Introduction to C

Standard Input and Output

Instructor: Yin Lou

02/11/2011

stdio.h

- Each source file that refers to an input/output library function must contain the line
  - #include <stdio.h>

stdio.h

- Each source file that refers to an input/output library function must contain the line
  - #include <stdio.h>
- When the name is bracketed by < and > a search is made for the header in a standard set of places
  - On Unix systems, typically in the directory /usr/include

Example

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

int main()
{
    int c;
    while ((c = getchar()) != EOF)
    {
        putchar(tolower(c));
    }
    return 0;
}
```

Formatted Output - printf

```
int printf(char *format, arg1, arg2, ...);
```

Formatted Output - printf

```
int printf(char *format, arg1, arg2, ...);
```

- printf converts, formats, and prints its arguments on the standard output under control of the format.
- It returns the number of characters printed.
- printf("x = %d, y = %d\n", x, y);

Basic printf Conversions

- d, i
  int; decimal number
- o
  int; unsigned octal number (without a leading zero)
- x, X
  int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10,...,15
- u
  int; unsigned decimal number
- c
  int; single character
- s
  char *; print characters from the string until a '\0' or the number of characters given by the precision.
- f
  double; [-]*m.dddddd*, where the number of *d*'s is given by the precision (default 6).

What Can Go Wrong?

```
printf(s);
```

What Can Go Wrong?

```
printf(s);
```

```
char *s = "My name is %s";
printf(s); // equivalent to printf("My name is %s");
printf("%s", s); // safe
```

Print to String

```
int sprintf(char *string, char *format, arg1, arg2, ...);
```

Print to String

```
int sprintf(char *string, char *format, arg1, arg2, ...);
```

- sprintf formats the arguments in arg1, arg2, etc., according to format as before, but places the result in string instead of the standard output
- string must be big enough to receive the result
- sprintf(s, "x = %d, y = %d\n", x, y);

Formatted Input - scanf

```
int scanf(char *format, ...);
```

Formatted Input - scanf

```
int scanf(char *format, ...);
```

- scanf reads characters from the standard input, interprets them according to the specification in format, and stores the results through the remaining arguments.
- Each of the other argument must be a pointer

Formatted Input - scanf

```
int scanf(char *format, ...);
```

- scanf reads characters from the standard input, interprets them according to the specification in format, and stores the results through the remaining arguments.
- Each of the other argument must be a pointer
- It returns as its value the number of successfully matched and assigned input items. On the end of file, EOF is returned. Note that this is different from 0, which means that the next input character does not match the first specification in the format string.
- The next call to scanf resumes searching immediately after the last character already converted.
- scanf("%d", &x);
- scanf("%s", name); // char name[1000];

Read from a String

```
int sscanf(char *string, char *format, arg1, arg2, ...);
```

Read from a String

```
int sscanf(char *string, char *format, arg1, arg2, ...);
```

The formate string main contain

- Blank or tabs, which are not ignored
- Ordinary characters
- Conversion specifications
- sscanf("x = 5, y = 6", "x = %d, y = %d", &x, &y);

Example

```
#include <stdio.h>

int main()
{
    double sum, v;
    while (scanf("%lf", &v) == 1)
    {
        printf("\t%.2f\n", sum += v);
    }
    return 0;
}
```

Error Handling - stderr and exit

- There is a second output stream, called stderr.
- Output written on stderr normally appears on the screen even if the standard output is redirected.
- exit(int signal) is function to terminate the program, with a signal specified. 0 signals that all is well; non-zero values usually signal abnormal situations

Error Handling - stderr and exit

- There is a second output stream, called stderr.
- Output written on stderr normally appears on the screen even if the standard output is redirected.
- exit(int signal) is function to terminate the program, with a signal specified. 0 signals that all is well; non-zero values usually signal abnormal situations

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

main()
{
    char *p = (char *) malloc(50 * sizeof(char));
    if (p == NULL)
    {
        fprintf(stderr, "Not enough memory!\n");
        exit(1);
    }
    // do something
    exit(0);
}
```