator—it takes *three* operands.
  - The first operand is a *condition*.
  - The second operand is the value for the entire conditional expression if the condition is *true*.
  - The third operand is the value for the entire conditional expression if the condition is *false*.

```
(condition)? (value for the true case):(value for the false case)
```

Conditional Operator

- For example, the puts statement

```
puts( grade >= 60 ? "Passed" : "Failed" );
```

  contains as its second argument a conditional expression that evaluates to the string "`Passed`" if the condition `grade >= 60` is true and to the string "`Failed`" if the condition is false.

- The puts statement performs in essentially the same way as the preceding `if...else` statement.

Conditional Operator

- The second and third operands in a conditional expression can also be actions to be executed.

- For example, the conditional expression

```
grade >= 60 ? puts( "Passed" ) : puts( "Failed" );
```

is read, "If `grade` is greater than or equal to `60` then puts("`Passed`"), otherwise `puts( "Failed" )`." This, too, is comparable to the preceding `if…else` statement.

Conditional Operator

DEMO

- **Conditional operator example**
  - Check that no comparison operators are used in the condition part.
  - We have three or more different ways to show the same result.

```
#include <stdio.h>
int main() {
  int num;
  scanf("%d", &num);
  num%2? printf("Odd\n"):printf("Even\n");     // or, printf(num%2? "Odd\n":"Even\n");
                                                // or, if(num % 2) { printf("Odd\n"); }
                                                       else { printf("Even\n"); }


  return 0;
}
```

Formatting Floating-Point Numbers

Formatting Floating-Point Numbers

- Sometimes we need to format floating-point numbers

```
printf( "%.2f\n", 3.446 );                    // prints 3.45
```

- The `printf` conversion specifier `%.2f` to print the value of a floating-point number can be used.
  - The f specifies that a floating-point value will be printed.
  - The .2 is the **precision** with which the value will be displayed—with 2 digits to the right of the decimal point.
  - When floating-point values are printed with precision, the printed value is **rounded** to the indicated number of decimal positions.
- The value in memory is unaltered.

More on Formatting Floating-Point Numbers

DEMO

```
printf( "%.2f\n", 3.446 );              // prints 3.45
printf( "%.1f\n", 3.446 );              // prints 3.4
printf( "%21.2f\n", amount);            // field width
printf( "%-6d\n", number);              // left justification
```

- The conversion specifier `%21.2f` is used to print the value of the variable `amount`.
  - The 21 in the conversion specifier denotes the field width in which the value will be printed.
  - The 2 specifies the precision (i.e., the number of decimal positions).

- Right or left justification
  - Automatic right justification is done with a smaller number of characters than the field width.
  - To left justify a value in a field, place a - (minus sign) between the % and the field width.
    - The minus sign may also be used to left justify integers (such as in %-6d) and character strings (such as in %-8s).

Exercise: Printing Factorials of odd numbers in Tabular Form

DEMO

- **Factorial of odd numbers**
  - Print the results as shown below. Factorials are printed out at the seventh position on each line regardless of the number of i's digits.

- Input: 4

```
i      !
1      1
3      6
```

- Input: 12

```
i      !
1      1
3      6
5      120
7      5040
9      362880
11     39916800
```

Exercise: Printing Factorials of odd numbers in Tabular Form

DEMO

- Skeleton Code

```
#include <stdio.h>

int main() {
  int input;
  scanf("%d",&input);
  printf("i     !\n");

  for(int i = 1; i <= input; i++) {
    if(i % 2 != 0) {
      int temp = 1;
      for(int j = 1; j <= i; j++) {
        temp *= j;
      }
      printf("%d", i);
      if(i >= 10) {
        printf("    ");
      } else {
        printf("     ");
      }
      printf("%d\n", temp);
    }
  }
  return 0;
}
```

Exercise: Printing Factorials of odd numbers in Tabular Form

DEMO

- Solution

```
#include <stdio.h>


int main() {
  int input;
  scanf("%d",&input);
  printf("i     !\n");

  for(int i = 1; i <= input; i++) {
    if(i % 2 != 0) {
      int temp = 1;
      for(int j = 1; j <= i; j++) {
        temp *= j;
      }
      printf("%-6d%d\n", i, temp);
    }
  }

  return 0;
}
```