Programming Fundamentals

Module E - Standard Libraries

Le The Anh
anhlt161@fe.edu.vn

FPT UNIVERSITY

Objectives

1. Input and Validation
   - Types of Input
   - getchar
   - scanf
   - Validation
   - In-Class Practice
2. Formatted Output
   - putchar
   - printf
   - Exercises
3. Standard Libraries - Library Functions
   - Standard
   - Time
   - In-Class Practice
   - Math
   - Character

Types of Input

- Input to a program may be unbuffered or buffered
- Interactive program uses unbuffered input. The program can respond to each and every keystroke directly.
- Buffered input enables data editing before submission to a program. That is, the program accepts one complete input record at a time rather than one keystroke at a time.
- A buffer is a region of memory that collects and holds data temporarily.

Buffered Input

- To transfer the contents of a buffer to a program the user must press the `\n` character (or fill the buffer completely).
- Two C functions provide buffered input facilities on the standard input stream: getchar, scanf

getchar

- getchar retrieves a single character from the standard input stream buffer without translating the input. The prototype for getchar is

```
int getchar (void);
```

- getchar returns eitherthe character code for the retrieved character or `EOF` (EOF=-1, ctrl+z in Windows, ctrl+d in Unix)

getchar

Clearing the Buffer and Pausing Execution

- To synchronize user input with program execution, we clear the input buffer of any characters that the user has entered and have remained in the buffer.

```
void clear (void)
{
    while (getchar() != '\n');
}
```

- getchar function can be used to pause the execution of a program at a certain point

```
printf("Press enter to continue ...");
while (getchar() != '\n');
```

scanf

- `scanf` retrieves data values from the standard input stream buffer under format control.

  `scanf(format string, &identifier, ...)`
- `scanf` extracts data values from the standard input stream buffer until `scanf` has
  - interpreted and processed the entire format string,
  - found a character that does not meet the next conversion specification in the format string, in which case `scanf` leaves the offending character in the buffer, or
  - emptied the buffer, in which case `scanf` waits until the user adds more data values

scanf

- Conversion specifiers

Specifier	Input Value	Use With
%c	character	char
%d	decimal	char, int, short, long, long long
%u	decimal	unsigned int, char, short, long, long long
%o	octal	unsigned int, char, short, long, long long
%x	hexadecimal	unsigned int, char, short, long, long long
%f	floating-point	float, double, long double

- Example

```
char c;
long n;
scanf("%d%x", &c, &n); // 98 a1
printf("%c %ld", c, n); // b 161
```

- `scanf` treats the whitespace between the input values as a separator. There is no need to place a blank character between the conversion specifiers.

scanf

- Several ways to handle unprocessed \n's:

```
scanf("%d", &items);
scanf("%*c%c", &tax);    /* swallow one character first */

scanf("%d", &items);
scanf(" %c", &tax); /* skip all whitespace first */

scanf("%d%*c", &items); /* swallow newline */
scanf("%c", &tax);

scanf("%d", &items);
clear();     /* clear the buffer */
scanf("%c", &tax);

scanf("%3d", &n); // keep only 3 digits
printf("%d", n);
```

scanf

- scanf returns the number of addresses successfully filled or EOF. A return value of
  - 0 indicates that scanf did not fill any address,
  - 1 indicates that scanf filled the first address successfully,
  - 2 indicates that scanf filled the first and second addresses successfully,
  - ...
  - EOF indicates that scanf did not fill any address and encountered an end of data character.
- The return code from scanf does not reflect success of %* conversions

Validation

- We cannot predict how the user will input the data values: whether the user will enter them as requested or not. One user may make a mistake. Another user may simply try to break the program.
- We write the program so that it traps all erroneous input, which includes:
  - invalid characters
  - trailing characters
  - out-of-range input
  - incorrect number of input fields

Validation

```
int getInt(int min, int max) {
    int value, keeptrying = 1, rc;
    char after;

    do {
        printf("Enter a whole number in the range [%d,%d]: ",
               min, max);
        rc = scanf("%d%c", &value, &after);
        if (rc == 0) {
            printf("**No input accepted!**\n\n");
            clear();
        } else if (after != '\n') {
            printf("**Trailing characters!**\n\n");
            clear();
        } else if (value < min || value > max) {
            printf("**Out of range!**\n\n");
        } else
            keeptrying = 0;
    } while (keeptrying == 1);

    return value;
}

void clear (void) {

    while (getchar() != '\n');
}
```

