Brian’s Concise ANSI C Reference Sheet

Basic Stuff

Comments
```
/* Multiline Comment Goes Here */
```

Program Start
```
main()  { ... }
```

Var Declarations
```
char, int, short, long, float, char*, (void)
int[10] foo;         /* array of ints, indexed 0-9 */
int myMatrix[5][10]; /* [row][col] */
const double e = 2.71828182845905;
```

Macro Consts
```
#define <NAME> <replacement-text>    (no semicolon)
```

Enum Consts
```
enum boolean { NO, YES };  (default: No = 0, Yes = 1)
enum escapes { BELL = '\a', TAB = '\t' }; (semicolon)
```

Logical Operators
```
%, !, &&, ||, ==
(%: 0 if divides exactly, remainder otherwise)
```

Concise Arithmetic
```
x++ (increment after eval), ++x (increment before eval)
x+=12; x-=1;
```

Cond. Expr.
```
maximum = (a > b) ? a : b;   (can use in printf stmts)
```

Control Stmts
```
if( <test> ) <do-this>;
if( <test> ) <do-this>; else <do-this>;
for( initializer; while this; do each time ) <do-this>;
do <do-this> while( <test> );
while( <test> ) <do-this>;
```

Switch Stmt
```
switch( <variable> )
{
   case <value1> : <actions>;
                   break;
   case <value2> : <actions>;
                   break;
   default       : <actions>;
                   break;
}

1. Breaks optional; w/o 'em, testing continues
2. Default flag is optional too
3. Braces not needed for multiple actions
4. Can test multiple cases at once like this:
     case '0': case '1' : printf("A Bit");
```

Functions

```
int fac(int x)
{
   if(x==1) return 1;
   else return x*fac(x-1);
}
```

Prototypes

Var names optional, e.g.: int power(int, int)

Conditional Preprocessing

```
#if SYSTEM == UNIX      #define HDR "unix.h"
#elif SYSTEM == MAC     #define HDR "mac.h"
#else                   #define HDR "windows.h"
#endif
#include HDR
```

1. Also, **#ifdef** and **#ifndef**

Pointers

Pointer Basics

1. **int* p** declares p as a pointer to an integer
2. **&** = address-of operator. **P = &c** makes p point to c
3. ***** = dereferencing operator: accesses the pointee
4. If **int *ip = &x**, then always read *ip as x

Simple Pointer Example

```
int x=1, y=2;
int *ip, *iq;    /* ip and iq are pointers to ints */
ip = &x;         /* ip points to x                 */
y = *ip;         /* y is now 1                     */
*ip = 0;         /* x is now 0                     */
iq = ip;         /* iq now points to x too         */
```

Value/Reference Pointer Example

```
void f(int val, int& ref) { val++; ref++; }
main() { int i=1; int j=1; f(i,j);
         printf("i = %d, j = %d", i, j); }
Output: i = 1, j = 2
```

Swap Example

```
/* By Reference: Use! */     /* By Value: Bad! */
void swap(int& x, int& y)    void swap(int* x, int* y)
{                            {
   int foo = x;                 int foo = *x;
   x = y;                       *x = *y;
   y = foo;                     *y = foo;
}                            }

void f(int i, int g)         void g(int i, int g)
{                            {
   swap(i,j);                   swap(&i, &j);
}                            }
```

NULL Pointer Args

1. If NULL or 0 is an arg in a function call, cast it to the ptr type expected by the fn being called!

Arrays of Ptrs

```
char* myArrayOfPointers[MAX];
```

1. Use for arrays of strings, of all diff sizes
2. Useful for sorting text lines: swap only pointers

Pointers and Arrays

```
int a[10];      int *p, *q;
p = &a[0];      q = a;
```

1. The two assignment stmts are synonyms
2. They both assign pointers to the first array element
3. ***(p+1)** then refers to the contents of a[1]
4. W/ arrays, can use ptr arithmetic: ++, --, >, <

Structs

Structs

```
struct point         /* Type Declaration */
{
  int x;
  int y;
};                   /* semicolon! */
struct point pt;     /* Var Declaration */
pt.x = 4;            /* How to access members */
```

