

# Libraries

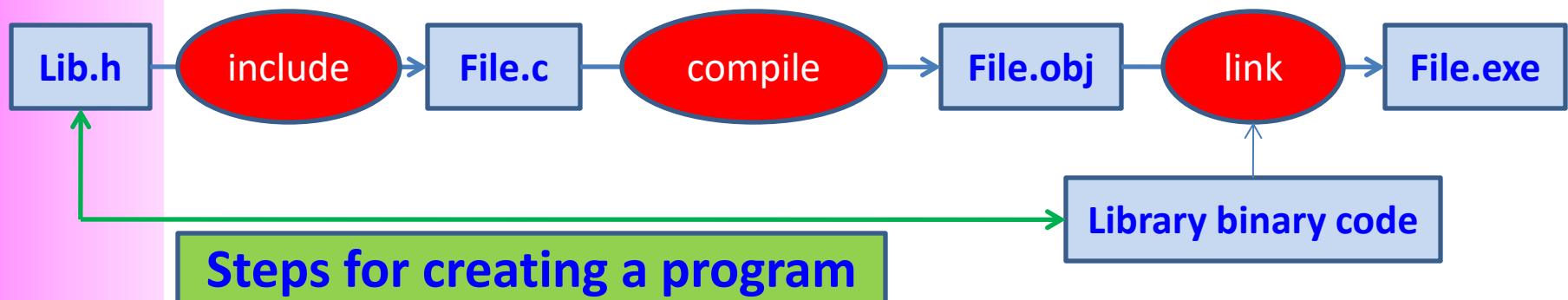
Module E

Libraries

Input and Validations

# Introduction

- Many basic tasks are very hard for programming beginners.
- The standard C libraries include functions to perform mathematical calculations, character analysis and character manipulation...
- Library header files have the extension .h
- Library binary files are implemented in files with specific filenames which are named by designers of tool suppliers (you may not know their names) and they are linked to your programs when they are compiled.



# Objectives

After studying this chapter, you should be able to use the following build-in libraries:

- Standard (stdlib.h)
- Time (time.h)
- Math (math.h)
- Character (ctype.h)

# Content

## The standard C libraries

- Standard
- Time
- Math
- Character

# 1- The Standard library: *Stdlib.h*

- You can open this file using WordPad to get more information.

## Common used functions

**int abs(int)                      long labs(long) :** Get absolute value of integral number

**int rand(void), int rand(unsigned int seed):** Get a random integer in [0.. RAND\_MAX]  
In DevC++ (RAND\_MAX=0x7FFF), start up the soft random mechanism

**Functions for dynamic memory allocating:** They are presented in the previous lecture

**void exit(int code):** Force the program terminating

**int system ( const char\* programName):** Request a program to execute

**char\* itoa (int num, char\* result, int base)      char\* ltoa (long num, char\* result, int base)**  
Convert integral number to ASCII string

**int atoi (const char\*)                      long atol (const char\*)                      double atof (const char\*)**  
Convert an ASCII string to number

# Stdlib.h...

- ***Hardware randomize***: this mechanism is supported by the hardware → Random sequence is the same in each time it performs.
- ***Software randomize***: A mechanism for generating a sequence of random number based on a mathematic algorithm using a initial number – **seed** (trị mầm) (It is presented in the subject Discrete Mathematics). So, the random sequence is different in each time it performs.

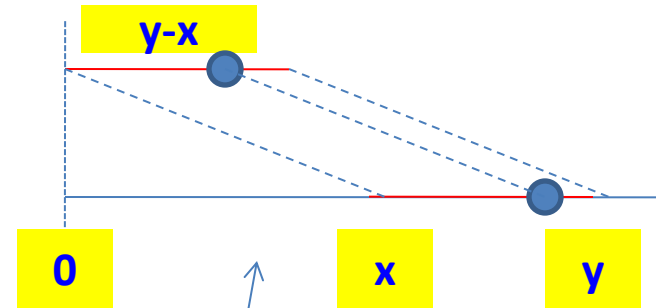
## Stdlib.h...

