Lesson 4
Standard input & variables

Trịnh Thành Trung

trungtt@soict.hust.edu.vn

Topic of this week

- Variables
  - Class Lecture Review
    + Variables
    + Basic data types
    + Constants
    + Standard input.
  - Programming Exercises

Identifiers

- Names of things (variables, functions, etc.)

```
int nMyPresentIncome = 0;
int DownloadOrBuyCD();
```

Identifier naming rules

- Letters, digits, underscores

  `i`

  `CSE_5a`

  `a_very_long_name_that_isnt_very_useful`

  `fahrenheit`

- First character cannot be a digit

  `-5a_CSE` is not valid!

- Case sensitive

  `CSE_5a` is different from `cse_5a`

What are variables?

- A named area in the computer memory, intended to contain values of a certain kind (integers, real numbers, etc.)
- They contain the data your program works with
- They can be used to store data to be used elsewhere in the program
- In short – they are the only way to manipulate data

Variables

- Named region of storage

```
int nRow = 0;
```

- Type (size and meaning of the storage)
- Scope
  - Block
  - Function args
  - Global
  - Be careful not to "hide" a variable
- Lifetime (storage class)
  - Automatic/temporary (block's lifetime)
  - Global (program's lifetime)
  - Local static (program's lifetime)

Variables declaration

Declaration:

- Tells compiler about variables and their type

Syntax

<typename> varname;

e.g:

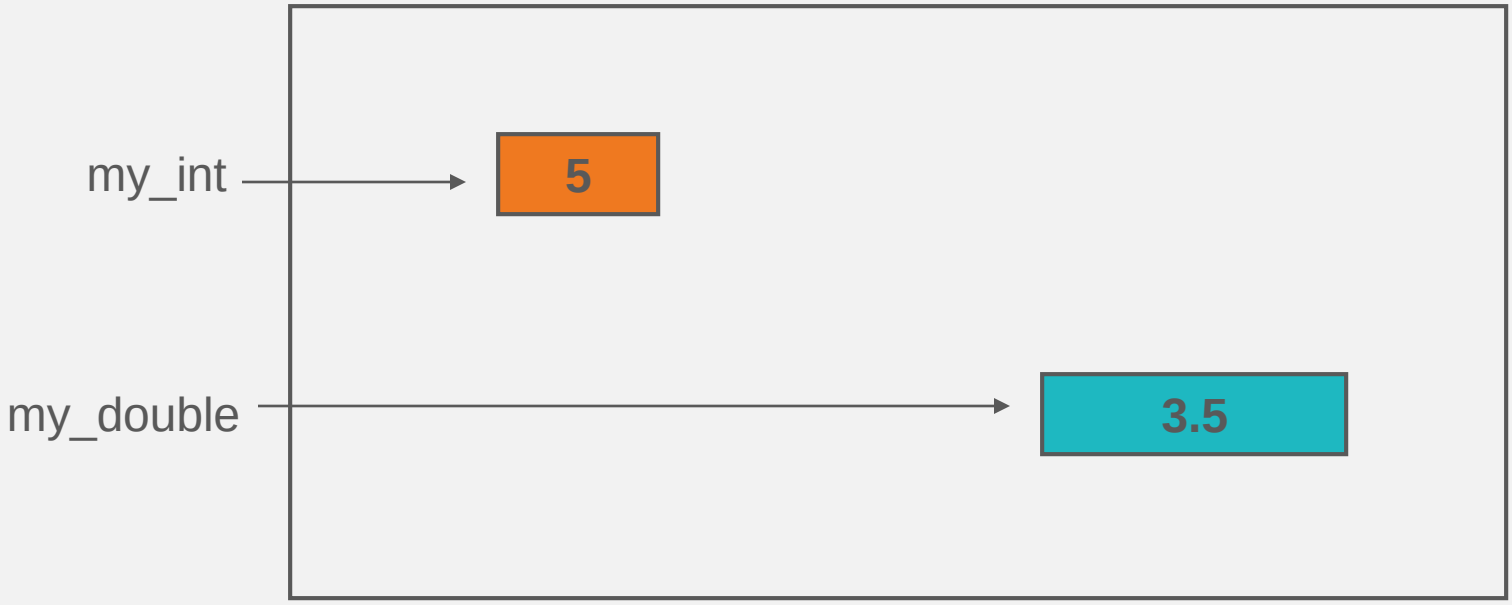
```
int i;
float x, y, z;
char c;
```

Assignment: <varname> = <value>;

```
i = 4;
x = 5.4;
y = z = 1.2;
```

Variables in memory

```
int my_int = 5;
double my_double = 3.5;
```



Declarations, definitions, initialization

- Declarations that reserve storage are called definitions

```
int j;
```

- Definitions may optionally assign a value (initialization)

```
int j = 0;
```

- Declarations specify meaning but may not reserve storage (e.g. *extern*)

```
extern int j;
```

- Release builds typically don't initialize variables by default!