Validation

```
Enter a whole number in the range [3,15]: we34
**No input accepted!**

Enter a whole number in the range [3,15]: 34.4
**Trailing characters!**

Enter a whole number in the range [3,15]: 345
**Out of range!**

Enter a whole number in the range [3,15]: 14
Program accepted 14
```

In-Class Practice

- Design and code a function named `get_double` that receives two double values - a lower limit and an upper limit - and returns user input that lies between the limiting values. Your function rejects any input that includes trailing characters or lies outside the specified limits.
- Write a program named `get_id` that guarantees that the input data contains exactly one capital character followed by four digits.

Summary

- Input and Validation
  - Types of Input
  - getchar
  - scanf
  - Validation
  - In-Class Practice

Q&A

Formatted Output

- Standard output is buffered. The standard output buffer empties to the standard output device whenever the buffer receives a newline character or the buffer is full.
- Buffering enables a program to continue executing without waiting for the output device to finish displaying the most recently received characters.
- The two C functions that provide output facilities for the standard output stream are `putchar` and `printf`.

putchar

- putchar writes the character received to the standard output stream buffer and returns the character written or EOF if an error occurred.
- Prototype:

```
int putchar(int);
```

- For example:

```
putchar('a');
```

printf

- printf sends data under format control to the standard output stream buffer and returns the number of characters sent.
- Prototype:

  ```
  printf(format string, value, ..., value)
  ```

  ** The format string is a literal string that consists of characters interspersed with conversion specifiers. Conversion specifier begins with a % and ends with a conversion character
- For example:

  ```
  printf("%s was %d years old", name, age);
  ```

Format String

- Between the % and the conversion character, there may be
  % `flags width . precision size conversion_character`
- `flags`
  - - align the field to the left (by default it is right aligned)
  - + print the sign of a number
  - 0 pads the field width with leading zeros.
  - # for real numbers, it inserts zeroes at the end and always print the comma; for numbers not in base 10, it adds a prefix denoting the base.
- `width` sets the minimum field width within which to display the converted value.
- `.` separates the field width from the precision.
- `precision` sets the number of digits to be printed after the decimal point for f conversions and the minimum number of digits to be printed for an integer (adding leading zeros if necessary)
- `size` identifies the size of data type of the value passed.

Conversion Specifiers

- A conversion specifier begins with a % and ends with a conversion character. The conversion character describes the data type of the value to be displayed.
- Conversion characters:

Specifier	Output As A	Use With
%c	character	char
%d	decimal	char, int, short, long, long long
%u	decimal	unsigned char, int, short, long, long long
%o	octal	char, int, short, long, long long
%x	hexadecimal	char, int, short, long, long long
%f	floating-point	float, double, long double
%g	general	float, double, long double
%e	exponential	float, double, long double

printf Examples

```
printf("|%d|\n", 123); // |123|
printf("|%5d|\n", 123); // |  123|
printf("|%-5d|\n", 123); // |123  |
printf("|%05d|\n", 123); // |00123|
printf("|%05.4d|\n", 123); // | 0123|

printf("|%x|%x|\n", 0xa1, 161);         // |a1|a1|
printf("|%#x|%#X|\n", 161, 161); // |0xa1|0XA1|

printf("|%o|%o|%#o|\n", 0241, 161, 161); // |241|241|0241|

printf("|%f|\n", 1.23456789); // |1.234568|
printf("|%.10f|\n", 1.23456789); // |1.2345678900|
printf("|%13.10f|\n", 1.23456789); // | 1.2345678900|
printf("|%#13.10f|\n", 1.23); // | 1.2300000000|

printf("|%g|%g|\n", 10000.2, 1000000.2); // |10000.2|1e+006|
printf("|%g|%g|\n", 0.0001, 0.00001); // |0.0001|1e-005|

printf("|%e|\n", 1.23); // |1.230000e+000|

printf("|%%%c|", 'c'); // |%c|
```

Exercises

- Try different flags, widths, and precisions.

Summary

- Formatted Output
  - putchar
  - printf
  - Exercises

Q&A

Library Functions

- The standard C libraries include functions to perform mathematical calculations, character analysis and character manipulation.
- The standard libraries are not part of the language proper.
- The prototypes for popular standard functions are in $<$ `stdlib.h` $>$.