Copy, paste, compile and run  
this program

```

/* stdlib_demo.c */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{ int i; int a=5, b=50;
  double x=3.5, y= 20.8;
  printf("10 Hardware random integers:\n");
  for (i=0;i<10; i++) printf("%d ", rand());
  /*Init randomise using system time-milliseconds
    Each time the program executes, the system time changes.
    So, random sequence changes
  */
  srand(time(NULL));
  printf("\n\n10 Software random integers:\n");
  for (i=0;i<10; i++) printf("%d ", rand());
  printf("\n\n10 random integers between:%d...%d\n", a, b);
  for (i=0;i<10; i++) printf("%d ", a + rand()%(b-a));
  printf("\n\n5 random double between:%lf...%lf\n", x, y);
  for (i=0;i<5; i++) printf("%lf ", x + (double)rand()/RAND_MAX*(y-x));
  getchar();
}

```



$0 \leq \text{value} < 1$

## 2- The Time library: *time.h*

### Some concepts

- *Date and time information* is presented in computer using an integral number. It is usually the data type ***long***.
- A *clock tick* is the unit by which processor time is measured and is returned by 'clock'.
- **CLOCKS\_PER\_SEC**: Number of clock ticks per second. A constant is defined in *time.h* (in Dev C++, CLOCKS\_PER\_SEC = 1000 means that 1 clock tick = 1millisecond).
- 2 data type are defined in the library *time.h*:
  - **time\_t** : ( in DevC++: `typedef long time_t;`)
  - **clock\_t** : ( in DevC++: `typedef long clock_t;`)



# The library *time.h*

## Common used functions

**time\_t time ( time\_t \*tptr );** returns the current calendar time and this time is stored in it's parameter.

**double difftime ( time\_t , time\_t );** returns the difference in seconds between two calendar time arguments.

**clock\_t clock ( void );** returns the current date time information using the unit clock tick

We can use these function to evaluate time cost for an algorithm.

# The library *time.h*

```
/* stdlib_demo.c */
/* Evaluate time cost of 1000000000 mathematic operations */
#include <stdio.h>
#include <time.h>
int main()
{ int i; int n=1; double x=1.5;
  /* Use time_t data type */
  time_t t1 = time(NULL); /*Get current time */ for (i=0; i<1000000000;i++) x= x+1;
  time_t t2 = time(NULL); /*Get current time */
  double dt = difftime(t2,t1);
  printf("\nCost of 1 billion real number adding operations: %lf sec\n", dt);
  t1 = time(NULL); /*Get current time */ for (i=0; i<1000000000;i++) n= n+1;
  t2 = time(NULL); /*Get current time */ dt = difftime(t2,t1);
  printf("\nCost of 1 billion integral number adding operations: %lf sec\n", dt);
  /* Use clock_t data type */
  n=1;
  clock_t ct1= clock(); /*Get current time */ for (i=0; i<1000000000;i++) n= n+1;
  clock_t ct2= clock(); /*Get current time */
  printf("\nCost of 1 billion integral number adding operations: %ld ticks\n", ct2-ct1);
  printf("\nor %lf secs\n", ((double)(ct2-ct1)/CLOCKS_PER_SEC));
  getchar();
}
```

Copy, paste, compile and run  
this program

## 3- The Math Library: *Math.h*

- It contains function prototypes for the mathematic functions
- *Constants*:

