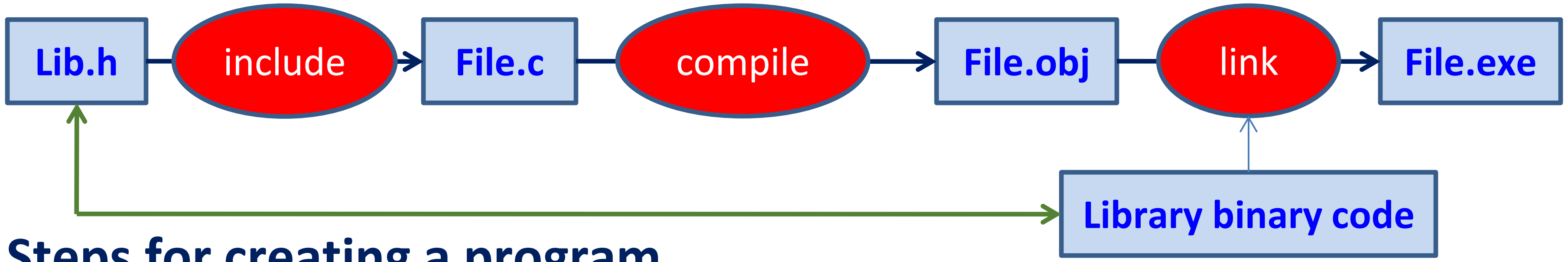
Fpt University

TRƯỜNG ĐẠI HỌC FPT

Libraries

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, …etc.
- 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.



Steps for creating a program

Objectives

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

- Stardard and Math libraries: stdlib.h, math.h → Mathematical Functions
- Time library: time.h
- The Character library: ctype.h

Contents

The standard C libraries

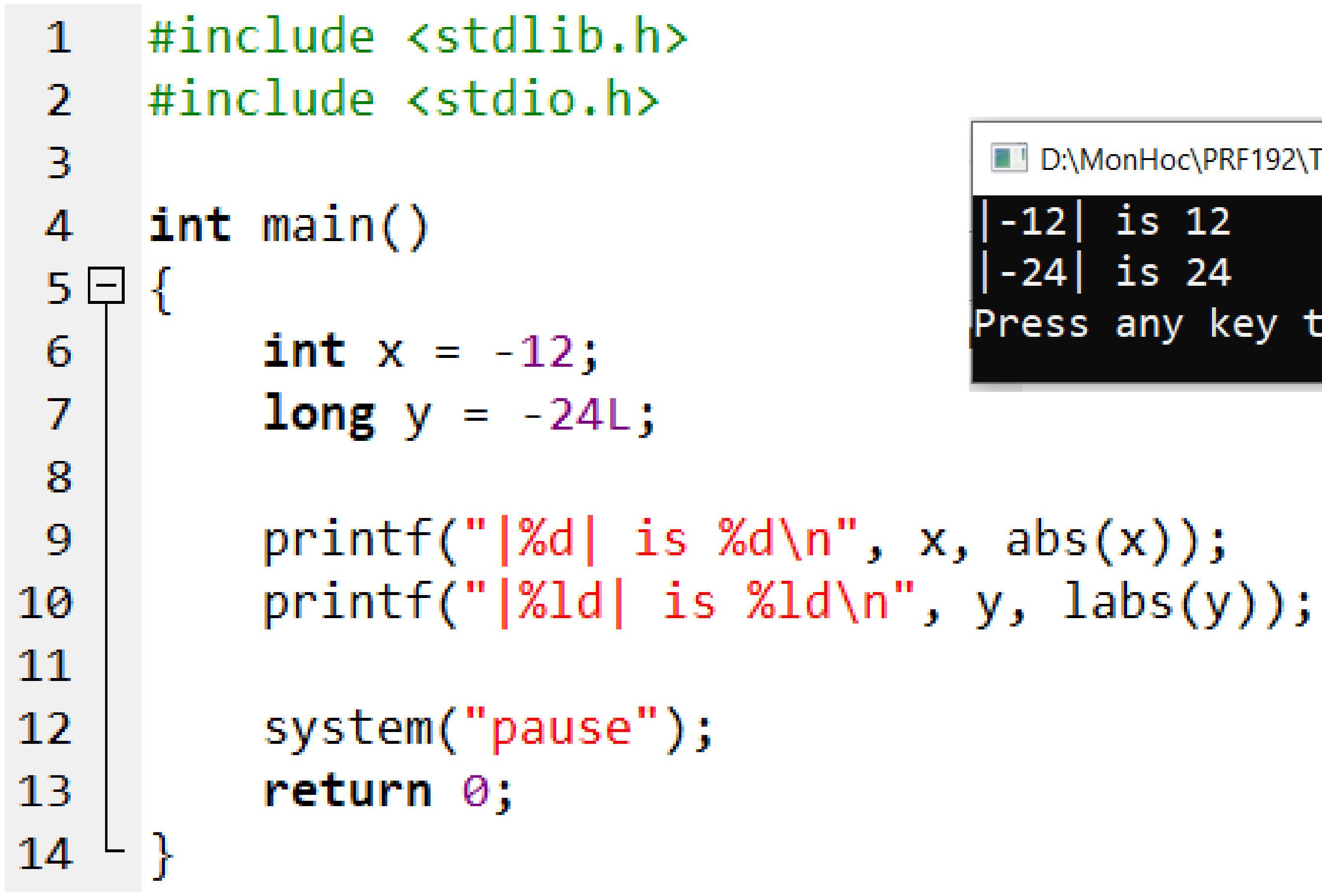
- Standard
- Math
- Time
- Character

1 - Standard Library

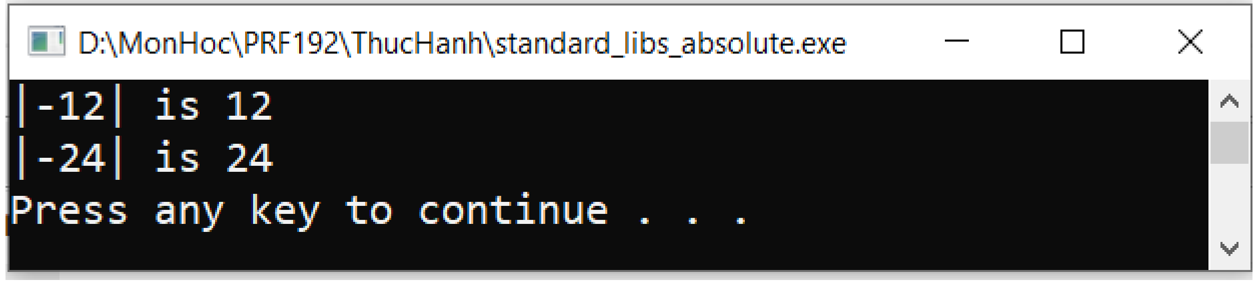
- The header file **<stdlib.h>** contains prototypes for the functions that perform the more general mathematical calculations.
- **Integer absolute value:**

Function	Syntax	Description
abs()	int abs(int);	return the absolute value of the argument
labs()	long labs(long);	
llabs()	long long llabs(long long);	

Integer absolute value: Example



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

int main()
{
    int x = -12;
    long y = -24L;

    printf("|%d| is %d\n", x, abs(x));
    printf("|%ld| is %ld\n", y, labs(y));

    system("pause");
    return 0;
}
```



D:\MonHoc\PRF192\ThucHanh\standard_libs_absolute.exe

```
|-12| is 12
|-24| is 24
Press any key to continue . . .
```

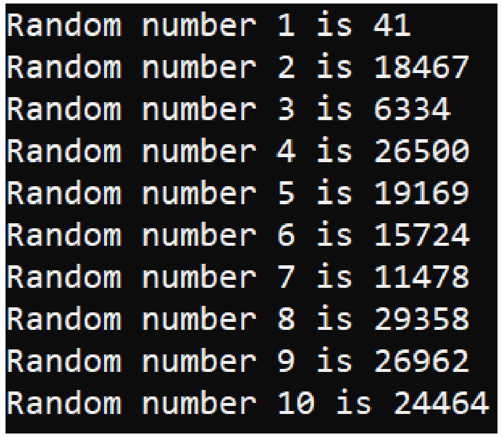
Random Number

- **rand()**: returns a pseudo-random integer in the range **0** to **RAND_MAX**. **RAND_MAX** is implementation-dependent but **no less** than 32767.
- Syntax:

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

- Example 1:

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

int main()
{
    int i;

    for (i = 0; i < 10 ; i++)
    {
        printf("Random number %d is %d\n", i+1, rand());
    }

    return 0;
}
```

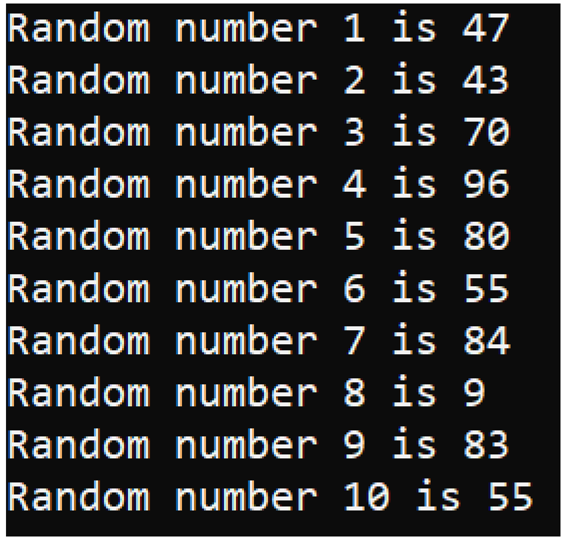


```
Random number 1 is 41
Random number 2 is 18467
Random number 3 is 6334
Random number 4 is 26500
Random number 5 is 19169
Random number 6 is 15724
Random number 7 is 11478
Random number 8 is 29358
Random number 9 is 26962
Random number 10 is 24464
```

Random Number: Example

- Example 2: Random a set of 10 pseudo-random integers between 6 and 100

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

int main()
{
    int i, n, a = 6, b = 100;

    for (i = 0; i < 10 ; i++)
    {
        n = a + rand() % (b + 1 - a);
        printf("Random number %d is %d\n", i+1, n);
    }

    return 0;
}
```

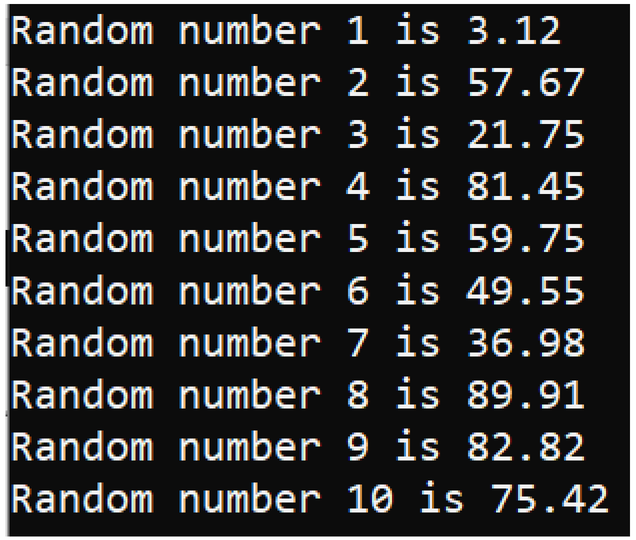


```
Random number 1 is 47
Random number 2 is 43
Random number 3 is 70
Random number 4 is 96
Random number 5 is 80
Random number 6 is 55
Random number 7 is 84
Random number 8 is 9
Random number 9 is 83
Random number 10 is 55
```

Random Number: Example (cont.)

- Example 3: Random a set of 10 pseudo-random floating-point numbers between 3.0 and 100

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

int main()
{
    int i;
    double x, a = 3.0, b = 100.0;

    for (i = 0; i < 10 ; i++)
    {
        x = a + ((double) rand() / RAND_MAX * (b - a));
        printf("Random number %d is %.2lf\n", i+1, x);
    }

    return 0;
}
```



```
Random number 1 is 3.12
Random number 2 is 57.67
Random number 3 is 21.75
Random number 4 is 81.45
Random number 5 is 59.75
Random number 6 is 49.55
Random number 7 is 36.98
Random number 8 is 89.91
Random number 9 is 82.82
Random number 10 is 75.42
```

Random Number (cont.)

