Lecture #3: Pointers

Reading: Holmes, Chapters 8 and 9
Handouts: Pointers, Confusion, and All That; num.c; prenv.c
Homework: Adssignment #2, due Thursday, October 16, at 6PM

---

1. Greetings and felicitations
2. Review precedence table quickly
3. Pointers and addresses
   - draw this, show simple example: for(i = 0, p = x; *p; i++, p++);
   - * operator, & operator
   - pointer arithmetic; why p - x is only legal operation; discuss p - 1, p + 1
4. Pointers and Arrays
   - rework above example as arrays
   - describe: pointer constant (array name), pointer variable
   - difference between int *x and int x[1]
   - multiple dimensions: p = &a[0][0] with int a[2][3]
5. Multiply indirect pointers
   - show array of pointers
   - walk through (and draw) pointer confusion
6. Strings
   - NUL byte
   - show strcat, strcpy (do shortest versions of this), index
   - do sprintf(buf, “%s: check out line %d”, filename, lineno);
7. Command-line interface as example of pointer args
   - argv, argc; show num x y z
   - envp; show printenv PATH
8. Arrays vs. Pointers again
   - allocation of arrays
   - malloc, free for pointers
   - realloc: warn about p = realloc(p, 10000)

Pointers, Confusion, and All That

This very short, very confusing program shows the type of code that programmers who become addicted to pointers in C can write. It also is an excellent exercise in using pointers; if you can read this and figure out what it prints, you will be able to read and understand (almost) any use of C pointers! This exercise is from Alan Feuer's marvelous book *The C Puzzle Book*.

```
char *c[] = {
       "ENTER",
       "NEW",
       "POINT",
       "FIRST"
};

char **cp[] = {  c+3, c+2, c+1, c };
char ***cpp = cp;

main()
{
       printf("%s", **++cpp );
       printf("%s ", *--*++cpp+3 );
       printf("%s", *cpp[-2]+3 );
       printf("%s\n", cpp[-1][-1]+1 );
}
```

num.c

This program prints a file, putting the file name and line number in front of each line. For example, if the file hello contains:

```
hi there
how are you?
```

the command

```
num hello
```

prints

```
hello,   1:hi there
hello,   2:how are you?
```

And now, the program!

```
/*
 * program to print a file with file name and line number
 * prefixing each line
 * invocation:
 *    num                 print stdin with line numbers
 *    num arg1 ... print contents of arg1 with line numbers
 * exit code: number of files not opened
 * author: Matt Bishop, bishop@cs.ucdavis.edu, 9/16/96
 */
#include <stdio.h>
#include <stdlib.h>

/*
 * print the given file, with the file name and line
 * number in front
 * arguments:      fn     pointer to file name
 *          fp     file pointer
 */
void cat(char *fn, FILE *fp)
{
      register int c;                    /* input character */
      register int lno = 0;              /* line number */
      register int nlstate = 1;  /* 1 if last char a newline */

      /*
       * read the file
       */
      while((c = getc(fp)) != EOF){
            /* was the previous char a newline */
            if (nlstate){
                  /* one more line */
                  lno++;
                  /* print the line number and file name */
                  if (fn == NULL) printf("%4d: ", lno);
                  else          printf("%s,%4d: ", fn, lno);
                  /* we now clear the record of the newline */
                  nlstate = 0;
            }
            /* print the character */
            putchar(c);
            /* if it's a newline, set the flag */
            if (c == '\n')
                  nlstate = 1;
      }
```

```
}

/*
 * the main routine
 */
int main(int argc, char *argv[])
{
      register int nerr = 0;     /* nuber of files not opened */
      register char **a;  /* used to walk argument list */
      register FILE *fp;  /* pointer to file being processed */

      /*
       * no argument -- use stdin
       */
      if (argc == 1){
            cat(NULL, stdin);
            return(EXIT_SUCCESS);
      }

      /*
       * walk the arg list, doing each file
       */
      for(a = &argv[1]; *a != NULL; a++)
            /* open the file */
            if ((fp = fopen(*a, "r")) == NULL){
                  /* oops ... say what happened */
                  perror(*a);
                  nerr++;
            }
            else{
                  /* print the file */
                  cat(*a, fp);
            }

      /*
       * status is the number of files not opened
       */
      return(nerr);
}
```

prenv.c

This program prints all environment variables that are exported to the subprocess. If no arguments are given, it prints all the environment variables and their values; if arguments ar givn, it prints the values of those environment variables.

```
/*
 * program to print environment variables
 * invocation:
 *    prenv        print argument list
 *    prenv arg1 ...      print value of argument(s)
 * exit code: number of environment variables not found
 * author: Matt Bishop, bishop@cs.ucdavis.edu, 9/16/96
 */
#include <stdlib.h>
#include <strings.h>
#include <stdio.h>

int main(int argc, char *argv[], char *envp[])
{
      register int i;            /* counter in for loops */
      register int narg;  /* number of current argument */
      register int len;   /* length of current argument */
      register int found; /* 1 if env. variable found */
      register int exstat = 0;   /* exit status code */

      /*
       * no arguments; just print the environment
       * variables and their values
       */
      if (argc == 1){
            /* just loop and print */
            for(i = 0; envp[i] != NULL; i++)
                  printf("%s\n", envp[i]);
            /* all done! */
            return(EXIT_SUCCESS);
      }

      /*
       * arguments given; just print the values
       * associated with these named variables
       */
      for(narg = 1; argv[narg] != NULL; narg++){
            /* just an optimization ... */
            len = strlen(argv[narg]);
            /* assume no such variable */
            found = 0;
            /*
             * now look for the variable
             */
            for(i = 0; envp[i] != NULL; i++)
                  /* see if this one is it*/
                  if (strncmp(envp[i], argv[narg], len) == 0){
                        /* name= means value follows = */
                        if (envp[i][len] == '='){
                              printf("%s\n", envp[i] + len + 1);
                              found++;
```

```
                        }
                        /* name means value is empty */
                        /* anything else, it doesn't match */
                        else if (!envp[i][len]){
                                putchar('\n');
                                found++;
                        }
                }
        /*
         * did we find it?
         */
        if (!found){
                /* nope -- print error message */
                fprintf(stderr, "%s: no such variable\n",
                                                        argv[narg]);
                /* one more unknown environment variable */
                exstat++;
        }

}

/*
 * return number of unknown environment variables
 */
return(exstat);
}
```