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

Standard - Integer Absolute Value

- Prototypes

```
int abs(int);
long labs(long);
long long llabs(long long);
```

- Example

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

main()
{
    int a = -10;
    long b = -10;
    long long c = -10;
    printf("%d %ld %lld", abs(a), labs(b), llabs(c));
}
```

Standard - Random Number

- rand returns a pseudo-random integer in the range 0 to RAND_MAX. RAND_MAX is implementation dependent but at least 32767. The prototype is

```
int rand(void);
```

- Example

```
int a = rand();
```

- rand without srand generates the same set of random numbers for successive runs (useful for debugging).
- srand sets the seed for the random number generator. time(NULL) is used to generate a unique time-based seed for each run.

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

main()
{
    srand(time(NULL));
    printf("%d", rand());
}
```

Random Number - Examples

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


main()
{
    int i, a = 10, b = 20;
    float c = 2.5, d = 8.5;
    srand(time(NULL));

    printf("10 pseudo-random integers:\n");
    for(i=0; i<10; i++)
            printf("%d ", rand());

    // RAND_MAX: an integer constant equal to the maximum value
    // (at least 32767) returned by the function rand()
    printf("\n\nRAND_MAX: %d\n", RAND_MAX);

    printf("\n10 pseudo-random integers in range [%d, %d]:\n", a, b);
    for(i=0; i<10; i++)
        printf("%3d", a + rand() % (b - a + 1));

    printf("\n\n10 pseudo-random float numbers in range [%.1f, %.1f]:\n", c, d);
    for(i=0; i<10; i++)
        printf("%5.2f", c + (float)rand() / RAND_MAX * (d - c));
}
```

Random Number - Examples

```
10 pseudo-random integers:
4388 12443 8879 32507 9903 15670 18029 9045 17953 1955

RAND_MAX: 32767

10 pseudo-random integers in range [10, 20]:
 18 10 18 20 13 20 17 20 11 12

10 pseudo-random float numbers in range [2.5, 8.5]:
 8.30 7.34 8.17 7.86 5.22 5.95 4.00 4.13 2.81 6.85
```

In-Class Practice

- You've just finished your first software, and you are thinking of commercializing it. The first step is to write a key generator that takes an integer `n` as input and generates `n` random keys consisting of four groups of four capital letters. Each group is separated by -. For example, `VPYV-IKMU-ATBW-WFWJ`.

In-Class Practice

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


main()
{
    int i, j, n, A = (int)'A', Z = (int)'Z';

    srand(time(NULL));

    printf("Enter n: ");
    scanf("%d", &n);

    for(i=0; i<n; i++)
    {
        printf("\n");

        for(j=1; j<=19; j++)
            if (j%5 == 0)
                printf("-");
            else
                printf("%c", A + rand() % (Z - A + 1));
    }
}
```

time

- `time_t time(time_t *second)` returns current time in seconds since 00:00:00 UTC, January 1, 1970. If second is not a null pointer, the returned value is also stored in the object pointed to by second.

```
time_t seconds;

seconds = time(NULL);
printf("Seconds since Jan 1 1970: %ld\n", seconds);

time(&seconds);
printf("Seconds since Jan 1 1970: %ld\n", seconds);

/**********************************
Seconds since Jan 1 1970: 1630140489
Seconds since Jan 1 1970: 1630140489
**********************************/
```

- If an error occurs, the `time` function returns `(time_t)(-1)`.

time

- `double difftime (time_t, time_t);` returns the difference in seconds between two time arguments. For example,

```c
#include <time.h>
#include <stdio.h>
#define NITER 1000

main()
{
    double x = 1;
    int i, j, k;
    time_t t0, t1;
    t0 = time(NULL);
    for (i = 0; i < NITER; i++)
        for (j = 0; j < NITER; j++)
            for (k = 0; k < NITER; k++)
                x = x * 1.0000000001;
    t1 = time(NULL);
    printf("Elapsed time is %.1lf secs\n", difftime(t1, t0));
    printf("Value of x is %.10lf\n", x);
}

// Elapsed time is 3.0 secs
// Value of x is 1.1051709272
```

clock

- `clock_t clock(void)` returns the number of clock ticks elapsed since the program was launched. To get the number of seconds used by the CPU, we need to divide by `CLOCKS_PER_SEC`.