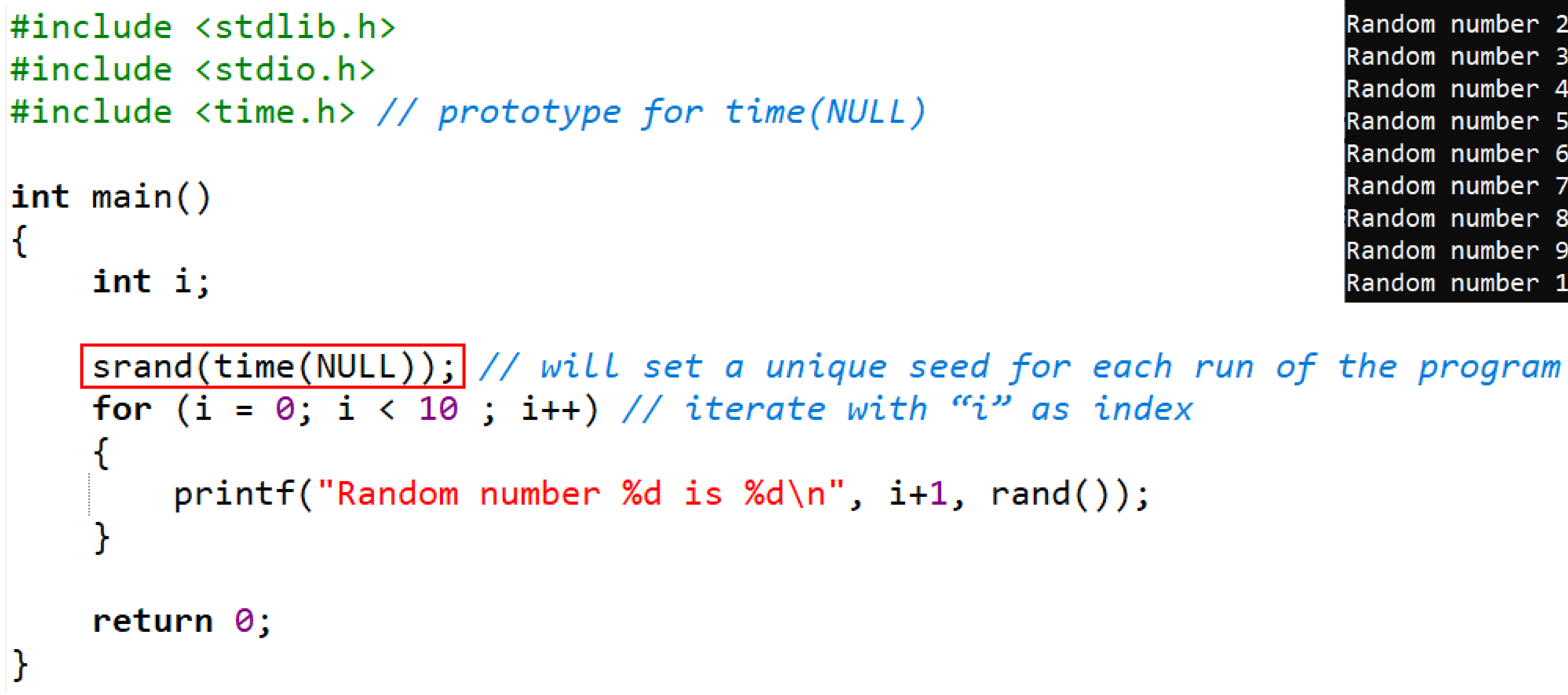
- To generate a different set of random numbers for every run we add a call to the function **srand()**. **srand()** sets the seed for the random number generator.
- Syntax:

```
int srand(unsigned seed);
```

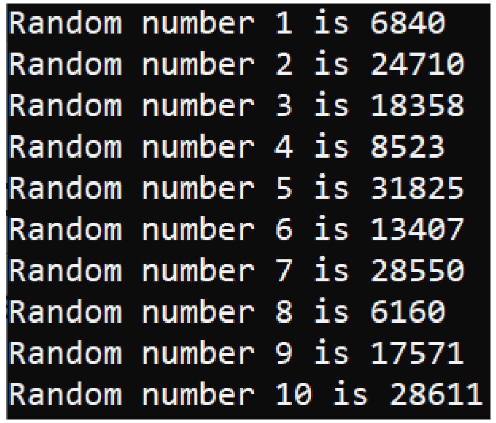
- Note:
  - unsigned is a type that only holds **non-negative** integer values.
  - Call srand() once with time(NULL) as the argument before the first call to rand(), typically at the start of your program.

Random Number: Example

- Program outputs a different set of 10 pseudo-random numbers with every run:



```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h> // prototype for time(NULL)

int main()
{
    int i;

    srand(time(NULL)); // will set a unique seed for each run of the program
    for (i = 0; i < 10 ; i++) // iterate with "i" as index
    {
        printf("Random number %d is %d\n", i+1, rand());
    }

    return 0;
}
```



```
Random number 1 is 6840
Random number 2 is 24710
Random number 3 is 18358
Random number 4 is 8523
Random number 5 is 31825
Random number 6 is 13407
Random number 7 is 28550
Random number 8 is 6160
Random number 9 is 17571
Random number 10 is 28611
```

Math Library

- The math library contains many functions that perform mathematical calculations.
- Their prototypes are listed in **<math.h>**

Common Functions	Syntax	Description	Example

fabs()
fabsf()
	double fabs(double);
	float fabsf(float);
		return the absolute value of the argument	fabs(-12.5) → 12.5
			fabsf(-12.5f) → 12.5

…	…	…	…

Math Library- math.h (cont.)

Common Functions	Syntax	Description	Example

floor()
floorf()
	double floor(double);
	float floorf(float);
		return the largest integer value not greater than the argument	floor(16.3) → 16.0

ceil()
ceilf()
	double ceil(double);
	float ceilf(float);
		return the smallest integer value not less than the argument	ceil(16.3) → 17.0

round()
roundf()
	double round(double);
	float roundf(float);
		return the integer value closest to the argument	round(16.3) → 16.0
			round(-16.3) → -16.0

trunc()
truncf()
	double trunc(double);
	float truncf(float);
		return the integer part of the argument	trunc(16.7) → 16.0
			trunc(-16.7) → -16.0

sqrt()
sqrtf()
	double sqrt(double);
	float sqrtf(float);
		return the square root of the argument	sqrt(16.0) → 4.0

Math Library - math.h (cont.)

Common Functions	Syntax	Description	Example

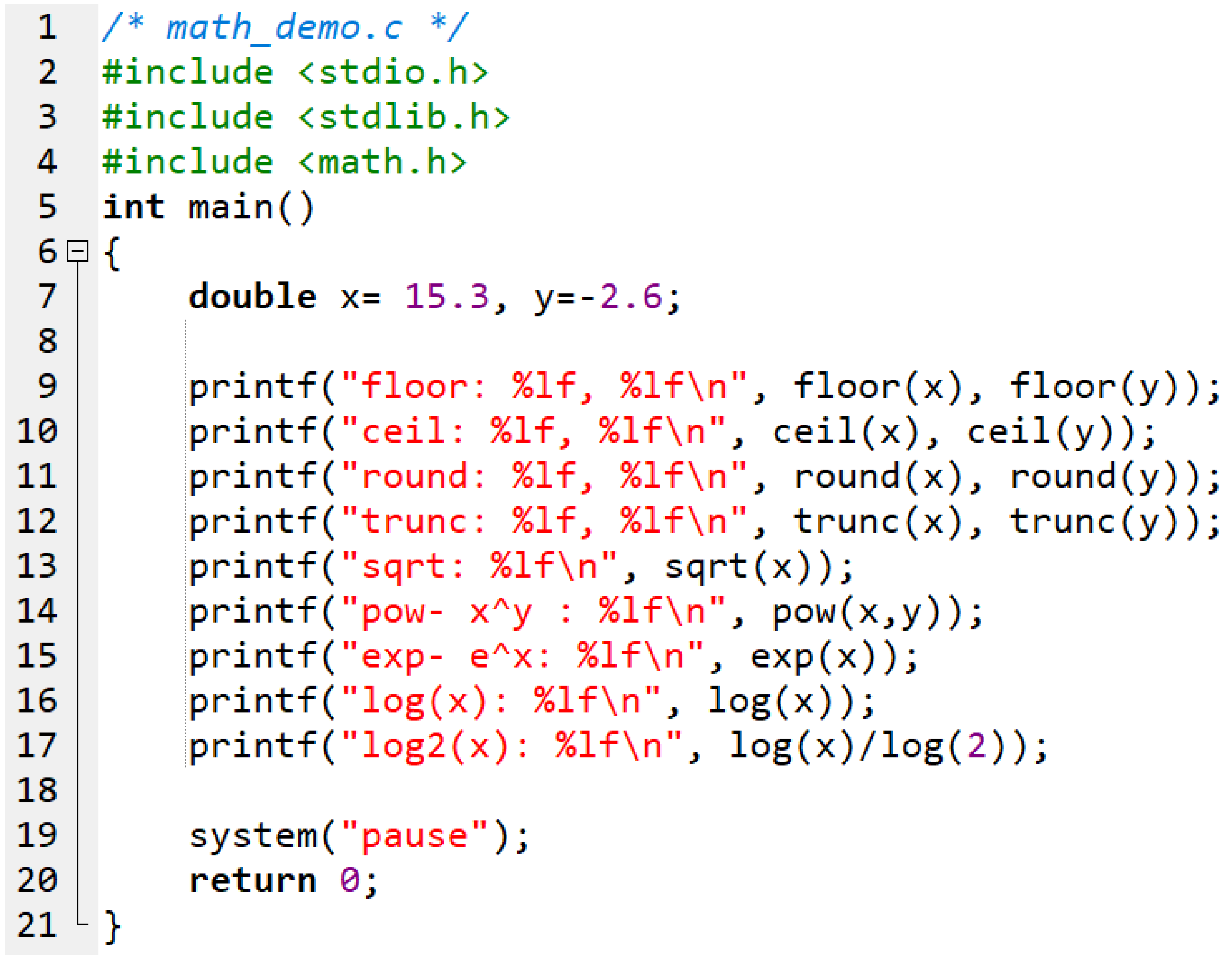
pow()
powf()
	double pow(double base, double exponent);
	float powf(float base, float exponent);
		return the result of the first argument raised to the power of the second argument	pow(12.5, 3)) → 1953.125

log()
logf()
	double log(double);
	float logf(float);
		return the natural logarithm of the argument	log(2.718281828459045) → 1.0

exp()
expf()
	double exp(double);
	float expf(float);
		return the natural anti-logarithm of the argument.	exp(1.0) → 2.718281828459045

Math Library: Demo

Write, compile and run this program



```c
/* math_demo.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.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));

    system("pause");
    return 0;
}
```

2 - Time Library: time.h

- **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 = 1milisecond).
- 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;)

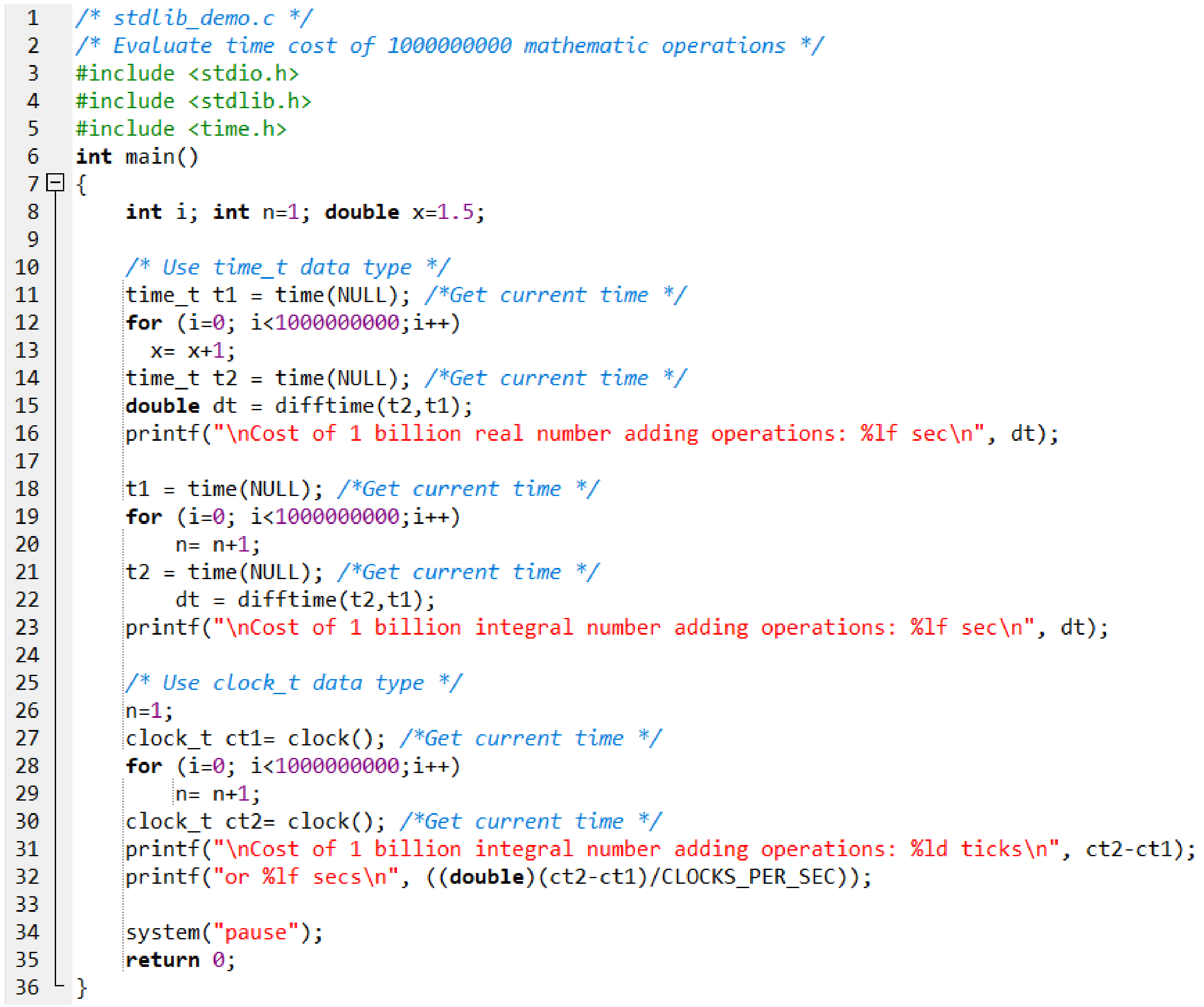
Time Library: time.h (cont.)

