CSC 270 – Survey of Programming Languages

C Lecture 8 – Input and Output

Input and Output

- C supports 2 different I/O libraries:
  - buffered (higher level functions supported by ANSI standards)
  - unbuffered (lower-level functions corresponding to UNIX systems calls)
- We will work with the buffered I/O funcitons only.

Standard Input, Output and Error

- All C programs running under UNIX have three standard files opened without any special commands:
  - **`stdin`** – standard input (usually the keyboard)
  - **`stdout`** – standard output (usually the monitor)
  - **`stderr`** – standard error (usually the monitor)
- All of these can be redirected to a file under Linux.

Redirection and Piping

- All three standard files can be redirected:
- The command
  - **`prog > outfile`** redirects output to a file named **`outfile`**, replacing it if it already exists.
  - **`prog >> outfile`** redirects output to a file named **`outfile`**, appending to the end if it already exists.
  - **`prog < infile`** redirect input to a file named **`infile`**.
  - **`prog | prog2`** – makes prog's standard output prog2's standard input. This is called ***piping***.

tolower.c

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

/*
 * main() - Convert input to lower case
 */
int     main(void)
{
        int c;

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

Formatted I/O

- **`printf()`** and **`scanf()`** both use a control string as its first argument. The can be either a constant or a variable:

```
char    s[] = "hello, world\n";
printf(s);
```

- It matches specifications beginning with `%` (except for `%%`) with an argument in the list of parameters that follows the control string.

Specifiers

- All specifiers are of the form:

  `%±w.pt` **`w`** = minimum width

  `p` = precision (decimal places)

  `t` = data type

Specifiers – An Example

- If our output is "**`Hello, world`**"

Control String	Printed as (output appears between the colons)
`%s`	`:Hello, world:`
`%10s`	`:Hello, world:`
`%15s`	`:   Hello, world:`
`%-15s`	`:Hello, world     :`
`%15.10s`	`:     Hello, wor:`
`%-15.10s`	`:Hello, wor     :`
`%.10s`	`:Hello, wor:`

Specifiers

Specifier types:

`d, i`	integer in decimal format
`o`	integer in octal format
`x`	integer in hexadecimal format
`u`	unsigned integer in decimal format
`c`	integer in character format
`s`	`char *`; prints all characters until the `'\0'`
`f`	`double [-]m.dddddd` – by default 6 decimal places
`e, E`	`double [-]m.ddddddexx or mddddddExx`
`g, G`	`double;` uses (e, E) if exponent is less than -4 or greater than or equal to precision; otherwise uses `f` format.

`scanf()`

- **`scanf()`** uses the same control string as printf but for different purposes. It seeks all characters and skips them before matching argument to a specifier.

scan.c

```
#include        <stdio.h>

int     main(void)
{
        int     x;
        scanf("10%d", &x); /* don't forget the &  */
        printf("x = %d\n", x);

        return(0);
}
```

If I input the line
`10   1`
It will print `1`

sprintf()

sprintf(char *string, char *format, arg1, arg2, ..)

places formatted output into string instead of into standard output.

e.g.,

```
sprintf(s, "X = %d\n", x = 8);
/* s now contains "x = 8\n"
```

sscanf()

- Also, **sscanf()** take its input from a string

```
#include        <stdio.h>
#define         MAXCHAR         80

int    main(void)
{
   int      x, y, z;
   char     s[MAXCHAR], t[MAXCHAR];

   sscanf("x = 654y = 489z = 22", "x = %dy = %dz = %d",
              &x, &y, &z);

   sprintf(t, "i = %d\tj = %d\tk = %d\n", y, z, x);
   printf(t);

   return(0);
}
```

Files

- Files are accessed by pointer to them; files pointers are defined by

  ```
  FILE   *fp;
  ```

- We open a file by writing

  ```
  fp = fopen(filename, t);
  /* Both arguments are strings */
  ```

- **t** is a string containing the mode for which the file is opened.

Opening Files

Legal modes include:

`"r"`	open file for reading text
`"w"`	open file for writing text – older version of the file is destroyed
`"a"`	open file for appending text – output is appended to the end of the existing file
`"rb"`	open for reading a binary file
`"wb"`	open for writing a binary file
`"ab"`	open for appending a binary file

Closing Files

- We close with
  ```
  fclose(fp);
  ```
- For text files, we can replace stream I/O functions with file function:

`putchar(c);`	`putc(c, fp);`
`getchar();`	`getc(fp);`
`printf("…", …);`	`fprintf(fp, "…", …);`
`scanf("…", …);`	`fscanf(fp, "…", …);`
`gets(s);`	`fgets(fp, s)`
`puts(s);`	`fputs(fp, s);`

tolower2.c

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

#define     MAXCHAR         80

/*
 * main() - Convert input to lower case
 */
int   main(int argc, char *argv[])
{
  FILE    *ifp, *ofp;
  int     c;
  char    ifilename[MAXCHAR], ofilename[MAXCHAR];
```

