Introduction to C (part 2)

PHYS 795/895

W. Douglas Cramer

Basic File I/O

- Opening file: `fp=fopen(char *name, char *mode)`
  - `fp` is file pointer (`FILE *fp`), `FILE` type contains info about file
  - mode: “r” (read), “w” (write), “a” (append), ...
- Closing file: `fclose(fp)`
- Check for end of file: `int feof(fp)` (0:not end of file, !=0:end of file)
- Some commonly used functions
  - `int putc(int c, FILE *fp)` – print character to file
  - `int getc(FILE *fp)` – read character from file
  - `int fprintf(FILE *fp, char *format, arg1, arg2, ...)` – print to file
  - `int fscanf(FILE *fp, char *format, arg1, arg2, ...)` – read from file

```
/* fileio.c */
#include <stdio.h>
int main(void)
{
  FILE *fp;
  int inchar, inval;
  fp=fopen("file.txt","w");
  fputc('a',fp); fprintf(fp,"\n%d\n",23);
  fclose(fp);
  fp=fopen("file.txt","r");
  inchar=fgetc(fp); fscanf(fp,"%d",&inval);
  fclose(fp);
  printf("%c %d\n",inchar,inval);
}
```

```
$ gcc fileio.c
$ ./a.out
a 23
$ cat file.txt
a
23
```

Dynamic Memory Allocation

- Useful when you don't know size of storage needed ahead of time
- Some commonly used functions
  - `void *malloc(numbytes)` – reserve numbytes of memory (no initialization)
  - `void *calloc(num, typesize)` – reserve num*typesize bytes of memory (and initialize to zero)
  - `void *realloc(void *pointer, numbytes)` – change reserved numbytes of memory
  - `void free(void *pointer)` – free reserved memory located at pointer

```
/* memory.c */
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
  float *ptr;
  ptr=calloc(10,sizeof(float));
  ptr[0]=10;
  *(ptr+1)=20;
  printf("%f %f %f\n",*ptr,ptr[1],ptr[2]);
  free(ptr);
}
```

```
$ gcc memory.c
$ ./a.out
10.000000 20.000000 0.000000
```

Constants and Derived Types

- Constants
  - `#define MAXLINE 1000`
  - `enum direction {UP[=1], DOWN, LEFT, RIGHT}`
- Structures and Unions – combine multiple members
  - initialize elements in order
  - access elements with 'dot' syntax (or "->" if pointer to structure)
  - in union, members overlap in memory (only use one)
- Typedef: create synonym for type
  - e.g., `typedef char Title[100]; Title mytitle;`

```
/* struct.c */
#include <stdio.h>
int main(void)
{
  struct vector {float x; float y; float z;} r={3,5,6};
  typedef struct vector Vector;
  Vector *ptr=&r;
  printf("%4.1f %4.1f %4.1f\n",r.x,r.y,r.z);
  printf("%4.1f %4.1f %4.1f\n",ptr->x,ptr->y,ptr->z);
}
```

```
$ gcc struct.c
$ ./a.out
 3.0  5.0  6.0
 3.0  5.0  6.0
```

Scope Rules

- Variables are local to the block in which they are declared
- Variables declared outside of functions are global to all

```
/* scope.c */
#include <stdio.h>
int val=1;
void func(void);
int main(void)
{
  int val=2;
  if (1) {
    int val=3;
    printf("%d\n",val);
  }
  func();
  printf("%d\n",val);
}
void func(void)
{
  printf("%d\n",val);
}
```

```
$ gcc scope.c
$ ./a.out
3
1
2
```

Makefiles

- Simplify building executables
  - only have to enter ‘make’ to compile and build
  - only rebuild when updates are made
- Note tab indentation

```
# Makefile
CC=gcc
CFLAGS=-Wall

include: include.o add.o
	$(CC) $(CFLAGS) -o include include.o add.o

add.o: add.c add.h
	$(CC) $(CFLAGS) -c add.c

clean:
	$(RM) include include.o add.o
```

Common Mistakes

- =/== mixup
- Using uninitialized variable
- No return type
- Missing \0 at end of strings
- Treating arguments as pass-by-reference
- Accessing outside of array bounds
- ++/-- in unexpected order
- Misplaced or missing semicolons
- Missing & in scanf
- Expression order of operations not as expected (use parenthesis)
- Using uninitialized pointers
- Failure to free allocated memory (memory leak)

Readability/Formatting

- Lots of comments!!!
- One command per line
- Indentation (for nested blocks)
- Naming (suggestions)
  - functions: lowerUpper
  - structures: Upper
  - variables: lower_lower_lower (make useful; no single character names!)
  - enum,#define: ALLUPPER

Exercise

- Main Task
  - Write a function that returns the greater of two numbers (remember: two files are necessary)
  - Write a main routine that takes a list of values and outputs the largest number (use arguments)
- Bonus
  - Read the list of values from a file