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.

Time Library: Demo

Write, compile and run this program



```c
/* stdlib_demo.c */
/* Evaluate time cost of 1000000000 mathematic operations */
#include <stdio.h>
#include <stdlib.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("or %lf secs\n", ((double)(ct2-ct1)/CLOCKS_PER_SEC));

    system("pause");
    return 0;
}
```

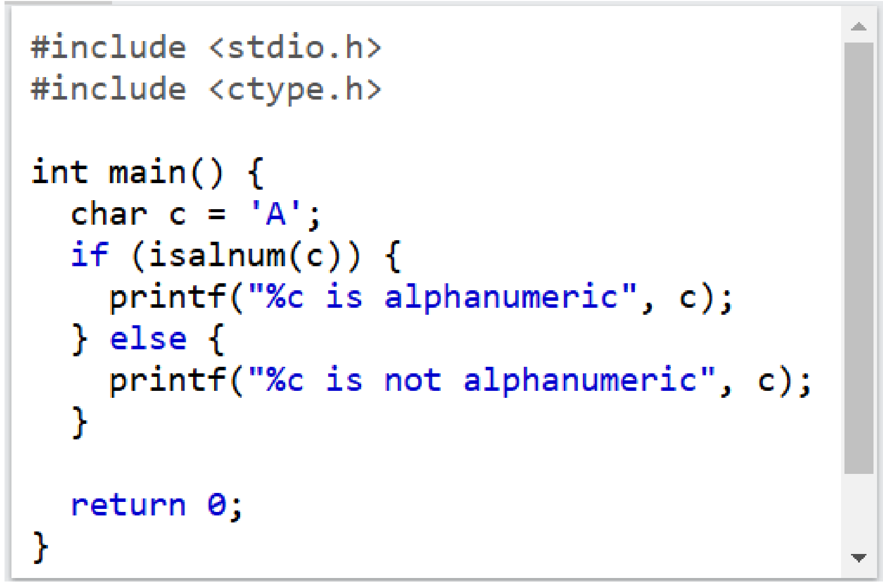
3 - The Character Library (ctype.h)

- The **<ctype.h>** header provides many functions for classifying and modifying characters.

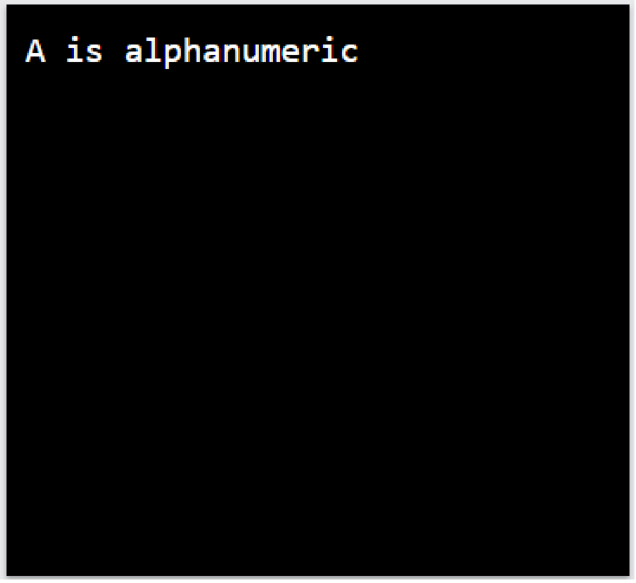
Common Functions	Description
isalnum()	Checks whether a character is alphanumeric
isalpha()	Checks whether a character is a letter
isblank()	Checks whether a character is a space or tab
isdigit()	Checks whether a character is a decimal digit
islower()	Checks whether a character is a lowercase letter
isupper()	Checks whether a character is an uppercase letter
isspace()	Checks whether a character is a whitespace character
tolower()	Returns a lowercase version of a character
toupper()	Returns an uppercase version of a character

isalnum() Function

- The **isalnum()** function returns a non-zero value (equivalent to boolean true) if a character is alphanumeric, meaning an **alphabet letter** (a-z) or **a number** (0-9).
- Example of characters that are not alphanumeric: **(space)!#%&?** ...etc.
- Syntax: **int isalnum(int c);**
- Example:



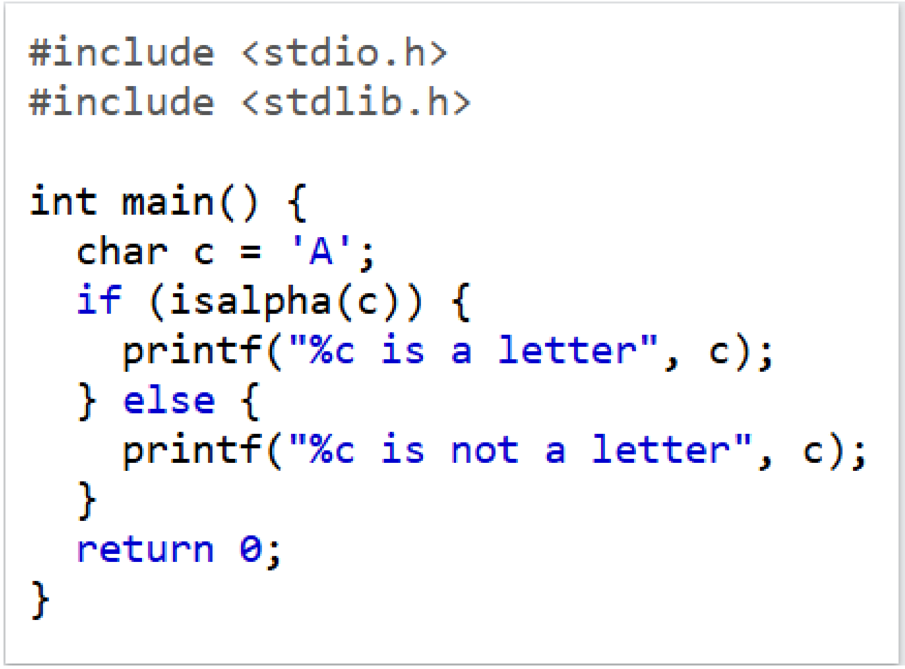
```c
#include <stdio.h>
#include <ctype.h>

int main() {
  char c = 'A';
  if (isalnum(c)) {
    printf("%c is alphanumeric", c);
  } else {
    printf("%c is not alphanumeric", c);
  }

  return 0;
}
```



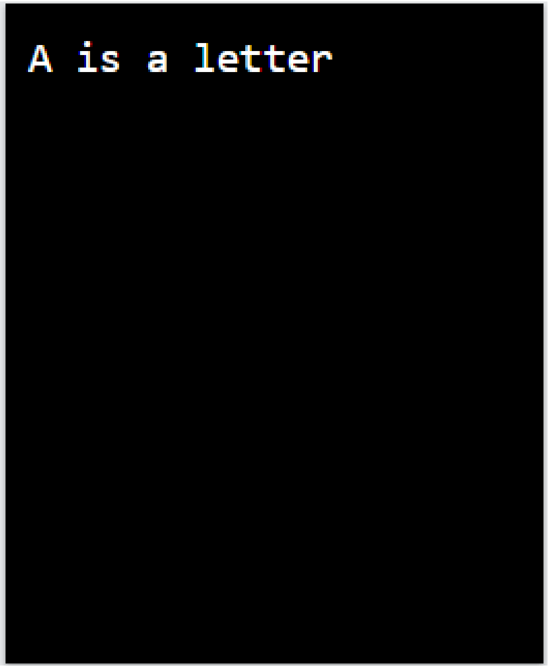
```
A is alphanumeric
```

isalpha() Function

- The **isalpha()** function returns a non-zero value (equivalent to boolean *true*) if a character is **an alphabet letter** (a-z).
- Syntax: **int isalpha(int c);**
- Example:



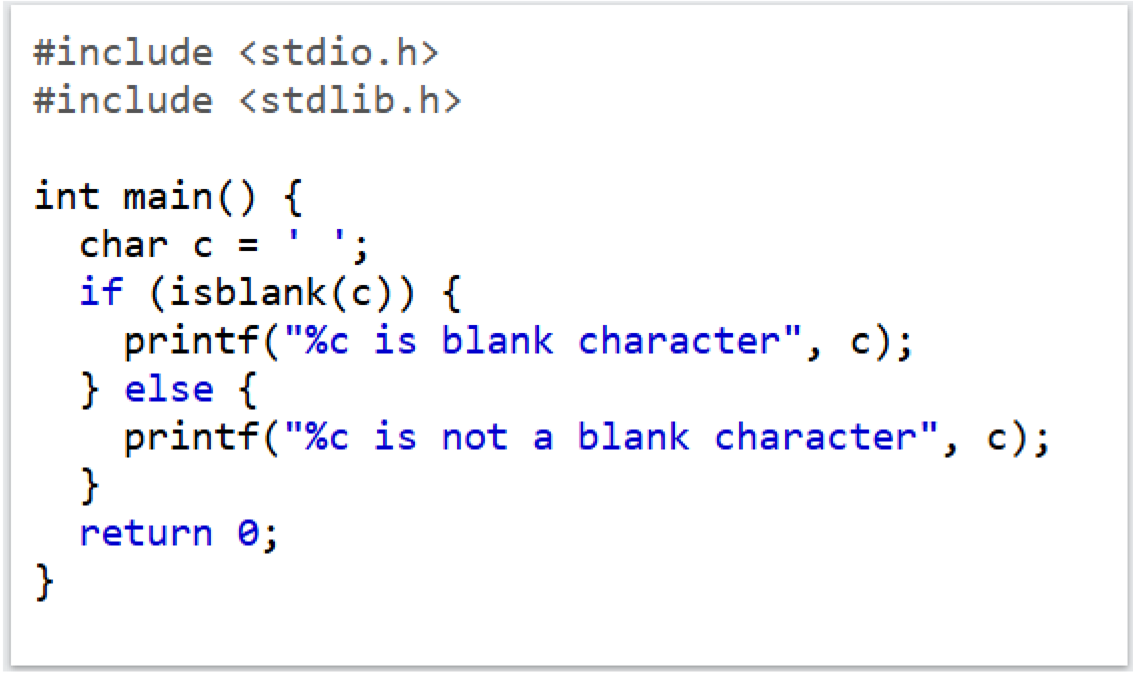
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  char c = 'A';
  if (isalpha(c)) {
    printf("%c is a letter", c);
  } else {
    printf("%c is not a letter", c);
  }
  return 0;
}
```



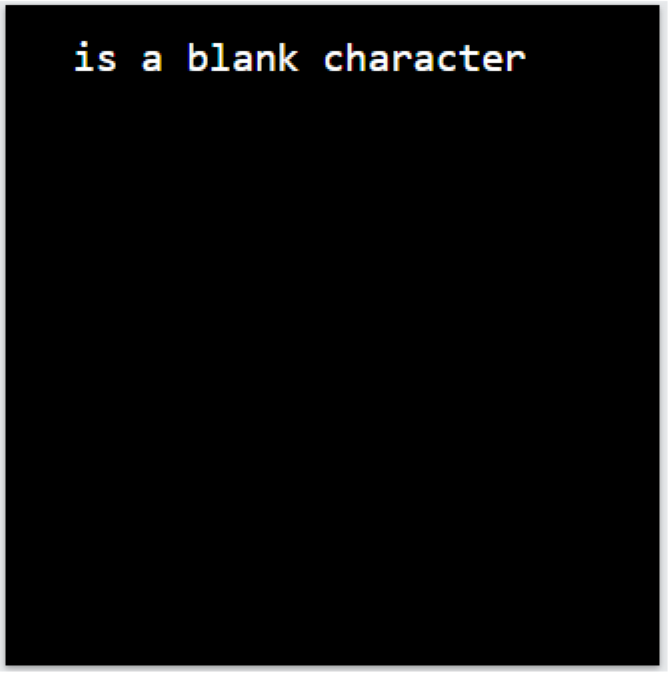
```
A is a letter
```

isblank() Function

- The **isblank()** function returns a non-zero value (equivalent to boolean true) if a character is a **space (blank)** or **a tab**.
- Syntax: **int isblank(int c);**
- Example:



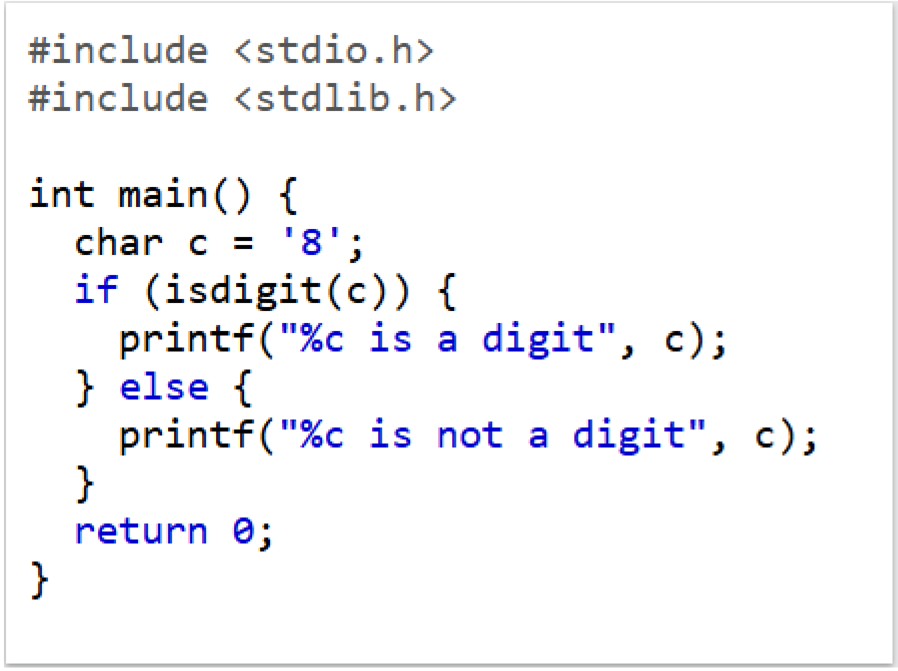
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  char c = ' ';
  if (isblank(c)) {
    printf("%c is blank character", c);
  } else {
    printf("%c is not a blank character", c);
  }
  return 0;
}
```