<code>#define M_E</code>	<code>2.7182818284590452354</code>
<code>#define M_LOG2E</code>	<code>1.4426950408889634074</code>
<code>#define M_LOG10E</code>	<code>0.43429448190325182765</code>
<code>#define M_LN2</code>	<code>0.69314718055994530942</code>
<code>#define M_LN10</code>	<code>2.30258509299404568402</code>
<code>#define M_PI</code>	<code>3.14159265358979323846</code>
<code>#define M_PI_2</code>	<code>1.57079632679489661923</code>
<code>#define M_PI_4</code>	<code>0.78539816339744830962</code>
<code>#define M_1_PI</code>	<code>0.31830988618379067154</code>
<code>#define M_2_PI</code>	<code>0.63661977236758134308</code>
<code>#define M_2_SQRTPI</code>	<code>1.12837916709551257390</code>
<code>#define M_SQRT2</code>	<code>1.41421356237309504880</code>
<code>#define M_SQRT1_2</code>	<code>0.70710678118654752440</code>

# The Math Library: *Math.h...*

Common used functions	Description	Example
double <b>fabs</b> (double) float <b>fabsf</b> (float)	Absolute	
double <b>floor</b> ( double ); float <b>floorf</b> ( float );		floor(16.3) → 16.0
double <b>ceil</b> ( double ); float <b>ceilf</b> ( float );		ceil(16.3) → 17.0
double <b>round</b> ( double ); float <b>roundf</b> ( float );		round(16.3) → 16.0 round(-16.5) → -17.0
double <b>trunc</b> ( double ); float <b>truncf</b> ( float ); long double <b>truncl</b> ( long double );	truncate	trunc(16.7) → 16.0
double <b>sqrt</b> ( double ); float <b>sqrtf</b> ( float ); long double <b>sqrtl</b> ( long double );	Square root	

# The Math Library: *Math.h...*

Common used functions	Description
double <b>pow</b> ( double base, double exponent ); float <b>powf</b> ( float base, float exponent ); long double <b>powl</b> ( long double base, long double exponent );	Power $\text{base}^{\text{exponent}}$
double <b>log</b> ( double ); float <b>logf</b> ( float ); long double <b>logl</b> ( long double	Natural logarithm
double <b>exp</b> ( double x); float <b>expf</b> ( float ); long double <b>expl</b> ( long double );	$e^x$
Trigonometric functions:	sin, cos, tan, asin, acos, atan

# The Math Library: *Math.h...*

```
/* math_demo.c */
#include <stdio.h>
#include <time.h>
int main()
{ double x= 15.3, y=-2.6;
  printf("floor: %lf, %lf\n", floor(x), floor(y));
  printf("ceil: %lf, %lf\n", ceil(x), ceil(y));
  printf("round: %lf, %lf\n", round(x), round(y));
  printf("trunc: %lf, %lf\n", trunc(x), trunc(y));
  printf("sqrt: %lf\n", sqrt(x));
  printf("pow- x^y : %lf\n", pow(x,y));
  printf("exp- e^x: %lf\n", exp(x));
  printf("log(x): %lf\n", log(x));
  printf("log2(x): %lf\n", log(x)/log(2));
  getchar();
}
```

**Copy, paste, compile and run  
this program**

# The Character Library: *ctype.h*

```
int isalnum (int c);  
int isalpha (int c);  
int iscntrl (int c);  
int isdigit (int c);  
int isgraph (int c);  
int islower (int c);  
int isprint (int c);  
int ispunct (int c);  
int isspace (int c);  
int isupper (int c);  
int isxdigit (int c);  
int tolower (int c);  
int toupper (int c);
```

**alpha**: alphabet

**num**: numeric/number

**cntrl**: control

**print**: printable

**punct**: punctuation characters (ký tự phân cách)

**xdigit**: hexadecimal digit

**They are implemented using macro-definitions  
using the pre-processor **#define****

# Summary

## The standard C libraries

- Standard: *stdlib.h*
- Time: *time.h*
- Math: *math.h*
- Character: *ctype.h*

# Q&A



# Practice

## Multiplication Table

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	20	24	28
5	15	20	25	30	35
6	18	24	30	36	42
7	21	28	35	42	49

for (i=low to high)

for (j=low to high)