Variable usage:

e.g: `printf("%d + %d = %d\n", a, b, c);`

Example: variable declaration

```
int i;
char c;
float f1, f2;
float f1=7.0, f2 = 5.2;
unsigned int ui = 0;
```

Example

```
#include <stdio.h>

int main()
{
    int a, b, c;
    printf("The first number: ");
    scanf("%d", &a);
    printf("The second number: ");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n",
           a, b, c);
    return 0;
}
```

Constants

- Constant: the value is invariable during the program.

  Declaration constant:

  ```
  #define <constantname> <value>
  ```

  example:

  ```
  #define TRUE 1
  #define FALSE  0
  ```

Constants (1)

- Integer constants

```
31          /* decimal */
037         /* octal */
0x1F        /* hexadecimal */
31L         /* long */
31LU        /* unsigned long */
```

- Float constants

```
123.4       /* double */
123.4F      /* float */
123.        /* double */
123.F       /* float */
123.4L      /* long double */
1e-2        /* double */
123.4e-3    /* double */
```

Constants (2)

- Character constants

```
'K'    /* normal ASCII – 'K' */
'\113' /* octal ASCII – 'K' */
'\x48' /* hexadecimal ASCII – 'K' */
'\n'   /* normal ASCII – newline */
'\t'   /* normal ASCII – tab */
'\\'   /* normal ASCII – backslash */
'\"'   /* normal ASCII – double quote */
'\0'   /* normal ASCII – null (marks end of string)*/
```

- String literals

```
"You have fifteen thousand new messages."
"I said, \"Crack, we're under attack!\"."
"hello," "world" becomes→ "hello, world"
```

Basic data types (1)

- Sizes and limits (may vary for machine)

type	*size in bits*	*range*
`char`	8	-128...127
`short`	16	-32,768...32,767
`int`	32	-2,147,483,648...2,147,483,647
`long`	32	-2,147,483,648...2,147,483,647
`float`	32	10^{-38}...$3x10^{38}$
`double`	64	$2x10^{-308}$...10^{308}

  - `float` has 6 bits of precision (on CUNIX)
  - `double` has 15 bits of precision (on CUNIX)
  - range differs from one machine to another
    + `int` is "native" size

Basic data types (2)

- You can also have unsigned values:

type	*size in bits*	*range*
unsigned char	8	0…255
unsigned short	16	0…65,535
unsigned int	32	0…4,294,967,295
unsigned long	32	0…4,294,967,295

- Look at `/usr/include/limits.h`

Formatting Input with *scanf*

- **`scanf`**
  - Input formatting
  - Capabilities
    - Input all types of data
    - Input specific characters
    - Skip specific characters
- Format

  ```
  scanf(format-control-string, other-arguments);
  ```

  - format-control-string - describes formats of inputs
  - other-arguments - pointers to variables where input will be stored
  - can include field widths to read a specific number of characters from the stream

Formatting Input with `scanf` (II)

Conversion specifier	Description
Integers	
d	Read an optionally signed decimal integer. The corresponding argument is a pointer to integer.
i	Read an optionally signed decimal, octal, or hexadecimal integer. The corresponding argument is a pointer to integer.
o	Read an octal integer. The corresponding argument is a pointer to unsigned integer.
u	Read an unsigned decimal integer. The corresponding argument is a pointer to unsigned integer.
x or **X**	Read a hexadecimal integer. The corresponding argument is a pointer to unsigned integer.
h or **l**	Place before any of the integer conversion specifiers to indicate that a **short** or **long** integer is to be input.
Floating-point numbers	
e, **E**, **f**, **g** or **G**	Read a floating-point value. The corresponding argument is a pointer to a floating-point variable.
l or **L**	Place before any of the floating-point conversion specifiers to indicate that a **double** or **long double** value is to be input.
Characters and strings	
c	Read a character. The corresponding argument is a pointer to **char**, no null (**'\0'**) is added.
s	Read a string. The corresponding argument is a pointer to an array of type **char** that is large enough to hold the string and a terminating null (**'\0'**) character—which is automatically added.
Scan set	
[scan characters	Scan a string for a set of characters that are stored in an array.
Miscellaneous	
p	Read an address of the same form produced when an address is output with **%p** in a **printf** statement.
n	Store the number of characters input so far in this **scanf**. The corresponding argument is a pointer to integer
%	Skip a percent sign (**%**) in the input.