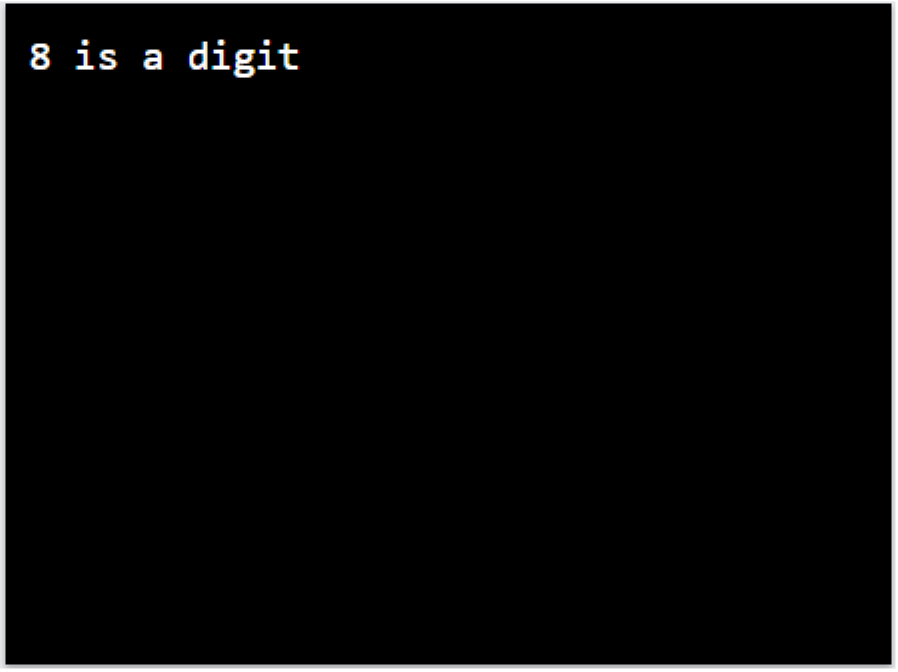
```
 is a blank character
```

isdigit() Function

- The **isdigit()** function returns a non-zero value (equivalent to boolean true) if the character is **a digit**.
- Syntax: **int isdigit(int c);**
- Example:



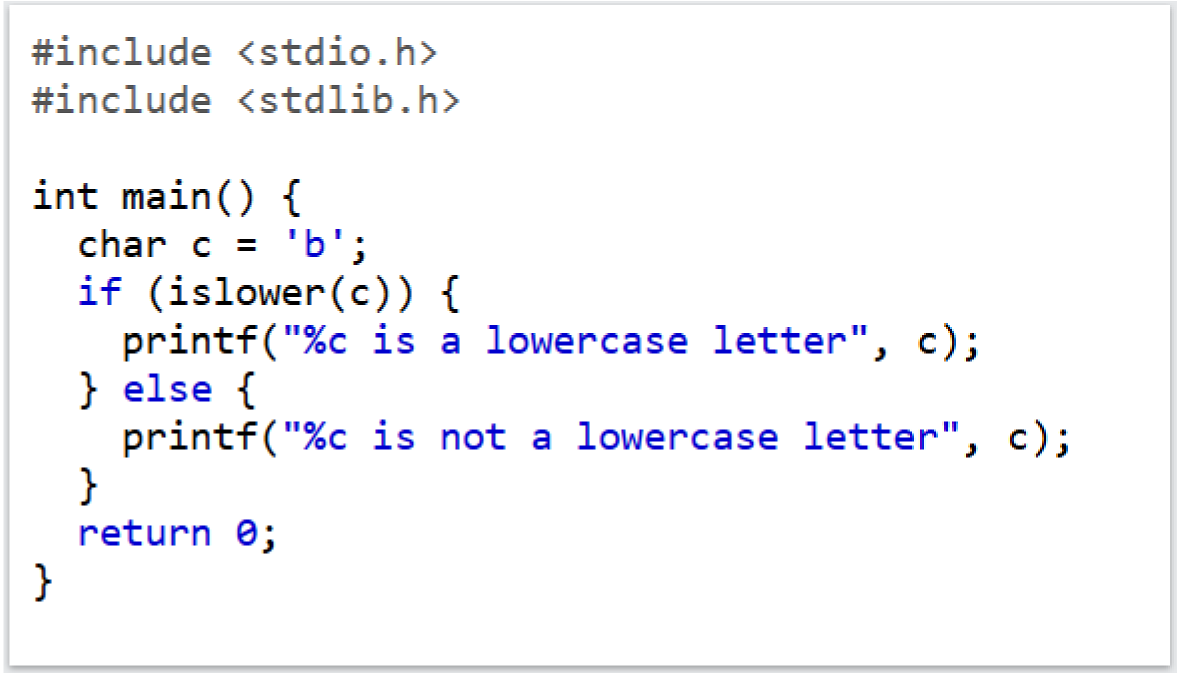
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  char c = '8';
  if (isdigit(c)) {
    printf("%c is a digit", c);
  } else {
    printf("%c is not a digit", c);
  }
  return 0;
}
```



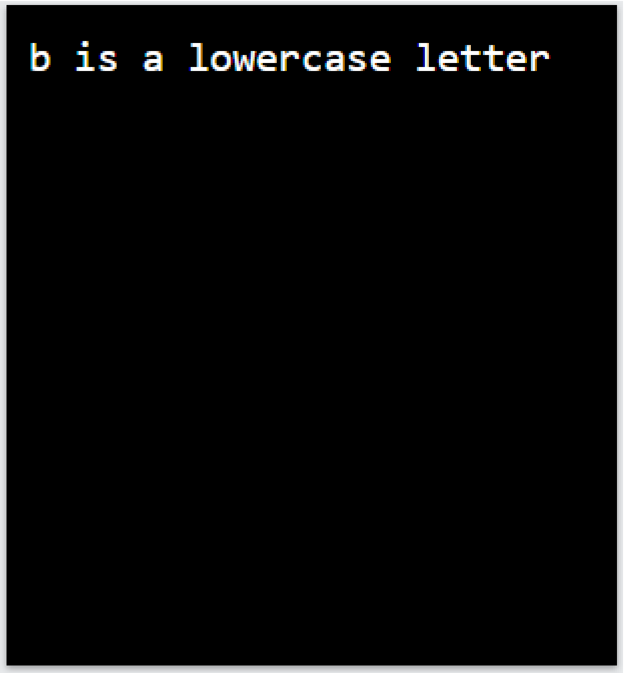
```
8 is a digit
```

islower() Function

- The **islower()** function returns a non-zero value (equivalent to boolean true) if a character is **a lowercase letter**.
- Syntax: **int islower(int c);**
- Example:



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

int main() {
  char c = 'b';
  if (islower(c)) {
    printf("%c is a lowercase letter", c);
  } else {
    printf("%c is not a lowercase letter", c);
  }
  return 0;
}
```



```
b is a lowercase letter
```

isupper() Function

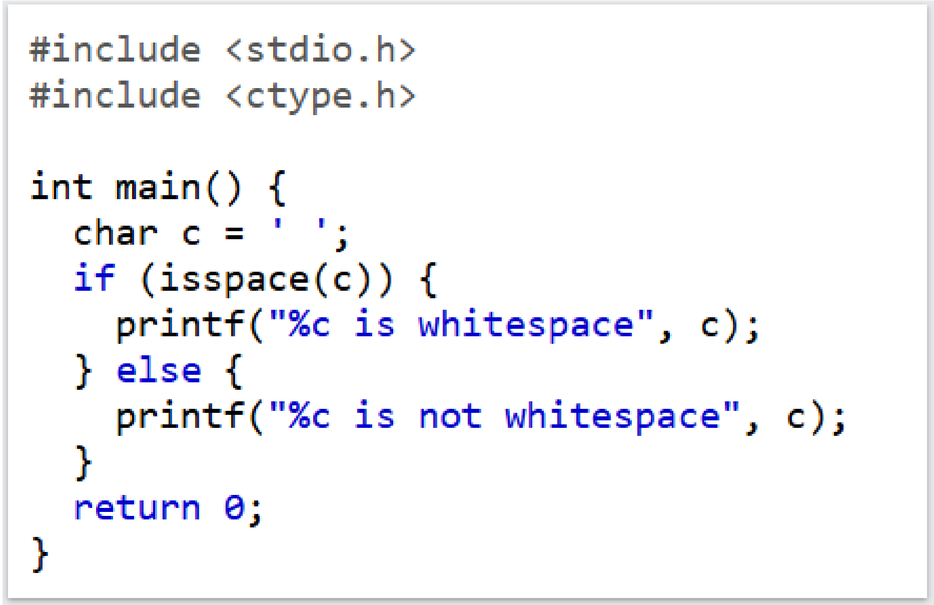
- The **isupper()** function returns a non-zero value (equivalent to boolean true) if a character is **an uppercase letter**.
- Syntax: **int isupper(int c);**
- Example:

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

int main() {
  char c = 'b';
  if (isupper(c)) {
    printf("%c is an uppercase letter", c);
  } else {
    printf("%c is not an uppercase letter", c);
  }
  return 0;
}
```

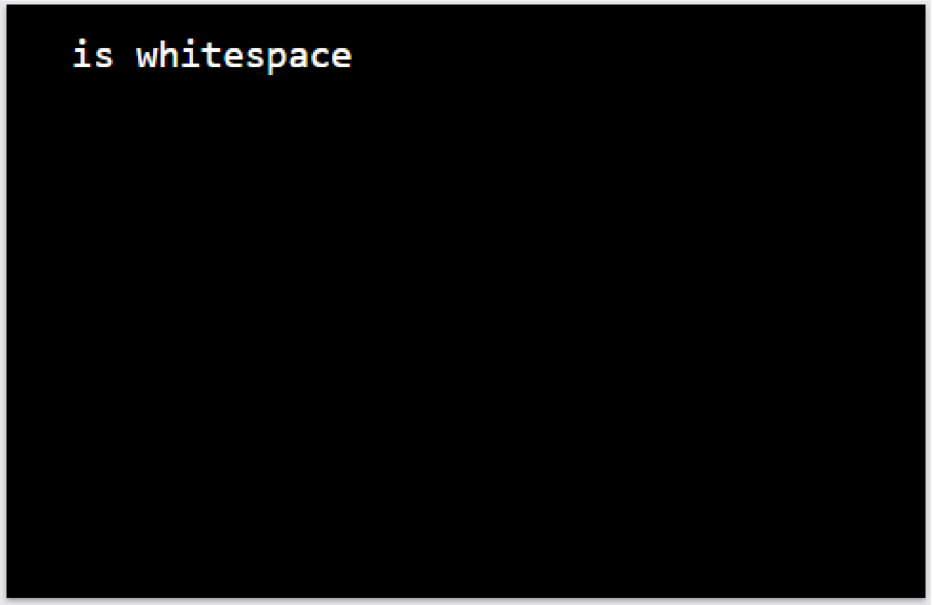
```
b is not an uppercase letter
```

isspace() Function

- The **isspace()** function returns a non-zero value (equivalent to boolean true) if a character is **a whitespace** character. Whitespace characters are **spaces**, **tabs** and **newline** characters.
- Syntax: **int isspace(int c);**
- Example:



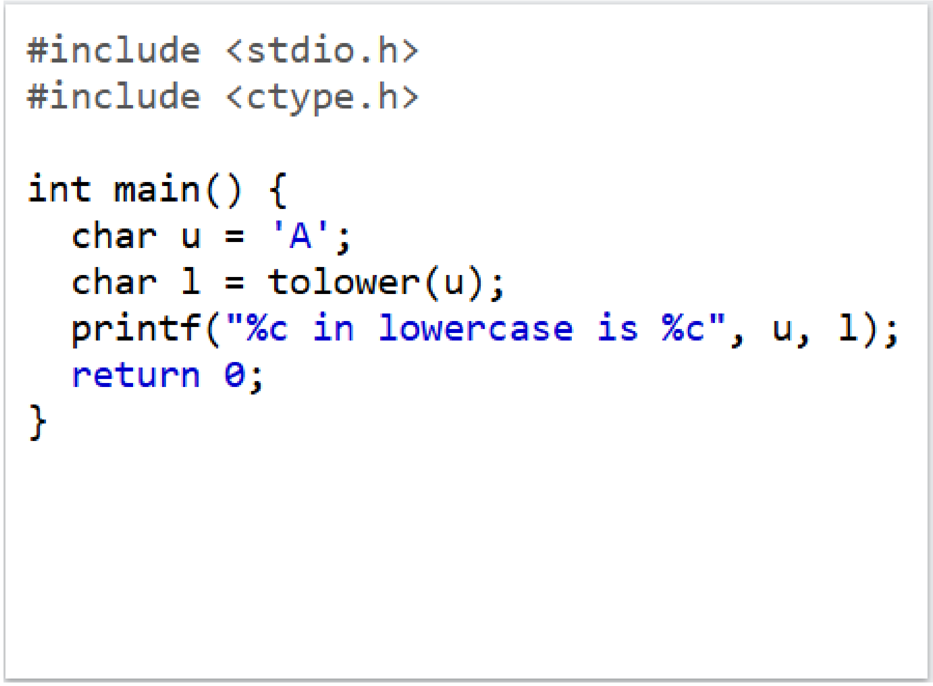
```c
#include <stdio.h>
#include <ctype.h>

