Introduction to C

File and Variable-length Argument Lists

Instructor: Yin Lou

02/14/2011

Streams

- In many programming languages, input/output are done in streams
- Data exists on the stream, you consume part of it and move on
- Examples:
  - stdout: standard output stream
  - stderr: standard error output stream
  - stdin: standard input stream
  - files
  - network sockets (network connections)

File I/O

- Before a file can be read or written, it has to be *opened* by the library function fopen. fopen takes an external name like `data.txt`, does some housekeeping and negotiation with the operating system, and returns a pointer to be used in subsequent reads or writes of the file.

File I/O

- Before a file can be read or written, it has to be *opened* by the library function fopen. fopen takes an external name like `data.txt`, does some housekeeping and negotiation with the operating system, and returns a pointer to be used in subsequent reads or writes of the file.
- This pointer points to a structure `FILE`.

File I/O

- Before a file can be read or written, it has to be *opened* by the library function fopen. fopen takes an external name like `data.txt`, does some housekeeping and negotiation with the operating system, and returns a pointer to be used in subsequent reads or writes of the file.
- This pointer points to a structure `FILE`.

```
FILE *fp;
```

fopen

```
FILE *fopen(char *name, char *mode);
```

- fopen returns a pointer to a FILE.
- FILE is a type name. It is defined with a `typedef`.
- mode can be
  - `r` - read
  - `w` - write
  - `a` - append
  - a "b" can be appended to the mode string to work with binary files. For example, "rb" means reading binary file.

Text Files - fprintf and fscanf

```
int fprintf(FILE *fp, char *format, ...);
int fscanf(FILE *fp, char *format, ...);
```

- Similar to printf and scanf.

Example

```
#include <stdio.h>

int main(int argc, char **argv)
{
    FILE *fp;
    int i, n;
    float value;

    fp = fopen(argv[1], "r");
    fscanf(fp, "%d", &n);
    for (i = 0; i < n; ++i)
    {
        fscanf(fp, "%f", &value);
        printf("%f\n", value);
    }
    return 0;
}
```

Binary Files - fread and fwrite

```
size_t fread(void *array, size_t size, size_t count, FILE *fp);
size_t fwrite(void *array, size_t size, size_t count, FILE *fp);
```

Binary Files - fread and fwrite

```
size_t fread(void *array, size_t size, size_t count, FILE *fp);
size_t fwrite(void *array, size_t size, size_t count, FILE *fp);
```

- array must be a pointer
- size - size of elements in this array
- count - number of elements in this array

Example

```
#include <stdio.h>

int main(int argc, char **argv)
{
    FILE *fp;
    int i, n;
    float value[3];

    fp = fopen(argv[1], "rb");
    fread(&n, sizeof(int), 1, fp);
    for (i = 0; i < n; ++i)
    {
        fread(value, sizeof(float), 3, fp);
        printf("%f\t%f\t%f\n", value[0], value[1], value[2]);
    }
    return 0;
}
```

Variable-length Argument Lists

- Let's implement a minimal version of printf.
- Use "..." to indicate that the number and types of these arguments may vary.
- The declaration "..." can only appear at the end of an argument list.

Variable-length Argument Lists

- Let's implement a minimal version of printf.
- Use "..." to indicate that the number and types of these arguments may vary.
- The declaration "..." can only appear at the end of an argument list.

```
void minprintf(char *format, ...)
```

What Do We Need?

- type va_list - declare a variable that will refer to each argument in turn, assume ap
- macro va_start - initialize ap to point to the first unnamed argument
- function va_arg - Each call of va_arg returns one argument and steps ap to the next; va_arg uses a type name to determine what type to return and how big a step to take.
- function va_end - do whatever cleanup is necessary. It must be called before the program returns.

Implementation

```
#include <stdarg.h>

void minprintf(char *format, ...) {
    va_list ap; // points to each unnamed arg in turn
    char *p, *sval;
    int ival;

    va_start(ap, format); // make ap point to 1st unnamed arg
    for (p = format; *p; ++p) {
        if (*p != '%') {
            putchar(*p);
            continue;
        }
        switch (*++p) {
            case 'd':
                ival = va_arg(ap, int);
                printf("%d", ival);
                break;
            case 's':
                for (sval = va_arg(ap, char *); *sval; ++sval) {
                    putchar(*sval);
                }
                break;
            default:
                putchar(*p);
        }
    }
    va_end(ap); // clean up when done
}
```