i\*j

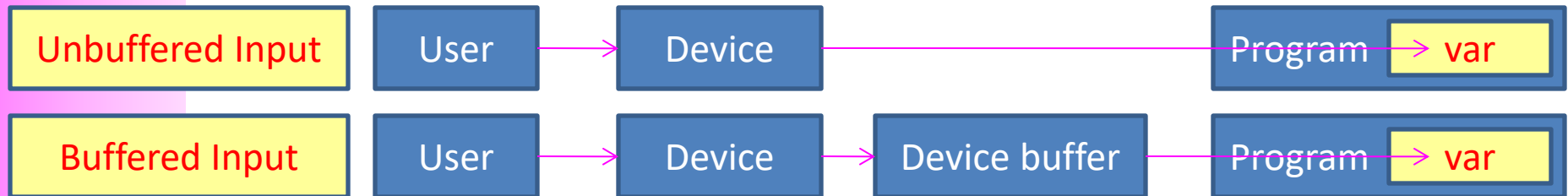
# **Input and validation**

## **Formatted output**

# Contents

- Types of Input
- Input a character: `getchar()`
- Input data: `scanf(...)`
- Validation

# Types of Input



- Interactive program (event-based program) uses unbuffered input. The program can respond to each and every keystroke directly.
- **Buffer:** A memory region is associated with a hardware such as keyboard, monitor, hard disk, ... It holds data temporarily. Information about some buffers can be seen from *MyComputer/ Properties/ Hardware/Device Manager/Select a device/Properties/Resources*
- Buffered input enables data editing before submission to a program. That means that input data can be treated as units and they can be pre-processed before they are passed to the program.
- Input functions will access device buffer to get data.

# Buffered Input

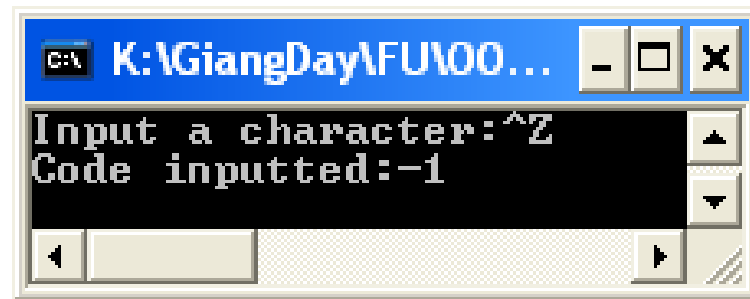
- To transfer the contents of a buffer to a program the user must press the '\n' character.
- Stream: A concept allows generalizing access data in a buffer as a chain of characters.
- Two C functions provide buffered input facilities on the standard input stream:
  - **getchar()** : get a character from the keyboard
  - **scanf (...)** : get data from the keyboard

# The getchar () function

- **getchar()** retrieves a single character from the standard input stream buffer without translating the input. Prototype: **int getchar(void );**
- **getchar** returns either the character code for the retrieved character or EOF.  
(EOF=-1, ctrl+z in Windows, ctrl+d in Unix)

Copy. Paste, compile and run the program with input: Ctrl + Z

```
#include <stdio.h>
int main()
{ char c;
  printf("Input a character:");
  c = getchar();
  printf("Code inputted:%d\n", c);
  getchar();
}
```



# getchar(): Clearing the buffer

- When the buffer input is used, in some cases, you may not get successfully data for a variable because some characters are remained in the keyboard buffer. How to remove them?

```
/* clear empties input buffer */
```