int main() {
  char c = ' ';
  if (isspace(c)) {
    printf("%c is whitespace", c);
  } else {
    printf("%c is not whitespace", c);
  }
  return 0;
}
```



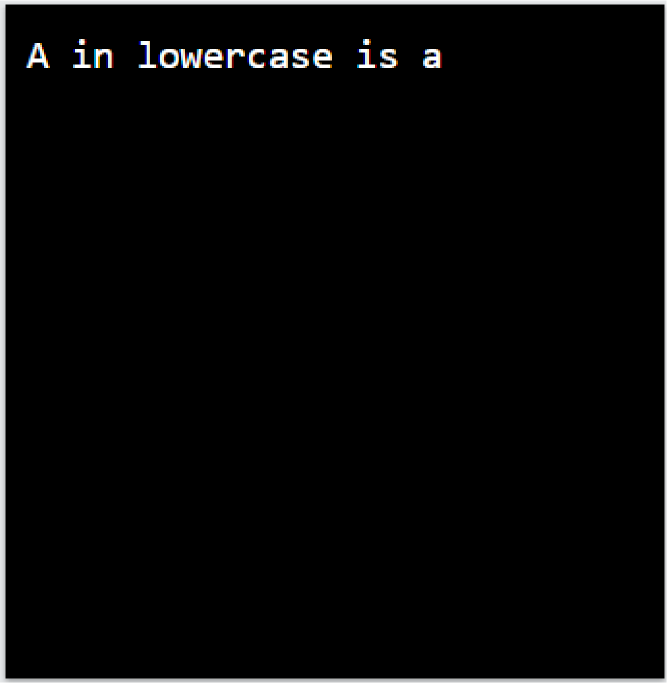
```
  is whitespace
```

tolower() Function

- The **tolower()** function returns the ASCII value of a lowercase version of the character.
- Syntax: **int tolower(int c);**
- Example:



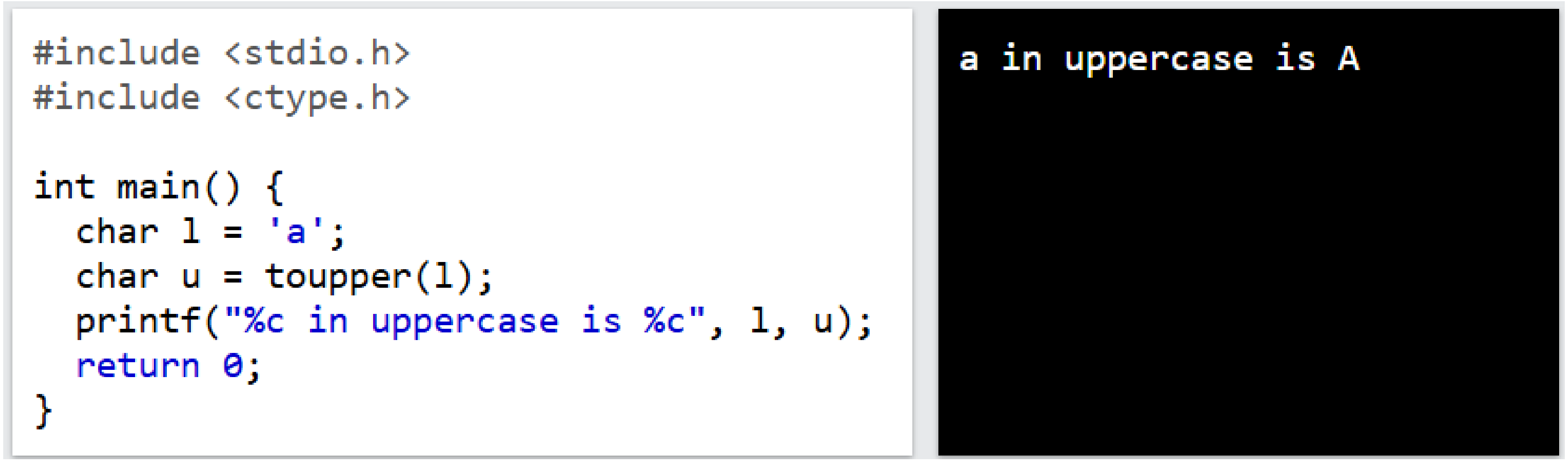
```
#include <stdio.h>
#include <ctype.h>

int main() {
  char u = 'A';
  char l = tolower(u);
  printf("%c in lowercase is %c", u, l);
  return 0;
}
```



```
A in lowercase is a
```

toupper() Function

- The **toupper()** function returns the ASCII value of an uppercase version of the character.
- Syntax: **int toupper(int c);**
- Example:



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

int main() {
  char l = 'a';
  char u = toupper(l);
  printf("%c in uppercase is %c", l, u);
  return 0;
}
```

```
a in uppercase is A
```

Exercise 1: Character Classification

- Write a program that takes a character as input and determines:
  1. Whether it is a digit, uppercase letter, lowercase letter, or none of these.
  2. If it is a letter, convert it to the opposite case (uppercase to lowercase and vice versa).

Summary

The standard C libraries

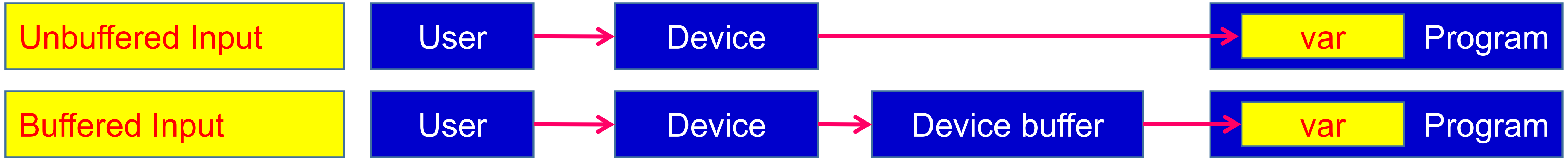
- Standard: **stdlib.h**
- Time: **time.h**
- Math: **math.h**
- Character: **ctype.h**

Q&A

Input and Validation

Contents

1 - 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.
- **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, using standard library <**stdio.h>**
  - **getchar()**: get a character from the keyboard
  - **scanf (…)**: get data from the keyboard

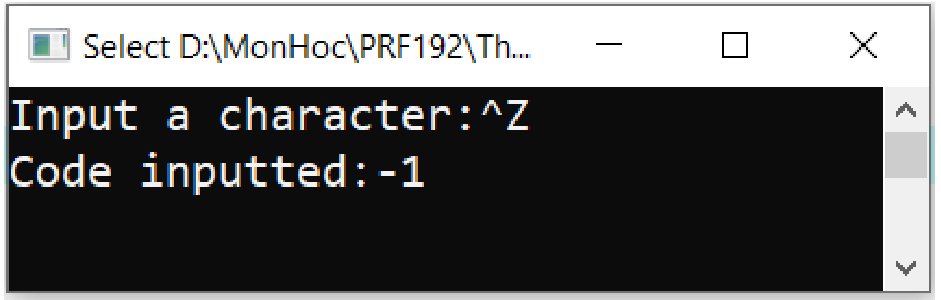
2 - The getchar() function

- **getchar()** retrieves a single character from the standard input stream buffer without translating the input.
- Syntax: **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);
   return 0;
}
```



```
Input a character:^Z
Code inputted:-1
```

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?**

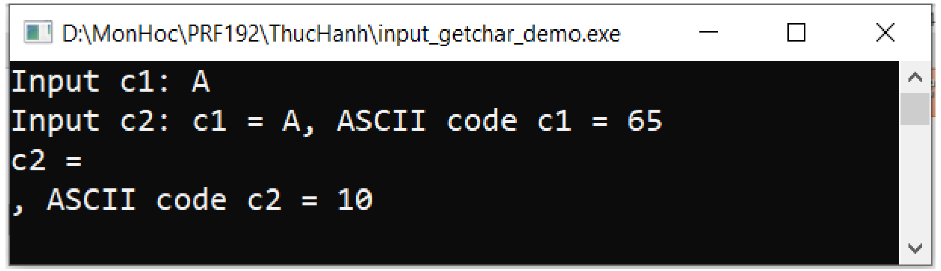
```
/* clear empties input buffer */
void clear (void) {
    while ( getchar() != '\n’ );
}
```

- 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() function: Problem

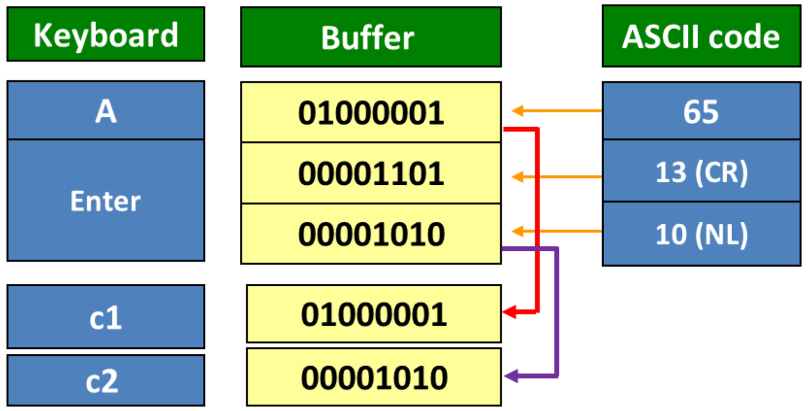
- Write a program input two character variables from STDIN. Print out the value and ASII code.

```c
#include <stdio.h>
int main()
{
    char c1, c2;

    printf("Input c1: ");
    scanf("%c", &c1);
    printf("Input c2: ");
    c2 = getchar();

    printf("c1 = %c, ASCII code c1 = %d\n", c1, c1);
    printf("c2 = %c, ASCII code c2 = %d\n", c2, c2);

    return 0;
}
```



D:\MonHoc\PRF192\ThucHanh\input_getchar_demo.exe

```
Input c1: A
Input c2: c1 = A, ASCII code c1 = 65
c2 =
, ASCII code c2 = 10
```

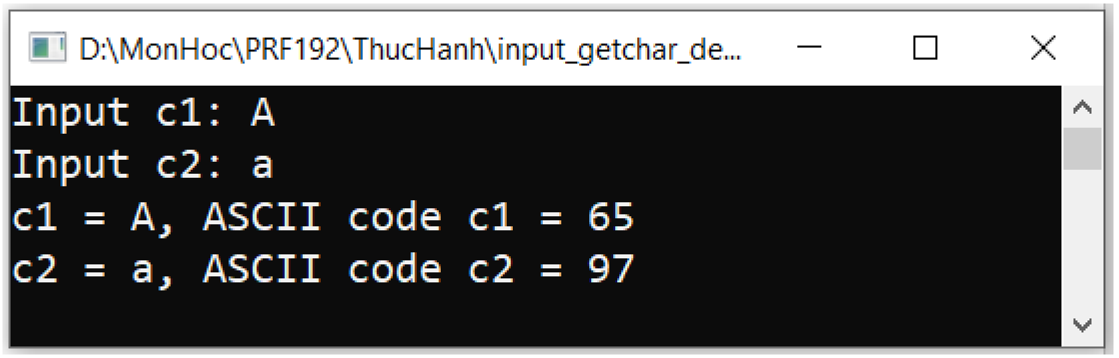


The problem is that after user presses ‘A’ and ENTER (2 code: 13, 10), ASCII codes of them are put into the keyboard buffer. The function scanf(…) will get ‘A’ to c1. The remaining codes, 13 and 10, are interpreted to the character ‘\n’ – code 10 only, to c2 by the function getchar(). So, you can not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

getchar() function: Solution 1

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
   char c1, c2;

   printf("Input c1: ");
   fflush(stdin);
   scanf("%c", &c1);
   printf("Input c2: ");
   fflush(stdin);
   c2 = getchar();

   printf("c1 = %c, ASCII code c1 = %d\n", c1, c1);
   printf("c2 = %c, ASCII code c2 = %d\n", c2, c2);

   return 0;
}
```

fflush(stdin) used to clear input buffer


D:\MonHoc\PRF192\ThucHanh\input_getchar_de...

