Lecture 14:

Characters

Prof. Shervin Shirmohammadi
University of Ottawa

Character Variables in C

- A character can be stored into a variable of type `char` which requires only one byte of memory.
- E.g.:

```
char character;
character = 'a';    // note the single quotes, not double!
character = 97;
```

  - Since all characters are represented as binary numbers having an integer equivalent, then the content of a variable of type `char` can be interpreted either as a character or as an integer.
  - For this reason, we can also use variables of type `int` to store characters, though they take more memory.

Representation of Characters

- Many applications require the manipulation of **characters** and character **strings**.
- As we know, all information stored in the computer is represented as binary numbers.
  - Binary numbers are comprised of the digits 0 and 1 only.
  - The representation of a real or integer number as a binary number is handled by C and is completely transparent to us.
  - We must, however, be familiar with the binary representation or integer equivalent of characters in order to manipulate them in C.
- A character can be represented in memory using a binary code.
  - The most commonly used code for representing characters is the **ASCII** (American Standard Code for Information Interchange) code.
  - The ASCII code represents 128 characters and special symbols using a sequence of 7 binary digits.
  - These binary codes have an integer equivalent:
    - The integer equivalent corresponds to the 7 bit binary code.

ASCII Binary Codes

Character	Integer equivalent	Binary code
0	48	0110000
1	49	0110001
2	50	0110010
3	51	0110011
4	52	0110100
5	53	0110101
6	54	0110110
7	55	0110111
8	56	0111000
9	57	0111001
:	58	0111010
;	59	0111011
<	60	0111100
=	61	0111101
>	62	0111110
?	63	0111111
@	64	1000000

ASCII Binary Codes, continued

Character	Integer equivalent	Binary code
A	65	1000001
B	66	1000010
C	67	1000011
D	68	1000100
E	69	1000101
F	70	1000110
G	71	1000111
H	72	1001000
I	73	1001001
J	74	1001010
K	75	1001011
L	76	1001100
M	77	1001101
N	78	1001110
O	79	1001111

ASCII Binary Codes, continued

Character	Integer equivalent	Binary code
P	80	1010000
Q	81	1010001
R	82	1010010
S	83	1010011
T	84	1010100
U	85	1010101
V	86	1010110
W	87	1010111
X	88	1011000
Y	89	1011001
Z	90	1011010
[	91	1011011
\	92	1011100
]	93	1011101
^ (Circumflex)	94	1011110
_ (Underscore)	95	1011111

ASCII Binary Codes, continued

Character	Integer equivalent	Binary code
` (Opening Single Quote)	96	1100000
a	97	1100001
b	98	1100010
c	99	1100011
d	100	1100100
e	101	1100101
f	102	1100110
g	103	1100111
h	104	1101000
i	105	1101001
j	106	1101010
k	107	1101011
l	108	1101100
m	109	1101101
n	110	1101110
o	111	1101111

ASCII Binary Codes, continued

Character	Integer equivalent	Binary code
p	112	1110000
q	113	1110001
r	114	1110010
s	115	1110011
t	116	1110100
u	117	1110101
v	118	1110110
w	119	1110111
x	120	1111000
y	121	1111001
z	122	1111010
{	123	1111011
|	124	1111100
}	125	1111101
~	126	1111110
DEL (Delete/Rubout)	127	1111111

- The characters in the range 1 through 31 inclusively, are special characters.
  - The character **`BEL`** having the integer equivalent 7 can be used to ring the bell on the computer.
- Careful with the representation of the characters `0` through `9`.
  - Their ASCII code does not correspond to their numerical value.
- Note that characters `A` through `Z`, and `a` through `z` have codes that are organized in increasing order of magnitude; note also that the uppercase letters have codes that are smaller than the lowercase letters.

Example

- Write a program that prints the character **A**, once by ASCII code and once by character.

Comparing Characters

- It is possible to compare characters in a logical expression.
  - The ASCII code of the characters are compared.