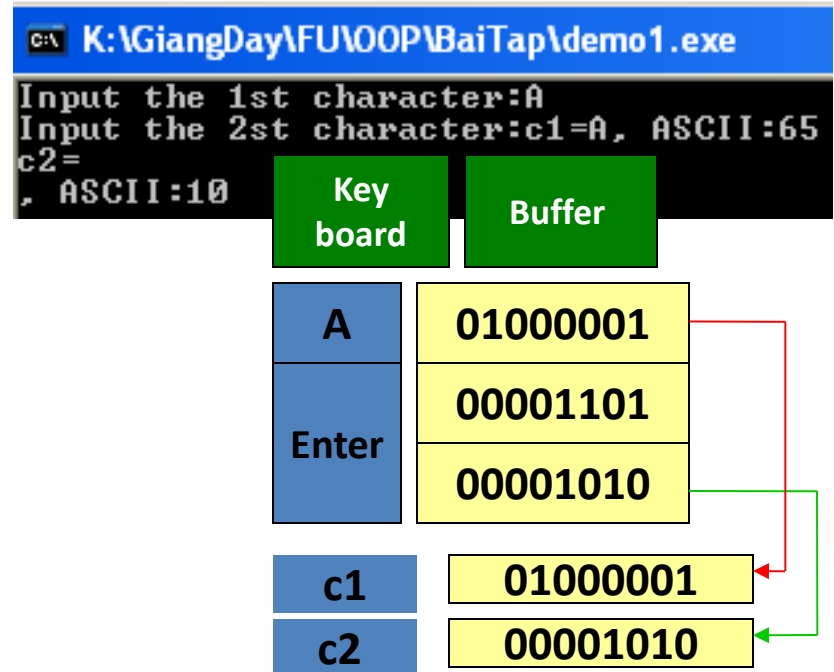
```
void clear (void) {  
    while ( getchar() != '\n' );  
}
```

- In some tools, the function `fflush(stdin)` is implemented for this purpose (in `stdio.h`).
- To assure that a program always receives new character data. Make clear the keyboard buffer before the operation of accepting a character (or a string)

# getchar()...

Copy, paste, compile and run the program:

```
#include <stdio.h>
int main()
{ char c1, c2;
  printf("Input the 1st character:");
  scanf("%c", &c1);
  printf("Input the 2st character:");
  c2 = getchar();
  printf("c1=%c, ASCII:%d\n", c1, c1);
  printf("c2=%c, ASCII:%d\n", c2, c2);
  getchar();
}
```



The problem is that after user presses 'A' and ENTER (2 code: 13, 10), ASCII codes of them are put into the keyboard buffer. The function scanf(...) will get 'A' to c1. The remaining codes, 13 and 10, are interpreted to the character '\n' – code 10 only, to c2 by the function getchar(). So, you can not get a new character for c2.



# getchar()...

Copy, paste, compile and run the program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ char c1, c2;
  printf("Input a character:");
  fflush(stdin);
  c1= getchar();
  printf("Input a character:");
  fflush(stdin);
  c2= getchar();
  printf("c1: %c, ASCII code: %d, %o, %X\n", c1,c1,c1,c1);
  printf("c2: %c, ASCII code: %d, %o, %X\n", c2,c2,c2,c2);
  system("pause");
}
```

Data input are proposed:  
A Z

The function **system("commandName")** , is prototyped in `stdlib.h` , allows calling a program stored in system file or an OS command.

# getchar()...

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

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

The user-defined function for clearing the keyboard buffer

```
main()
{
    char ch1, ch2;
    printf("input the first character: ");
    scanf("%c", &ch1);
    clear();
    printf("input the second character: ");
    ch2=getchar();
    printf("ch1=%c, ASCII code: %d\n", ch1, ch1);
    printf("ch2=%c, ASCII code: %d\n", ch2, ch2);
    getch();
}
```