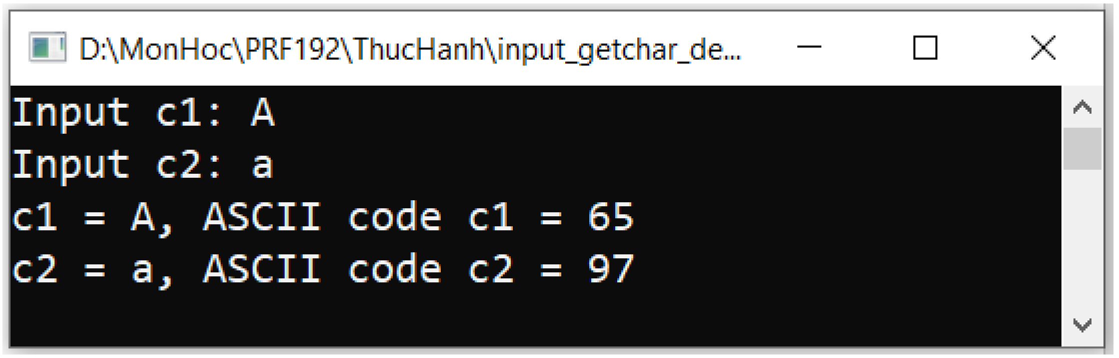
```
Input c1: A
Input c2: a
c1 = A, ASCII code c1 = 65
c2 = a, ASCII code c2 = 97
```

Warning: The fflush(stdin) function should not be overused. This function is not defined in the C standard library and is not supported by other compilers.

getchar() function: Solution 2

```c
#include <stdio.h>

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

int main()
{
   char c1, c2;

   printf("Input c1: ");
   scanf("%c", &c1);
   clear();
   printf("Input c2: ");
   c2 = getchar();

   printf("c1 = %c, ASCII code c1 = %d\n", c1, c1);
   printf("c2 = %c, ASCII code c2 = %d\n", c2, c2);

   return 0;
}
```

The user-defined function for clearing the keyboard buffer


D:\MonHoc\PRF192\ThucHanh\input_getchar_de...

```
Input c1: A
Input c2: a
c1 = A, ASCII code c1 = 65
c2 = a, ASCII code c2 = 97
```

Exercise 2: Input Characters

- 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.
- *Hint*:

Using library functions in ctype.h

```
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;
```

3 - scanf(…) function

- **scanf** retrieves data values from the standard input stream buffer under format control.
- Syntax: **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(…) function (cont.)

- 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

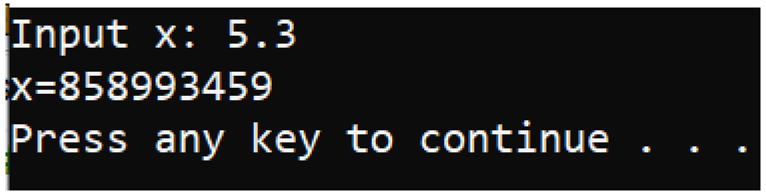
Specifier	Input value	Use with
%c %*c	character	char Remove one character in the input buffer
%d	decimal	char, int, short, long, long ong
%u	decimal	unsigned int, char, int, short, long, long long
%o	octal	unsigned int, char, int, short, long, long long
%x %X	hexadecimal	unsigned int, char, int, short,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

scanf(…): Conversion Specifiers (cont.)

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

int main()
{
    long x;

    printf("Input x: ");
    scanf("%lf", &x);
    printf("x=%ld\n", x);

    system("pause");
    return 0;
}
```

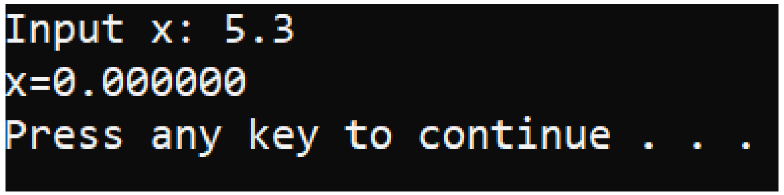
The conversion specifier does not match the data type long



```
Input x: 5.3
x=858993459
Press any key to continue . . .
```

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

int main()
{
    double x;

    printf("Input x: ");
    scanf("%ld", &x);
    printf("x=%lf\n", x);

    system("pause");
    return 0;
}
```

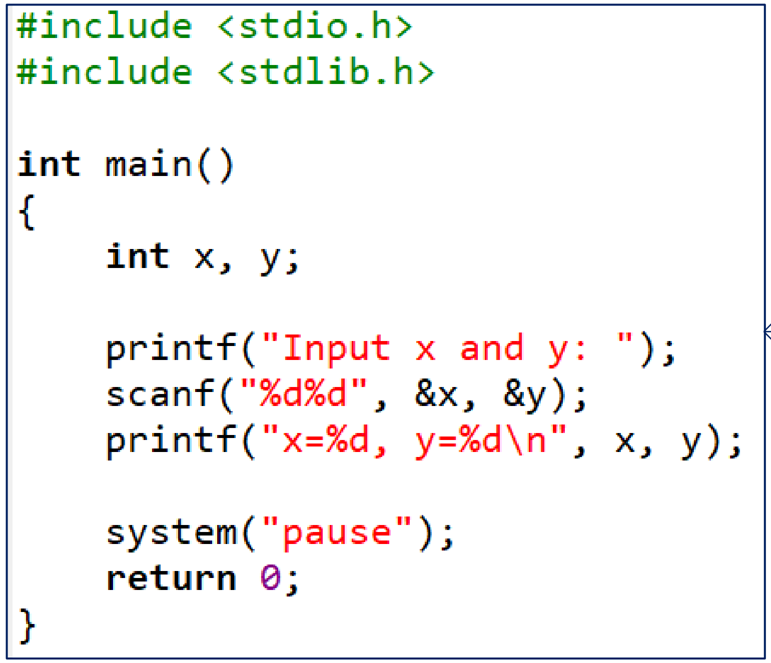
The conversion specifier does not match the data type double



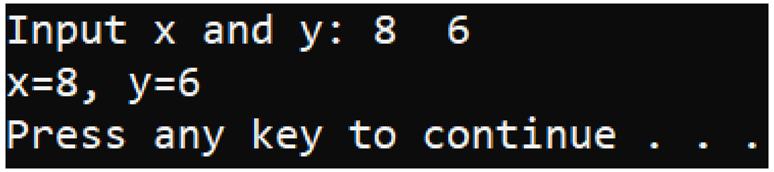
```
Input x: 5.3
x=0.000000
Press any key to continue . . .
```

scanf(…) function: 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**).

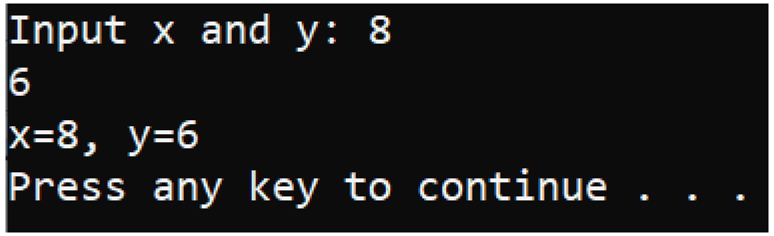


```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x, y;

    printf("Input x and y: ");
    scanf("%d%d", &x, &y);
    printf("x=%d, y=%d\n", x, y);

    system("pause");
    return 0;
}
```



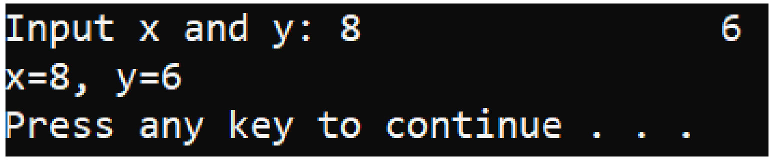
```
Input x and y: 8   6
x=8, y=6
Press any key to continue . . .
```

Space character



```
Input x and y: 8
6
x=8, y=6
Press any key to continue . . .
```

New line / Enter

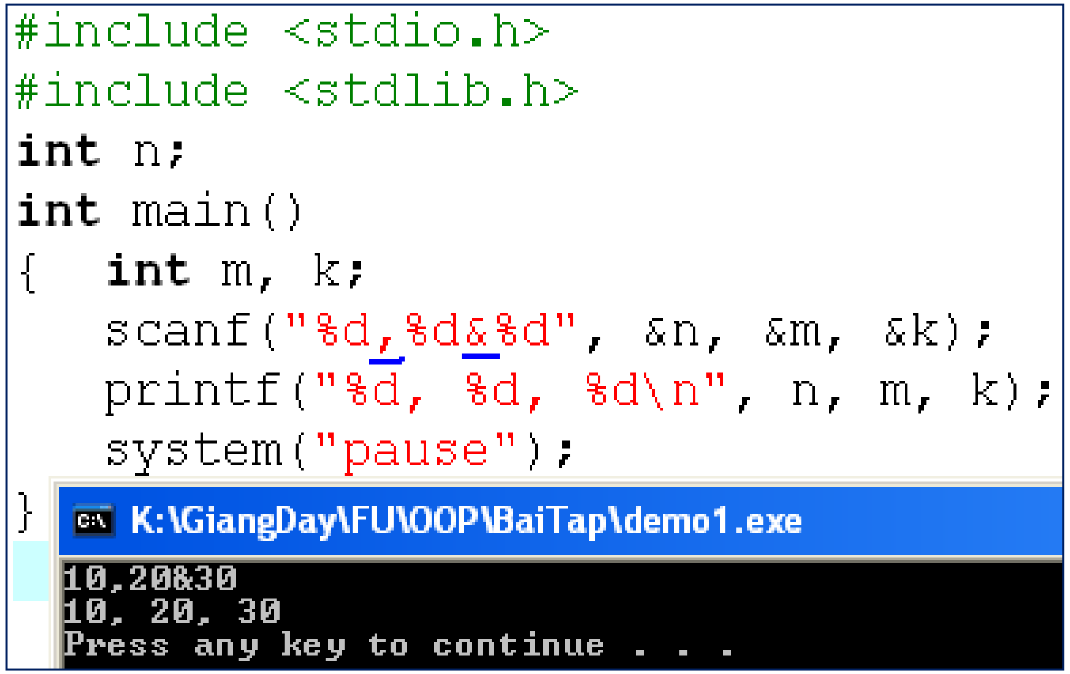


```
Input x and y: 8                6
x=8, y=6
Press any key to continue . . .
```

Tab character

scanf(…) function: User-defined Separators

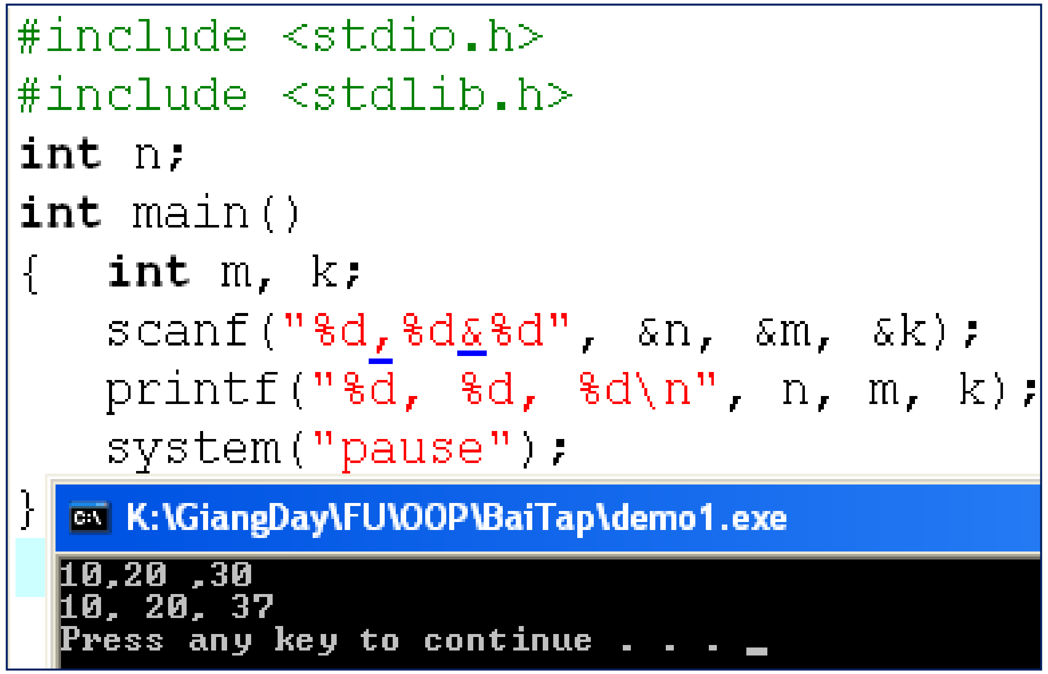
- User can specify the character for separating input data.
- User-define separators will override default separators



```
#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");
}
```

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

```
10,20&30
10, 20, 30
Press any key to continue . . .
```