E.g.:
```
char character_1 = 'a', character_2 = 'b';
    if(character_1 == character_2)
    printf("Same character.\n");

if(character_1 == 'a')
      printf("The letter a has been detected.\n");

if(character_1 < character_2)
    printf("Characters 1 and 2 are ordered.\n");
```

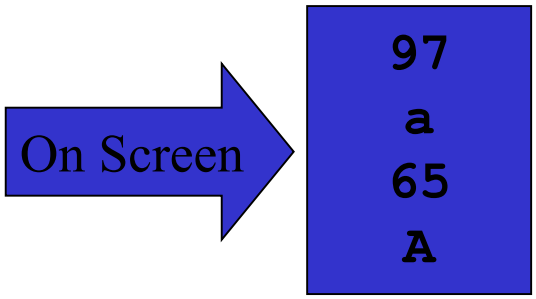
- Recall that the ASCII code is organized such that the equivalent integers of the uppercase and lowercase letters are in increasing order of magnitude.
  - ➔ Alphabetical ordering can be deduced from the relative magnitude of the numerical representations.

Character Output

- Recall that
  - A content of a variable of type `char` can be interpreted as a character or an integer
  - A variable of type `int` containing an integer in the range `0` through `127` inclusively, can be interpreted as being a character or an integer.
  - The content of a variable of type `int` can be interpreted as an integer.
- The conversion specifier `%d` is to be used when we wish to print out the content of a variable as an integer and the conversion specifier `%c` is used when we wish to interpret the variable as a character.

E.g.:

```
char character_1 = 'a';
int integer_1 = 65;
printf("%d\n", character_1);
printf("%c\n", character_1);
printf("%d\n", integer_1);
printf("%c\n", integer_1);
```



Character Input

- It is possible to read in a character from the keyboard using the function `scanf` and the conversion specifier `%c`.
  - E.g.: `scanf("%c", &character_1);`
  - `scanf("%c", &integer_1);`
- The `scanf` reads data by examining all characters typed at the keyboard.
  - The program does not see the characters before the carriage return is hit
  - It is the operating system that reads the characters and puts them in a region of memory called a « buffer ».
- Hardware issue: the input stream.
  - The buffer containing the input data corresponds to the input stream.
  - Consider the entry of a single character:
    - ➢ We type the character on the keyboard and then we hit the enter or carriage return key to signify that we have completed typing in the entry.
      - ✓ This signals the OS that the data can be passed to the program in the buffer of the input stream.
    - ➢ The carriage return inserts the character new line (ASCII equivalent 10) into the input stream after our typed character.
    - ➢ The input stream thus contains two item:
      - o The character typed in and the new line symbol (ASCII equivalent 10).

  ➔ When we read in a character using `scanf`, we must usually clear the input stream to get rid of the new line symbol; this can be achieved by invoking the function `fflush`.

- The input stream is referenced with `stdin`, which is used as the argument in `fflush`.
    - E.g.: `fflush(stdin);`
  - The header file `stdio.h` contains the prototype of `fflush` and the definition of `stdin`
  - `stdin` stands for standard input stream; it is a pointer to the standard input that contains the memory buffer where the keyboard entries are stored by the OS.
- It is good practice to clear the input stream using `fflush` before reading in characters.

Example

- Write a program that continuously prompts the user for a character and terminates only when the user inputs the special character '?'

The getchar and putchar Functions

- The function `getchar` can be used to read in a single character from the input stream.
  - Recall that characters stored in the input stream have been typed on the keyboard.
  - The prototype of `getchar` is found in the header file `stdio.h` and reads:

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

  - thus **`getchar`** returns an integer to the caller and expects no arguments.
  - The integer returned by `getchar` is the ASCII code that corresponds to the next character to be read form the input stream.
  - `getchar` reads characters from the input stream. The characters typed at a keyboard are added to the input stream buffer only after the carriage return key has been typed!
- The function **`putchar`** can be used to print out a single character on the screen.
  - The prototype of `putchar` is found in the header file `stdio.h` and reads:

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

  - thus `putchar` expects an integer as an argument and returns an integer to the caller.
  - The integer sent to `putchar` as an argument is the ASCII code that corresponds to the character to be printed out on the screen.
  - The integer returned by `putchar` is either:
    - the argument itself (we call this an echo) if it was correctly printed on the screen, or
    - the `EOF` symbol if an error was encountered during the execution of `putchar`.

Example

- Write a program that prompts the user for a single lower case character and prints the corresponding upper case equivalent (Use getchar and putchar functions for character input/output).