```
Input the first character: A
Input the second character: B
ch1=A, ASCII code: 65
ch2=B, ASCII code: 66
```

# getchar()...

- Develop a program that will accept a string of characters until the key ENTER is pressed then number of digits, number of alphabets, and number of other characters are printed out.

## Do yourself

Variable: char c;

int numOfDigits=0, numOfAlpha=0, numOfOthers = 0

Algorithm

While (c=getchar() != '\n')

{ if c is a digit then numOfDigits++ ;

else if c is an alphabet then numOfAlpha++;

else numOfOthers++;

}

Print out numOfDigits;

Print out numOfDigits;

Print out numOfOthers;

# The scanf(...) function

- **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 (tội lỗi) character in the buffer, or emptied the buffer, in which case **scanf** waits until the user adds more data values.

# The scanf(...) function...

- Some things are concerned when the scanf(...) function is used:
  - How to specify the input format string?
  - How to separate input data?
  - How to realize that how many data were inputted successfully?
  - How to remove the byte '\n' in the keyboard buffer after an input operation? Is clearing the buffer is the only way?
- We will get the answers in the following slides.

# scanf ...: Conversion Specifiers

Specifier	Input value	Use with
%c %*c	character	char Remove one character in the input buffer
%d	decimal	char, int, short, long, long ong
%u	decimal	unsigned int, char, int, short, long, long long
%o	octal	unsigned int, char, int, short, long, long long
%x %X	hexadecimal	unsigned int, char, int, short, long, long long
%ld	long	long
%f	float	float
%lf	double	double
%llf	long double	long double

Data from the keyboard are passed to the buffer as characters (ASCII codes). So, conversion specifiers are instructions that specifying how to converse character data to typed-data. If a specifier does not match the data type of a variable, the conversion is incorrect.

# Conversion Specifiers...

```
#include <stdio.h>
#include <conio.h>
main()
{   long x;
    printf("x=  ");
    scanf("%lf", &x);
    printf("x= %ld ", x);
    getch();
}
```

```
x=  5.3
x= 858993459
```

The conversion specifier does not match the data type long

```
#include <stdio.h>
#include <conio.h>
main()
{   double x;
    printf("x=  ");
    scanf("%ld", &x);
    printf("x= %lf ", x);
    getch();
}
```


```
x=  5.3
x= 124030539581313790000000
```

The conversion specifier does not match the data type double

# scanf...: Default Separators

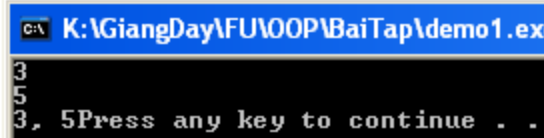
- If a input value is number lead by a whitespace , **scanf** treats the whitespace as a separator (*The whitespace characters include **newline, horizontal tab, form feed, vertical tab and space characters***).

```
#include <stdio.h>
#include <stdlib.h>
int n;
int main()
{   int m;
    scanf("%d%d", &n, &m);
    printf("%d, %d", n, m);
    system("pause");
}
```



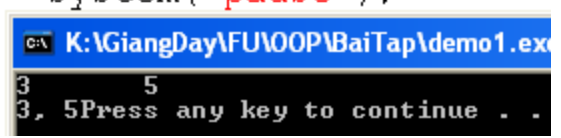
Space character

```
#include <stdio.h>
#include <stdlib.h>
int n;
int main()
{   int m;
    scanf("%d%d", &n, &m);
    printf("%d, %d", n, m);
    svstem("pause");
}
```



New line / Enter

```
#include <stdio.h>
#include <stdlib.h>
int n;
int main()
{   int m;
    scanf("%d%d", &n, &m);
    printf("%d, %d", n, m);
    system("pause");
}
```



Tab character



# scanf...: User-Defined Separators

- User can specify the character for separating input data.
- User-define separators will override default separators*

```
#include <stdio.h>
#include <stdlib.h>
int n;
int main()
{
    int m, k;
    scanf("%d,%d&%d", &n, &m, &k);
    printf("%d, %d, %d\n", n, m, k);
    system("pause");
}
```

```
C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe
10,20&30
10, 20, 30
Press any key to continue . . .
```

```
#include <stdio.h>
#include <stdlib.h>
int n;
int main()
{
    int m, k;
    scanf("%d,%d&%d", &n, &m, &k);
    printf("%d, %d, %d\n", n, m, k);
    system("pause");
}
```

```
C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe
10 20 30
10, 2, 37
Press any key to continue . . .
```