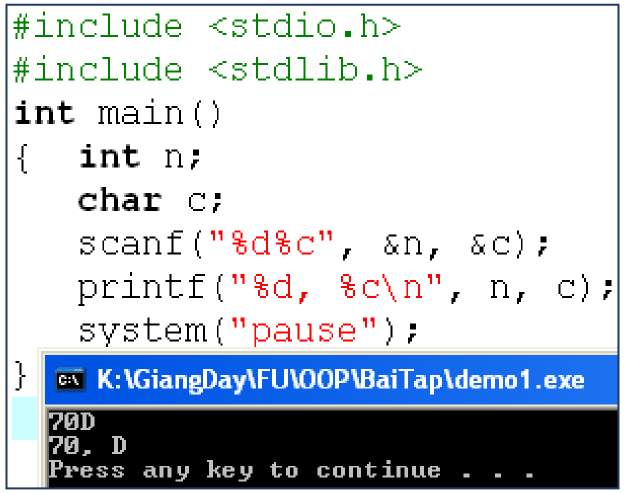


```
#include <stdio.h>
#include <stdlib.h>
int n;
int main()
{   int m, k;
    scanf("%d,%d&%d", &n, &m, &k);
    printf("%d, %d, %d\n", n, m, k);
    system("pause");
}
```

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

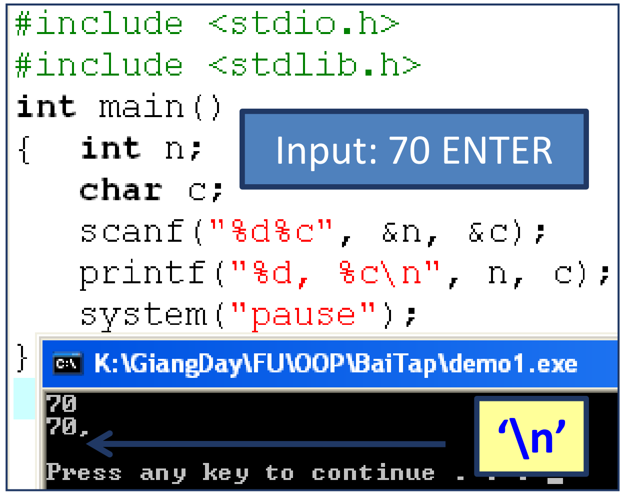
```
10,20 ,30
10, 20, 37
Press any key to continue . . . _
```

scanf(…) function: Using %*c for removing a character



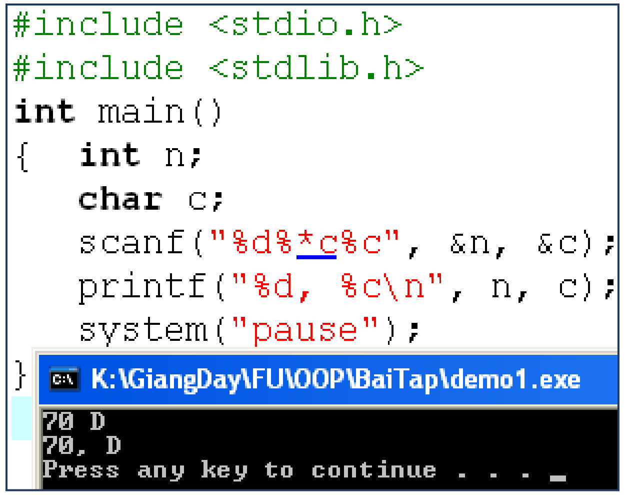
```
#include <stdio.h>
#include <stdlib.h>
int main()
{   int n;
    char c;
    scanf("%d%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

```
K:\GiangDay\FU\OOP\BaiTap\demo1.exe
70D
70, D
Press any key to continue . . .
```



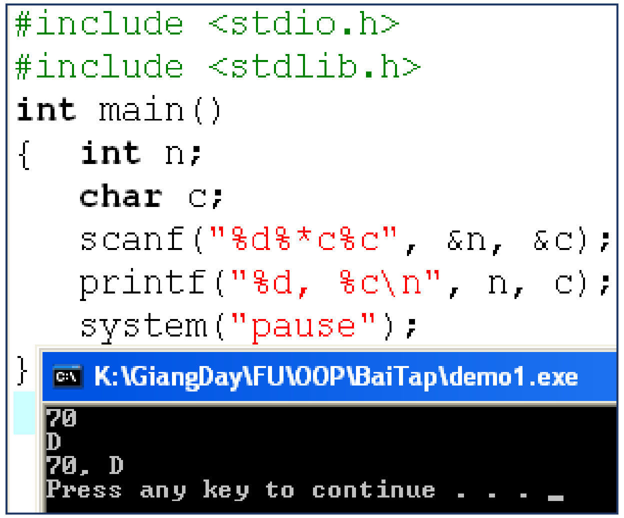
```
#include <stdio.h>
#include <stdlib.h>
int main()
{   int n;
    char c;
    scanf("%d%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

```
K:\GiangDay\FU\OOP\BaiTap\demo1.exe
70
70,
Press any key to continue . . .
```



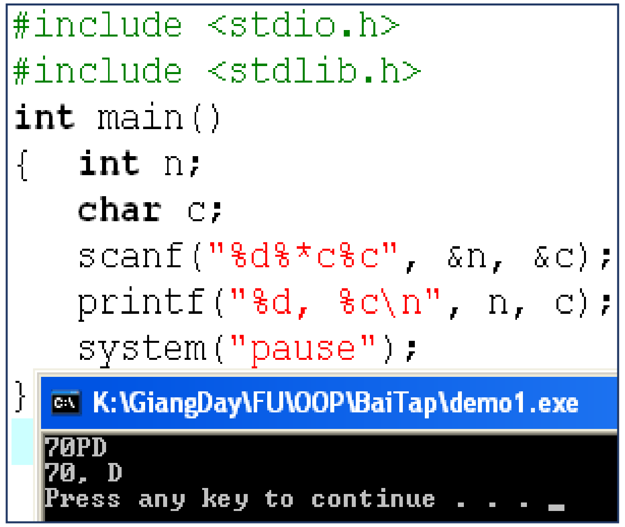
```
#include <stdio.h>
#include <stdlib.h>
int main()
{   int n;
    char c;
    scanf("%d%*c%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

```
K:\GiangDay\FU\OOP\BaiTap\demo1.exe
70 D
70, D
Press any key to continue . . .
```



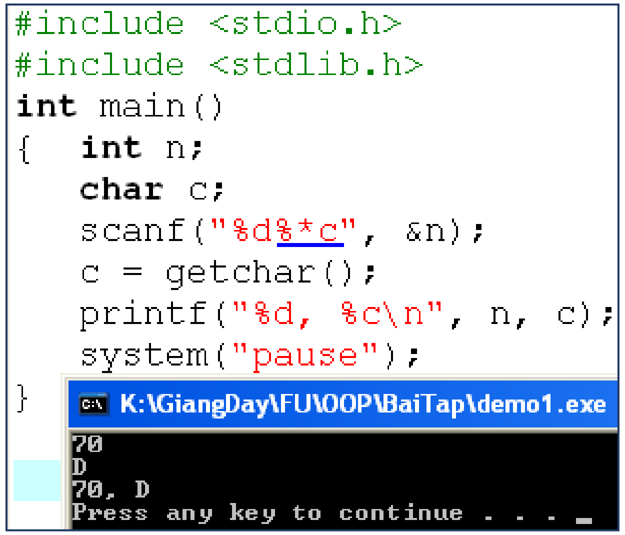
```
#include <stdio.h>
#include <stdlib.h>
int main()
{   int n;
    char c;
    scanf("%d%*c%c", &n, &c);
    printf("%d, %c\n", n, c);
    system("pause");
}
```

