

Review

- Pointer
 - Declaration
 - Swap (Call by Reference)
- Array
 - Single data type, fixed size
 - Relationship with Pointer
 - Passing by Reference
 - Multi-dimensional Array

I/O in C

Lecture 31

Hyung-Sin Kim



SNU Graduate School of Data Science

Contents

- **I/O**
 - **Buffered I/O**
 - **Formatted I/O**



I/O

One Character I/O

- putchar – print a single character (an input is assumed to be an ASCII value)
 - `char c = 'h';`
 - `putchar(c);`
 - `putchar('h');`
 - `putchar(104);`
- getchar – get a single character input (returns its ASCII value)
 - `char c;`
 - `c = getchar();`

Buffered I/O

- Reading each character right after a keyboard button is pushed is error-prone
 - To address the problem, keyboard input is buffered until we press Enter
 - Before Enter is pressed, you can modify the input stream freely
 - When Enter is pressed, the whole buffered input **including** ‘\n’ is delivered to the program
- Output is buffered too.
 - Output from putchar is buffered until it prints out ‘\n’

Buffered I/O – Example

- `#include <stdio.h>`
-
- `int main(void) {`
- `char c1;`
- `char c2;`
- `printf("Input char1:\n");`
- `c1 = getchar();`
- `printf("Input char2:\n");`
- `c2 = getchar();`
- `printf("char1 is %c and char2 is %c\n", c1, c2);`
- `return 0;`
- `}`

Let's type, compile, and execute!

What do you see on your screen? 😊

We want to I/O more various data types!

Formatted I/O – printf

- **printf** prints out ASCII text embedded with values
 - In doing so, it must convert any non-ASCII value, such as integer, into an ASCII value
- `printf(“format string”, values);`
 - Format string consists of normal characters, special characters, and conversion specifications
 - `printf` examines each character in the format string sequentially
 - If the character is a normal character, it simply print this out
 - If the character is ‘%’, it recognizes a conversion specification, such as `%d`. Then, the next character indicates how the next pending parameter should be interpreted
 - If the character is ‘\’, it recognizes a special character, such as ‘\n’

Formatted I/O – printf

- printf conversion specifications

printf Conversions	Printed as
%d, %i	Signed decimal
%o	octal
%X, %x	Hexadecimal (a-f or A-F)
%u	Unsigned decimal
%c	Single character
%s	String, terminated by \0
%f	Floating point in decimal notation
%e, %E	Floating point in exponential notation
%p	Pointer

Formatted I/O – scanf

- scanf reads formatted ASCII data and **converts** it into another data type, if needed
 - In doing so, it must convert an ASCII value to a relevant non-ASCII value, such as integer
- scanf(“format string”, &variable or pointer);
 - As that in printf, format string consists of normal characters, special characters, and conversion specifications
 - scanf format string represents the format of the input stream
 - If the character is a normal character, it simply expects the next received ASCII value to be the same character and ignores it
 - If the character is ‘%’, it recognizes a conversion specification, such as %d. Then, the next character indicates how the next pending parameter should be interpreted
 - If the character is ‘\’, it recognizes a special character, such as ‘\n’

Formatted I/O – scanf

- Examples
 - `scanf(“%d”, &a);`
 - Convert a sequence of **non-white space characters** into an integer and store the value in **a**
 - White space: space, tab, new line, carriage return ...
 - `scanf(“%d/%d”, &a, &b);`
 - Convert a sequence of **non-white space characters** into an integer and store the value in **a**
 - Expect to receive ‘/’ and ignore it
 - Convert the next sequence of **non-white space characters** into an integer and store the value in **b**
 - `scanf(“%c”, &a);`
 - Store any **single character** (including white space characters) in **a**

Thanks!