```
#include <stdio.h>
#include <stdlib.h>
int n;
int main()
{
    int m, k;
    scanf("%d,%d&%d", &n, &m, &k);
    printf("%d, %d, %d\n", n, m, k);
    system("pause");
}
```

```
C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe
10,20 ,30
10, 20, 37
Press any key to continue . . .
```

# scanf(...): Using %\*c for removing a character

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n;
    char c;
    scanf("%d%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe

70D  
70, D  
Press any key to continue . . .

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n;
    char c;
    scanf("%d%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

Input: 70 ENTER

C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe

70  
70,  
Press any key to continue . . .

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n;
    char c;
    scanf("%d%*c%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe

70 D  
70, D  
Press any key to continue . . .

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n;
    char c;
    scanf("%d%*c%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe

70  
D  
70, D  
Press any key to continue . . .

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n;
    char c;
    scanf("%d%*c%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe

70PD  
70, D  
Press any key to continue . . .

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n;
    char c;
    scanf("%d%*c", &n);
    c = getchar();
    printf("%d, %c\n", n, c);
    system("pause");
}
```

C:\K:\GiangDay\FU\OOP\BaiTap\demo1.exe

70  
D  
70, D  
Press any key to continue . . .

# scanf...: Number of data fields are inputted

- **Return values from the scanf function:**
  - **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 (-1)** 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.

# Number of data fields are inputted...

```
#include <stdio.h>
#include <stdlib.h>
int main()
{   int m, n; double x;
    int count;
    char c;
    count=scanf("%d%d%lf", &m, &n, &x);
    printf("count=%d, m=%d, n=%d, x=%lf\n", count,m,n,x);
    system("pause");
}
```

C:\ K:\GiangDay\FU\OOP\BaiTap\demo1.exe

^Z  
count=-1, m=2, n=37, x=0.000000  
Press any key to continue . . . \_

C:\ K:\GiangDay\FU\OOP\BaiTap\demo1.exe

asjklfghjk  
count=0, m=2, n=37, x=0.000000  
Press any key to continue . . . \_

C:\ K:\GiangDay\FU\OOP\BaiTap\demo1.exe

12dfghjkl;  
count=1, m=12, n=37, x=0.000000  
Press any key to continue . . . \_

C:\ K:\GiangDay\FU\OOP\BaiTap\demo1.exe

12 789 asd  
count=2, m=12, n=789, x=0.000000  
Press any key to continue . . . \_

C:\ K:\GiangDay\FU\OOP\BaiTap\demo1.exe

12 789 12.7803  
count=3, m=12, n=789, x=12.780300  
Press any key to continue . . . \_

# Input 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.
- The program should have code for trapping all erroneous input, which includes:
  - *invalid characters*
  - *trailing characters*
  - *out-of-range input*
  - *incorrect number of input fields*

# Input Validation: Example

The following function will ensure that inputted integer must be in the range [min,max] and no invalid characters or trailing character following inputted digits(EXCEPT the ENTER key)

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

    do {
        printf("Enter a whole number\n"
               "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;
}
```

```
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 **getDouble** 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.

# Summary

## Input and Validation

- Types of Input
- getchar
- scanf
- Input validation

# 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 library *stdio.h* containing functions for print out data:
  - *putchar(int)* : character
  - *printf (format\_string, varList)*: list of data

# The function putchar(int)

- **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');`

```
#include <stdio.h>
#include <conio.h>
main()
{ char c;
  int result;
  printf("Input a character: ");
  c=getchar();
  printf("the inputted character is: ");
  result=putchar(c);
  printf("\n The return of putchar is %d", result);
  getch();
}
```

```
Input a character: A
The inputted character is A
The return of putchar is 65
```

```
Input a character:
The inputted character is
The return of putchar is 10
```

# The function printf(...)

- **printf** sends data under format control to the standard output stream buffer and returns the number of characters sent.
- Syntax: **printf ( format string , value, ..., value )**

*The format string is a literal string that consists of characters interspersed (đặt rải rác) with conversion specifiers.*