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

int main()
{
    double x = 1;
    int i, j, k;
    time_t t0, t1;
    clock_t c0, c1;

    t0 = time(NULL);
    c0 = clock();
    for (i = 0; i < 1000; i++)
        for (j = 0; j < 1000; j++)
            for (k = 0; k < 1000; k++)
                x = x * 1.0000000001;
    t1 = time(NULL);
    c1 = clock();
    printf("Elapsed time is %.1lf secs\n", difftime(t1, t0));
    printf("Process time is %.3lf secs\n", (double)(c1-c0)/CLOCKS_PER_SEC);
}

// Elapsed time is 4.0 secs
// Process time is 3.515 secs
```

In-Class Practice

- Design and code a program named coinToss.c that
  - prompts the user for the number of times to toss a coin,
  - accepts a positive value as the number of times,
  - simulates the coin tosses for the requested number of times, and
  - reports the percentage of head results and the percentage of tail results for the coin tosses.

Math

- The prototypes for the math functions are in <math.h>.
- Floating-point absolute value:

```
double fabs(double);
float fabsf(float);
long double fabsl(long double);
```

- Floor functions return the largest integer value not greater than the value received

```
double floor(double);
float floorf(float);
long double floorl(long double);
```

- Ceiling functions return the smallest integer value not less than the value received. Their prototypes are

```
double ceil(double);
float ceilf(float);
long double ceill(long double);
```

Math

- Rounding functions return the integer value closest to the value received

```
double round(double);
float roundf(float);
long double roundl(long double);
```

- Truncating functions return the integer component of the value received

```
double trunc(double);
float truncf(float);
long double truncl(long double);
```

- Square root functions return the square root of the floating-point value received

```
double sqrt(double);
float sqrtf(float);
long double sqrtl(long double);
```

Math

- Power functions return the result of the first floating-point value received raised to the power of the second floating-point value

```
double pow(double base, double exponent);
float powf(float base, float exponent);
long double powl(long double base, long double exponent);
```

- Logarithms functions return the natural logarithm of the floating-point value received

```
double log(double);
float logf(float);
long double logl(long double);
```

- Power of e functions return the natural anti-logarithm of the floating-point value received

```
double exp(double);
float expf(float);
long double expl(long double);
```

Math - Examples

```c
#include <stdio.h>
#include <math.h>

main()
{
    double a = -1.2, b = 1.44;
    float c = -3.4;
    long double d = -5.6;

    printf("%.1lf %.1f\n", fabs(a), fabsf(c)); // 1.2 3.4
    // if this code doesn't work properly on DevCpp
    // try it on onlinegdb.com instead
    printf("%.1Lf\n", fabsl(d)); // 5.6
    printf("%.1lf\n", floor(b)); // 1.0
    printf("%.1lf\n", ceil(b)); // 2.0
    printf("%.1lf\n", round(b)); // 1.0
    printf("%.1lf\n", trunc(b)); // 1.0
    printf("%.1lf\n", sqrt(b)); // 1.2
    printf("%.1lf\n", pow(b, a)); // 0.6
    printf("%.3lf\n", log(b)); // 0.365, e = 2.718...
    printf("%.3lf\n", exp(b)); // 4.221
}
```

Character

- The prototypes for character analysis and character manipulation functions are in <ctype.h>.
- In checking for a true value, we check that the value is not false: that is, value != 0, rather than assume that true is represented by unity.

```
int islower(int);
int isupper(int);
int tolower(int);
int toupper(int);
// check if received character is alphabetic
int isalpha(int);
int isdigit(int);
// whitespace characters: ' ', '\t', '\n', '\v', '\f'
int isspace(int);
// check if received character is space or tab
int isblank(int);
```

- Example

```
printf("%c %d", toupper('a'), isspace('a')); // A 0
```

Summary

- Standard C Libraries
  - Standard
  - Time
  - In-Class Practice
  - Math
  - Character

Q&A

In-Class Practice

- Design and code a program that displays a multiplication table. Your program prompts the user for the range of integer values that the table covers and displays the table in a columnar format. The output from your program might look something like

```
Enter the low end of the range: 3
Enter the high end of the range: 7

    3    4   5   6   7
3   9   12  15  18  21
4  12   16  50  24  28
5  15   20  25  30  35
6  18   24  30  36  42
7  21   28  35  42  49
```