```
  /*
   *  The input file's name or both file names may
   *  be given as command line arguments
   */
  switch(argc)    {
  case 3:   /* Just copy the names */
            strcpy(ofilename, argv[2]);
            strcpy(ifilename, argv[1]);
            break;

  case 2:   /* Copy input file name
                       - get output file name */
            strcpy(ifilename, argv[1]);
            printf("Enter output file name\t?");
            scanf("%s", &ofilename);
            break;
```

```
    case 1:   /* Get both file names */
              printf("Enter input file name\t?");
              scanf("%s", &ifilename);
              printf("Enter output file name\t?");
              scanf("%s", &ofilename);
              break;

    default: fprintf(stderr,
    "usage: tolower <input file> <output file>\n");
              exit(1);
    }

    /* Open the input file */
    if ((ifp = fopen(ifilename, "r")) == NULL)    {
        fprintf(stderr, "Cannot open %s\n",
                    ifilename);
        exit(2);
    }
```

```
    /* Open the output file */
    if ((ofp = fopen(ofilename, "w")) == NULL)    {
        fprintf(stderr, "Cannot open %s\n",
                     ofilename);
        exit(2);
    }

    /* Copy the file in lower case */
    while ((c = getc(ifp)) != EOF)
        putc(tolower(c), ofp);

    /* Close both files and terminate */
    fclose(ifp);
    fclose(ofp);
    return(0);
}
```

Reading Binary Files

- Binary files are stored in internal representations. We use structures in part to structure file records.

```
size_t fread(void *ptr,
   size_t object_size,
   size_t n_objects,
   FILE *fp)
```

- reads *n* data objects from the file to which **`fp`** points and stores them at the location to which **`ptr`** points.
- **`fread`** returns the number of objects read.

Writing Binary Files

```
size_t fwrite(void *ptr,
   size_t object_size,
   size_t n_objects,
   FILE *fp)
```

- writes *n* data objects in the file to which **`fp`** points and takes them from the location to which **`ptr`** points.
- **`fwrite`** returns the number of objects written

writerec.c

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

#define     MAXLINE     100

typedef struct  {
  char      first[11], initial, last[16];
  char      address[31], city[13], state[3], zip[6];
  int       balance;
} personrec;

int   getline(FILE *ifp, char s[], int lim);
FILE  *ifp, *ofp;
```

```
int   main(void)
{
  personrec person;
  char      filename[26], line[MAXLINE];

  printf("Enter input file name\t?");
  scanf("%s", &filename);
  if ((ifp = fopen(filename, "r")) == NULL)     {
      printf("Cannot open %s\n");
      exit(1);
  }

  printf("Enter output file name\t?");
  scanf("%s", &filename);
  if ((ofp = fopen(filename, "wb")) == NULL)    {
      printf("Cannot open %s\n", filename);
      exit(1);
  }
```

```
    /* Keep reading records and display them */
    while (getline(ifp, line, MAXLINE) > 0) {
        /* Because address and city can contain a
         * space read up until the ! and then skip it
         */
        sscanf(line, "%s %c%s %[^!]!%[^!]!%s%s%d",
              &person.first, &person.initial,
              &person.last, &person.address,
              &person.city, &person.state,
              &person.zip, &person.balance);

        /* Display the record read */
        printf("\\%s\\%c\\%s\\%s\\%s\\%s\\%s\\%d\\\n",
              person.first, person.initial,
              person.last, person.address,
              person.city, person.state,
              person.zip, person.balance);
```

```
        fwrite(&person, sizeof(person), 1, ofp);
    }
    return(1);
}

/*
 * getline() -  Get a line of input, read it into s
 *              and return its length
 */
int     getline(FILE *ifp, char s[], int lim)
{
    int     c, i;

    /*
     *  We will read as long as we haven't exceeded
     *  the maximum characters, reached the end of
     *  the line or ran out of input
     */
```

```
  for (i = 0;  i < lim - 1
            && (c = getc(ifp)) != EOF
            && c != '\n'; )
      s[i++] = c;
  if (c == '\n')
      s[i++] = c;

  s[i] = '\0';
  return(i);
}
```

readrec.c

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

#define     MAXLINE     100

typedef struct  {
  char      first[11], initial, last[16];
  char      address[31], city[13], state[3], zip[6];
  int       balance;
} personrec;

FILE    *ifp, *ofp;
```

```
/*
 * main() - Convert a binary file into a text
 *          file
 */
int   main(void)
{
  personrec person;
  char      filename[26], line[MAXLINE];

  /*
   * Get the names of the input and output files
   * and open them */
  printf("Enter input file name\t?");
  scanf("%s", &filename);
  if ((ifp = fopen(filename, "rb")) == NULL)    {
      printf("Cannot open %s\n");
      exit(1);
  }
```