*Conversion specifier begins with a % and ends with a conversion character*

# printf(...): Format string

- Between the % and the **conversion character**, there may be  
% **flags** **width** . **precision** **size** **conversion\_character**

## flags

- prescribes left justification of the converted value in its field
- 0** pads the field width with leading zeros

**size** identifies the size of data type of the value passed.

Size Specifier	Use With
none	int
hh	char
h	short
l	long
ll	long long

Size Specifier	Use With
none	float
l	double
L	long double

# printf(...): Conversion Specifiers

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

# printf(...): Some Examples

```
printf("-----\n");
printf("%d|<--      %%d\n", 4321);
printf("%10d|<--   %%10d\n", 4321);
printf("%010d|<--   %%010d\n", 4321);
printf("%-10d|<--   %%-10d\n", 4321);
/* floats */
printf("\n* floats *\n");
printf("00000000011\n");
printf("12345678901\n");
printf("-----\n");
printf("%f|<--  %%f\n", 4321.9876546);
/* doubles */
printf("\n* doubles *\n");
printf("00000000011\n");
printf("12345678901\n");
printf("-----\n");
printf("%lf|<--  %%lf\n", 4321.9876546);
printf("%10.3lf|<--  %%10.3lf\n", 4321.9876546);
printf("%010.3lf|<--  %%010.3lf\n", 4321.9876546);
printf("%-10.3lf|<--  %%-10.3lf\n", 4321.9876546);
/* characters */
printf("\n* chars *\n");
printf("00000000011\n");
printf("12345678901\n");
printf("-----\n");
printf("%c|<--      %%c\n", 'd');
printf("%d|<--      %%d\n", 'd');
printf("%o|<--      %%o\n", 'd');
```

```
-----
4321|<--      %d
          4321|<-- %10d
0000004321|<-- %010d
4321      |<--  %-10d

* floats *
00000000011
12345678901
-----
4321.987655|<-- %f

* doubles *
00000000011
12345678901
-----
4321.987655|<-- %lf
          4321.988|<-- %10.3lf
004321.988|<-- %010.3lf
4321.988  |<--  %-10.3lf

* chars *
00000000011
12345678901
-----
d|<--      %c
100|<--    %d
144|<--    %o
```

# printf(...): Some Examples...

```
#include <stdio.h>
#include <conio.h>
main()
{ int n;
  long lo;
  char ch;
  double d;
  printf("input a integer, a long number, a character: ");
  printf("and a double number: (use blank)");
  scanf("%d %ld %c %lf",&n, &lo, &ch,&d);
  printf("%-20s:%10d%15ld%5c%12.6lf\n","The inputted values are",n,lo,ch,d);
  getch();
}
```

```
Input a integer number, a long number, a character, and a double number: (use blank) 5 135000 A 45.23
The inputted values are:
```

	5	135000	A	45.230000
Size= 20 left align	Size=10 right align	Size=15 right align	5 right align	Size=12 including the dot character, right align

# Summary

## Formatted Output

- putchar
- printf

## Q&A



# Exercise

Write a C program using the following simple menu:

- 1- Processing date data
- 2- Character data
- 3- Quit

Choose an operation:

- When user chooses 1: User will enter values of date, month, year then the program will announce whether this date is valid or not.
- When user chooses 2: User will enter two characters, then the program will print out ASCII codes of characters between them using descending order. Examples:

Input: ca

Output: c: 99, 63h

b: 98, 62h

a: 97, 61h

# Exercise

Write a C program using the following simple menu:

- 1- Quadratic equation ( phương trình bậc 2)
- 2- Bank deposit problem
- 3- Quit

Choose an operation:

- When user chooses 1: User will enter values describing a quadratic equation then the program will print out its solution if it exists.
- When user chooses 2: User will enter his/her deposit ( a positive number), monthly rate ( a positive number but less than or equal to 0.1), number of months ( positive integer), then the program will print out his/her amount after this duration.