Struct Init.

```
struct point origin = { 320, 200 };   (semicolon!)
```

Nested Structs

```
struct rectangle
{
   struct point upLeft;
   struct point lowRight;
};
rectangle rect;
rect.upLeft.x = 4;
```

Pointers and Structs

1. If p points to struct, can access member w/ **->**
2. E.g. **(*myPtr).age = 23**; == **myPtr->age = 23**;

```
struct point origin = { 320, 200 };
struct point *pp;
pp = &origin;
printf("Origin is (%d,%d)\n", (*pp).x, pp->y);
```

Pseudo-Constructor

```
struct point makePoint(int x, int y)
{
   struct point foo;
   foo.x = x;
   foo.y = y;
   return foo;
}
rect.upLeft = makePoint(3,4);
```

Variables

Global Vars

1. Must be _defined_ exactly once, outside everything
2. Must be _declared_ in each using fn (after it)

```
char name[10];
void foo()
{
   extern char name[];
   name[1] = 'r';
}
main()
{
   extern char name[];
   line[0] = 'B';
   foo();
}
```

Hiding Vars Local to Mult Functions

```
static int x=0;
int fn1() { ... <uses x> ... }
int fn2() { ... <uses x> ... }
```

1. x only avail _in this source file_, hidden elsewhere
2. E.g. push & pop both need a shared but hidden stack

Persistent Local-to-1-fn Storage

```
int foo() { static int x=0; ... }
```

1. x accessible only in foo, but persists after foo
2. E.g. to tally # times a fn is called: just increment

A Fn to Return a Random Int Between 1 and N

```
#include <stdlib.h>
int randInt(int n)
{
   return (int)((double)rand() /
          ((double)RAND_MAX + 1) * n);
}
```

Command Line Args

Command Line Arguments

1. Call main as: **main(int argc, char* argv[])**
2. **argc**, argument-count, is # of args
3. **argv**, argument-vector, is ptr to an array of char-strings that contain the args, 1 per string
4. **argv[0]** contains name of calling program, so...
5. **argc==1** means no command line args; 2 ==> 1, etc.
6. So, **argv[1]** is first optional arg
7. Finally, **argv[argc]** is NULL, by convention
8. If progs below called w/ "echo hello, world", argc==3, argv[0]=="echo", argv[1]=="hello,", argv[2]=="world", and argv[3]==NULL

Echo Example
With Arrays

```
main(int argc, char *argv[])
{
  int i;
  for(i=1; i < argc; i++)
    printf("%s%s", argv[i], (i < argc-1) ? " " : "");
    printf("\n");
}
```

Echo Example
With Pointers

```
main(int argc, char *argv[])
{
   while(--argc > 0)
     printf("%s%s", *++argv, (argc > 1) ? " " : "");
   printf("\n");
}
```

Reading Unknown
# of Command Line
Args as Filenames

```
main(int argc, char* argv[])
{
   FILE *fp;
   char *progName = argv[0];  /* for errors */
   if(argc == 1)  /* no command line args */
      <do-this>;
   else
      while(--argc > 0)
         if((fp = fopen(*++argv, "r")) == NULL)
         {
            fprintf(stderr, "%s: can't open %s\n",
               progName, *argv);
            exit(1);
         }
         else
         {
            <do-this>;  /* do stuff w/ current file */
            fclose(fp);
         }
      exit(0);
}
```

Printing, Files

---

printf Parameters

%d	integer
%f	float
%6d	integer, at least 6 wide
%6f	decimal, at least 6 wide
%.2f	decimal, 2 chars a/ decimal pt.
%6.2f	decimal, at least 6 wide, w/ 2 a/ decimal pt
%s	char*
%c	single char
%6s	string, minimum length 6
%.5s	print at most 5 chars f/ a string
%%	the percent sign