Example of *scanf*

```
int d,m,y,x;
char ch1,ch2;
float f;
scanf("%d", &x);

scanf("%2d%2d%4d", &d,&m,&y);

scanf("%d/%d/%d", &d,&m,&y);

scanf("%c%c", &ch1,&ch2);

scanf("%f", &f);
```

Result

```
4
// x=4
22062007
// d=22, m=6, y=2007
22/06/2007
// d=22, m=6, y=2007
Ab
// ch1='A', ch2='b'
2.3
// f=2.300000
```

Formatting Input with *scanf* (III)

- Scan sets
  - Set of characters enclosed in square brackets `[]`
    - Preceded by `%` sign
  - Scans input stream, looking only for characters in scan set
    - Whenever a match occurs, stores character in specified array
    - Stops scanning once a mismatch is found
  - Inverted scan sets
    - Use a caret `^`: `[^aeiou]`
    - Causes characters not in the scan set to be stored

Formatting Input with *scanf* (IV)

- Skipping characters
  - Include character to skip in format control
  - Or, use * (assignment suppression character)
    - + Skips any type of character without storing it

Example 2

- Reading characters and strings

```c
#include <stdio.h>

int main()
{
   char x, y[ 9 ];

   printf( "Enter a string: " );
   scanf( "%c%s", &x, y );

   printf( "The input was:\n" );
   printf( "the character \"%c\" ", x );
   printf( "and the string \"%s\"\n", y );

   return 0;
}
```

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```

Example 3

- Using an inverted scan set

```
2  #include <stdio.h>
3
4  int main()
5  {
6     char z[ 9 ] = { '\0' };
7
8     printf( "Enter a string: " );
9     scanf( "%[^aeiou]", z );
10    printf( "The input was \"%s\"\n", z );
11
12    return 0;
13 }
```

```
Enter a string: String
The input was "Str"
```

Exercise

- Write a program that read in your name and your studentID, assuming a student ID is in this form
- HUSTYYY: Y- digit
  - HUST123, HUST124
- Use scanset to limit the wrong characters of input.
- After input, display your name and ID to verify.

Example 4

- Reading and discarding characters from the input stream

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int month1, day1, year1, month2, day2, year2;
6
7     printf( "Enter a date in the form mm-dd-yyyy: " );
8     scanf( "%d%*c%d%*c%d", &month1, &day1, &year1 );
9     printf( "month = %d  day = %d  year = %d\n\n",
10             month1, day1, year1 );
10    printf( "Enter a date in the form mm/dd/yyyy: " );
14    scanf( "%d%*c%d%*c%d", &month2, &day2, &year2 );
15    printf( "month = %d  day = %d  year = %d\n",
16             month2, day2, year2 );
17
18    return 0;
19 }
```

```
Enter a date in the form mm-dd-yyyy: 11-18-2000
month = 11  day = 18  year = 2000

Enter a date in the form mm/dd/yyyy: 11/18/2000
month = 11  day = 18  year = 2000
```

Exercises 4.1

- Write a program that reads an integer and a double from user, use a floating-point and an integer variable to store and then show to screen.
  - a) Normal Input
  - b) the integer number is inputted as hexadecimal and the floating point is entered in the scientific format.

Exercises 4.2

- Write and run this program to see the limit of basic data types: int, long.
- Widen this program for other basic data types.
- Use limits.h library to build your programs.

Exercises 4.3

- Write a program that reads a string from the keyboard by using a scan set.

Exercises 4.4

- Write a program that inputs data with a field width.
- Widen to all basic data types.

Exercise 4.5

- Write a program ask user to input the radius of a circle. Use constant for PI.
  - a) Display its area and circumference.
  - b) Now consider the input data is the radius of a sphere. Display its area and volume.

Exercise 4.6

- Write a program that calculates and displays an employee's total wages for week. The regular hours for the work week are 40 and any hours worked over 40 are considered overtime. The employee earns 25000 VND per hour for regular hours, and 40000 VND per hour for overtime hours. This week employee has worked 50 hours.