```
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");
}
```

```
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");
}
```

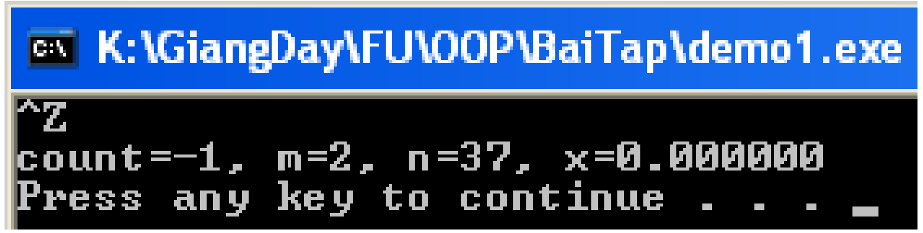
```
K:\GiangDay\FU\OOP\BaiTap\demo1.exe
70
D
70, D
Press any key to continue . . .
```

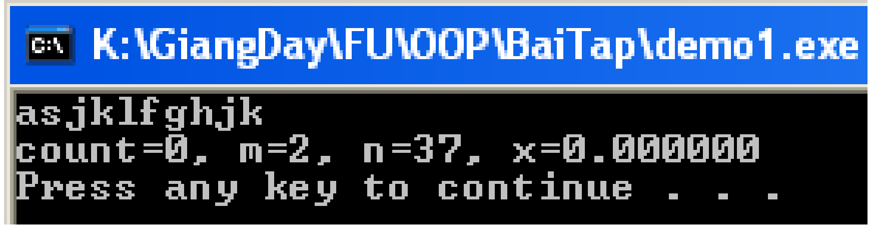
scanf(…) function: Number of data fields are inputted

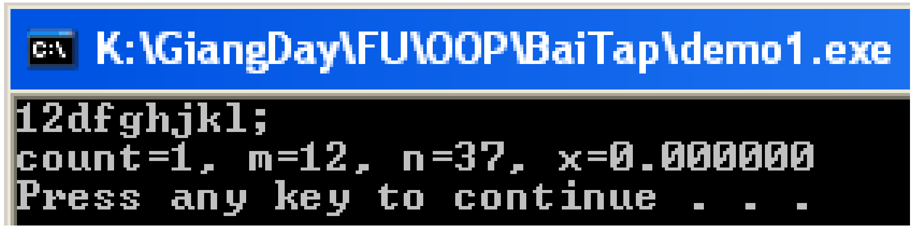
- **Return values from the scanf function:**
  - **scanf** returns the number of addresses successfully filled or EOF.
    - 0 indicates that scanf did not fill any address,
    - 1 indicates that scanf filled the first address successfully,
    - 2 indicates that scanf filled the first and second addresses 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.

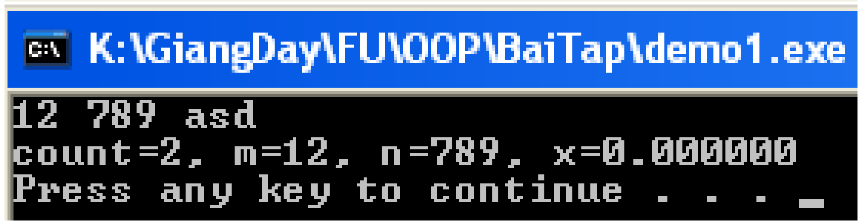
Number of data fields are inputted (cont.)

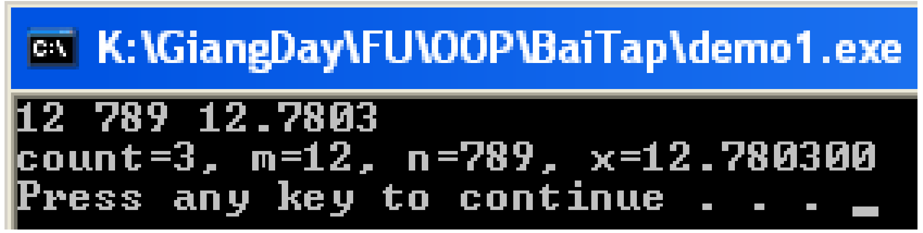
```c
#include <stdio.h>
#include <stdlib.h>
int main()
{   int m, n; double x;
    int count;
    char c;
    count=scanf("%d%d%lf", &m, &n, &x);
    printf("count=%d, m=%d, n=%d, x=%lf\n", count,m,n,x);
    system("pause");
}
```


K:\GiangDay\FU\OOP\BaiTap\demo1.exe
```
^Z
count=-1, m=2, n=37, x=0.000000
Press any key to continue . . .
```


K:\GiangDay\FU\OOP\BaiTap\demo1.exe
```
asjklfghjk
count=0, m=2, n=37, x=0.000000
Press any key to continue . . .
```


K:\GiangDay\FU\OOP\BaiTap\demo1.exe
```
12dfghjkl;
count=1, m=12, n=37, x=0.000000
Press any key to continue . . .
```


K:\GiangDay\FU\OOP\BaiTap\demo1.exe
```
12 789 asd
count=2, m=12, n=789, x=0.000000
Press any key to continue . . .
```

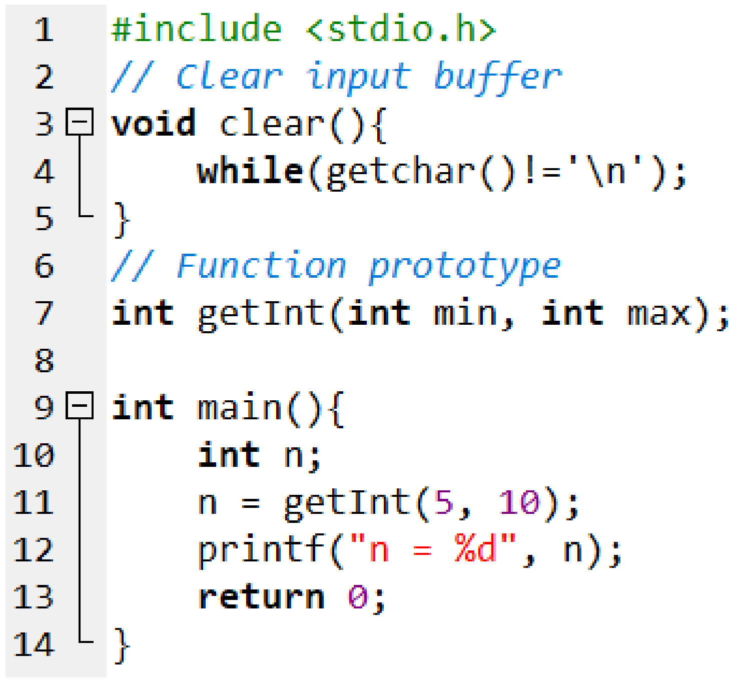

K:\GiangDay\FU\OOP\BaiTap\demo1.exe
```
12 789 12.7803
count=3, m=12, n=789, x=12.780300
Press any key to continue . . .
```

4 - Input Validation

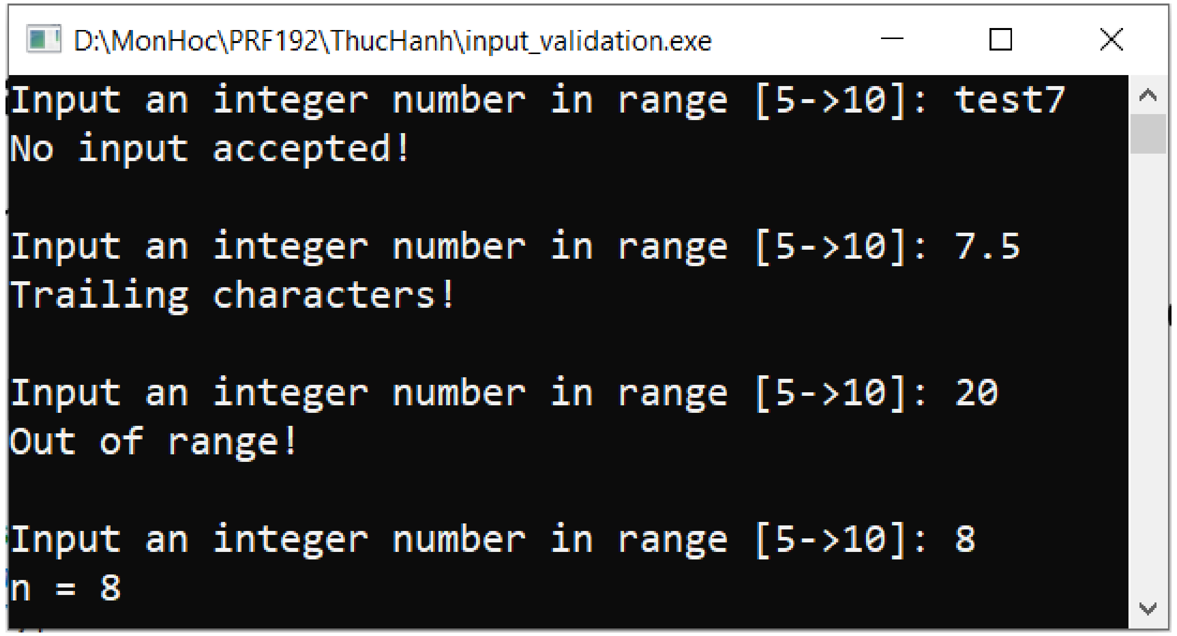
- We cannot predict how the user will input the data values:
  - Whether the user will enter them as requested or not.
  - One user may make a mistake
  - Or another user may simply try to break the program.
- The program should have code for trapping all erroneous input, which includes:
  - Invalid characters
  - Trailing characters
  - Out-of-range input
  - Incorrect number of input fields

Input Validation: Example

- This is a 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)



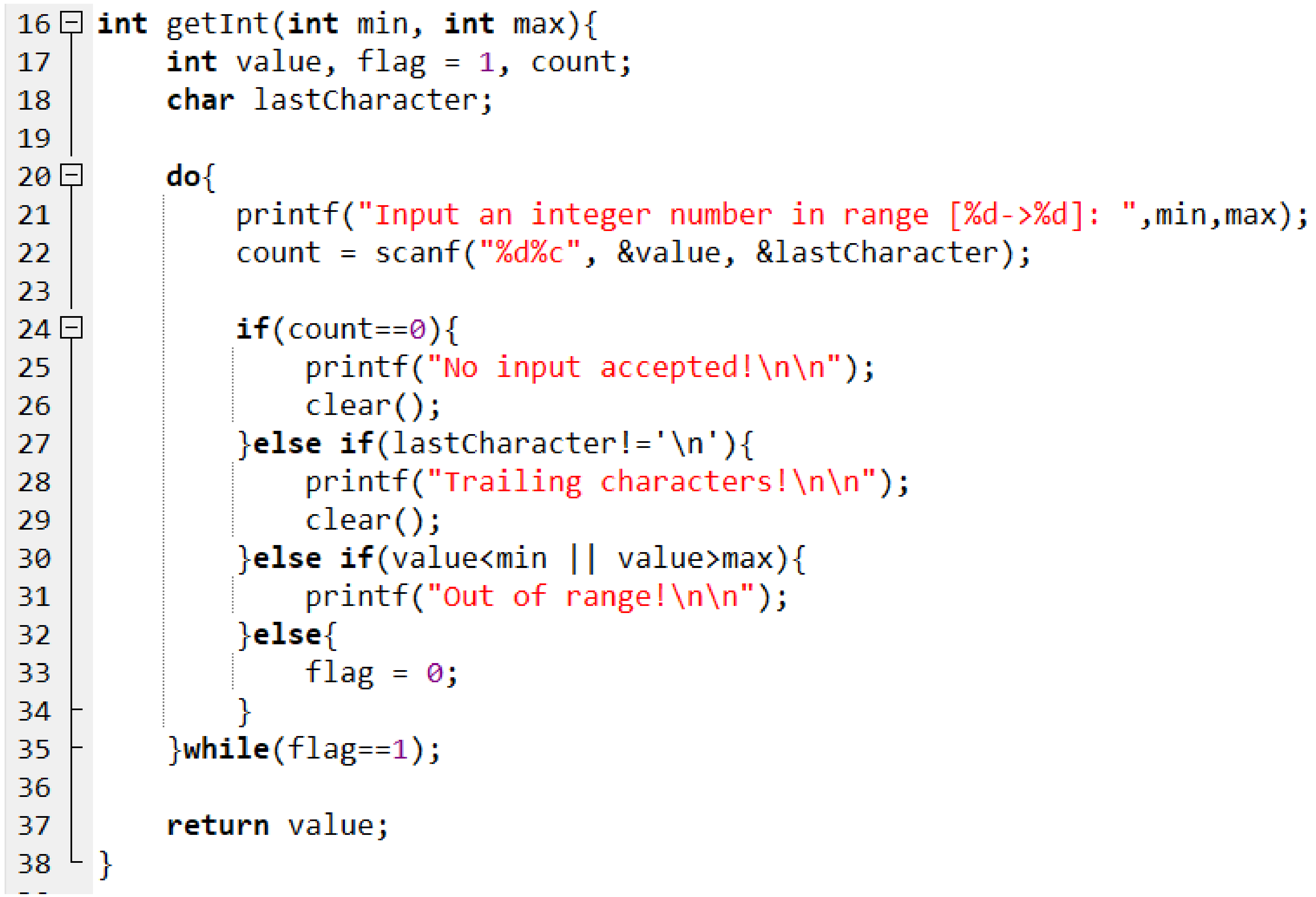
```c
#include <stdio.h>
// Clear input buffer
void clear(){
	while(getchar()!='\n');
}
// Function prototype
int getInt(int min, int max);

int main(){
	int n;
	n = getInt(5, 10);
	printf("n = %d", n);
	return 0;
}
```



D:\MonHoc\PRF192\ThucHanh\input_validation.exe

```
Input an integer number in range [5->10]: test7
No input accepted!

Input an integer number in range [5->10]: 7.5
Trailing characters!

Input an integer number in range [5->10]: 20
Out of range!

Input an integer number in range [5->10]: 8
n = 8
```

Input Validation: Example (cont.)



```c
int getInt(int min, int max){
    int value, flag = 1, count;
    char lastCharacter;

    do{
        printf("Input an integer number in range [%d->%d]: ",min,max);
        count = scanf("%d%c", &value, &lastCharacter);

        if(count==0){
            printf("No input accepted!\n\n");
            clear();
        }else if(lastCharacter!='\n'){
            printf("Trailing characters!\n\n");
            clear();
        }else if(value<min || value>max){
            printf("Out of range!\n\n");
        }else{
            flag = 0;
        }
    }while(flag==1);

    return value;
}
```

Exercise 3: Input Validation

- 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

TRƯỜNG ĐẠI HỌC FPT

Formatted Output

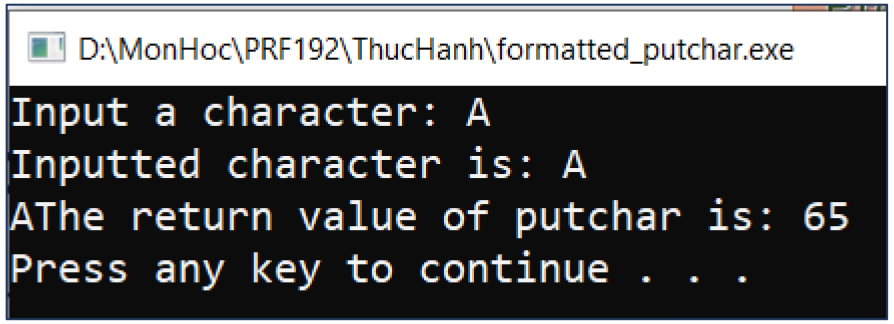
Formatted Output

- 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

putchar(int) function

- **putchar** writes the character received to the standard output stream buffer and returns the character written or **EOF** if an error occurred.
- Syntax: int putchar(int);
- For example: putchar('a');

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

int main()
{
    char ch;
    int code;

    printf("Input a character: ");
    ch = getchar();
    printf("Inputted character is: %c\n", ch);
    code = putchar(ch);
    printf("The return value of putchar is: %d\n", code);

    system("pause");
    return 0;
}
```


D:\MonHoc\PRF192\ThucHanh\formatted_putchar.exe

```
Input a character: A
Inputted character is: A
AThe return value of putchar is: 65
Press any key to continue . . .
```

printf(…) function

- **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)**
- **Note:**
  - 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**

printf(…) function: 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(...) function: Conversion Specifiers

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

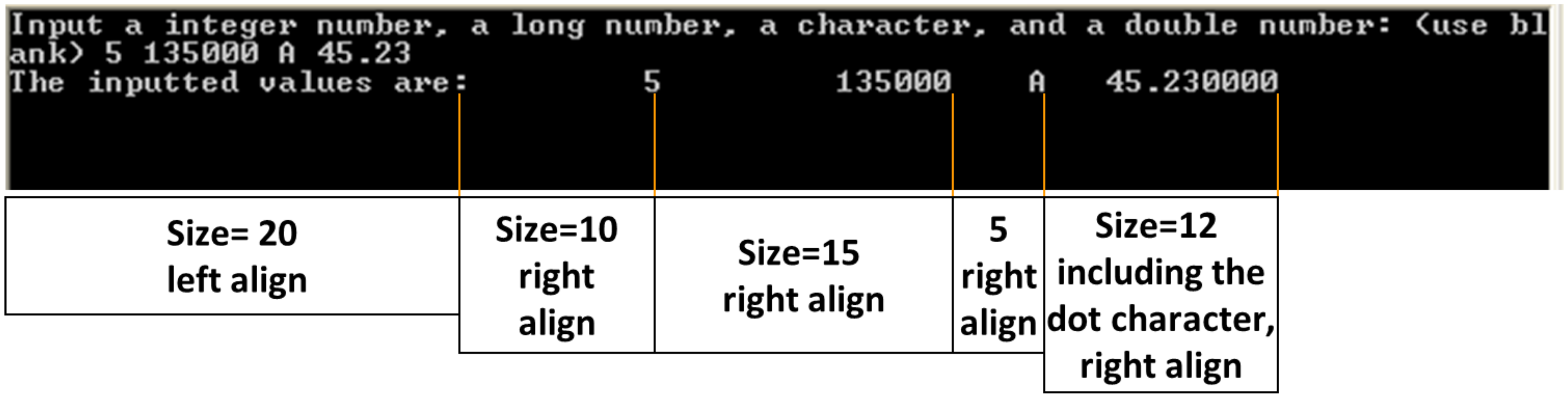
printf(...) function: Example 1

```c
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(...) function: Example 2

```c
#include <stdio.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);
  return 0;
}
```



```
Input a integer number, a long number, a character, and a double number: (use bl
ank) 5 135000 A 45.23
The inputted values are:         5         135000    A   45.230000
```

Summary

Formatted Output

- putchar
- printf

Q&A

Exercise 4:

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. Example:

Input: ca

Output:

```
c: 99, 63h
b: 98, 62h
a: 97, 61h
```

Exercise 5:

Write a C program using the following simple menu:

1- Quadratic equation

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.