Runtime width

```
printf("%.*s", max, str);
/* prints at most max chars f/ str */
```

```
Escape Seqs        \n = newline     \t = tab     \' = single quote
                   \b = backspace   \a = bell    \" = double quote
                   \\ = backslash   \? = "?"

Simple File I/O    FILE *fp;  /* fp is a ptr to a file */
                   fp = fopen("myFile.txt", "r");
                   <Use fscanf and fprintf>
                   fclose(fp);

                   "r" --> open for reading
                   "w" --> open for writing (destroys previous contents)
                   "a" --> open for appending (saves previous contents)
                   "r+" -> open for both reading and writing

                   1. 1st arg of fscanf & fprintf is the file ptr
                   2. E.g. fprintf(fp, "Wow.");
                   3. A nonexistant file for "w" and "a" is created

Getting Input      int c;  /* use int so it can handle EOF */
Char by Char       while ((c = getchar()) != EOF)  <do-this>;

Formatted Input    /* e.g. to read "25 Dec 1988" */
                   int day, year; char month[20];
                   scanf("%d %s %d", &day, month, &year);

                   /* e.g. to read "11--13--71" */
                   int day, year, month;
                   scanf(%d--%d--%d", &month, &day, &year);

                   1. Must scanf into _pointers_! (month already a ptr)
                   2. Ignores spaces in its format string
                   3. Skips over whitespace in its input
                   4. Returns # succesfully matched & assigned ptrs

Read a line of     int maxLine = 80;  char line[80];  FILE *fp;
a File into a Str  fp = fopen("myFile.txt", "r");
                   line = fgets(line, maxLine, fp);

                   1. Gets next input line f/ file _fp_ into character
                        array _line_, reading at most _maxLine-1_ chars
                   2. Returns the line!
```

Standard Library Functions

```
stdlib.h
double atof(str)    Converts string to float
int    atoi(str)    Converts string to int
int    system(str)  Executes Unix system command, e.g. system("date");
int    rand()       Returns a random int between 0 and 32767

1. To get a random int between 0 and N, write a fn to return:
     (int)((double)rand() / ((double)RAND_MAX + 1) * N)
```

stdio.h

sprintf	Print to a string, 1st arg is the string to print to
sscanf	Read from a string, 1st arg is the string to read f/
remove	Remove a file, e.g. remove(fp);
ungetc	Push a char back onto input a/ being read. E.g.: char c = getchar(); if(c=='\') ungetc(c, fp);

ctype.h

islower(c)	lowercase letter?
isupper(c)	uppercase letter?
isalpha(c)	islower || isupper?
isdigit(c)	decimal digit?
isalnum(c)	isalpha || isdigit?
isspace(c)	space, newline, tab, or formfeed?
c = tolower(c);	Just returns c if doesn't operate
c = toupper(c);	Just returns c if doesn't operate

math.h

double sin	(x)	sine of x; also cos(x) and tan(x)
double sinh	(x)	hyperbolic sine of x; also cosh(x) and tanh(x)
double asin	(x)	arcsine of x; also acos(x) and atan(x)
double exp	(x)	e to the xth power
double log	(x)	ln(x)
double log10	(x)	$\log_{10}$(x)
double pow	(x)	x to the yth power
double sqrt	(x)	square root of x
double ceil	(x)	smallest int not less than x
double floor	(x)	largest int not greater than x
double fabs	(x)	absolute value of x

string.h

char* strcpy	(s, ct)	Copy string ct to string s, incl '\0', return s
char* strncpy	(s, ct, n)	As above, at most n chars, pad w/ '\0's if necc
char* strcat	(s, ct)	Concatenate ct to end of s, return s
char* strncat	(s, ct, n)	As above, at most n chars, terminates w/ '\0'
int strcmp	(cs, ct)	Return 0 if cs==ct, <0 if cs<ct, >0 if cs>ct
int strncmp	(cs, ct, n)	As above, compare at most n chars
ch ar* strchr	(cs, c)	Return pointer to 1st c in cs, NULL if none
char* strrchr	(cs, c)	As above, but _last_ c in cs
char* strstr	(cs, ct)	Return ptr to 1st ct in cs, NULL if none

1. Beware of Strcat: It'll only work as listed if s already contains enough room for t. I.e. if s was just declared as char* s="foo", it won't work; you've got to make s big enough to hold a concat first!