Intro to C: Tutorial Examples

Adapted from Kernigan & Ritchie, 2nd Ed.

1.1 Getting Started

```
#include <stdio.h>

main()
{
  printf("hello, world\n");
}
```

- *Concepts:* Functions; `main()`; header files; `printf()`; string constants; escape sequences; compiling
- Most C header files are stored in `/usr/include`
- Some other escape sequences: `\t` (tab), `\"` (double quote), `\\` (backslash)

1.2 Variables and Arithmetic Expressions

```
#include <stdio.h>

/* print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300 */
main()
{
  int fahr, celsius;
  int lower, upper, step;

  lower = 0;     /* lower limit of temperature table */
  upper = 300;   /* upper limit */
  step = 20;     /* step size */

  fahr = lower;
  while (fahr <= upper) {
    celsius = 5*(fahr - 32)/9;
    printf("%d\t%d\n", fahr, celsius);
    fahr = fahr + step;
  }
}
```

- *Concepts:* Structured programming; comments; declarations; variables; arithmetic expressions; loops; formatted output
- Some other basic data types: `char`, `float`, `double`
- Some other conversion specifiers: `%c` (character), `%s` (string), `%f` (float)

```
#include <stdio.h>

/* print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300; floating-point version */
main()
{
  float fahr, celsius;
  int lower, upper, step;

  lower = 0;     /* lower limit of temperature table */
  upper = 300;   /* upper limit */
  step = 20;     /* step size */

  fahr = lower;
  while (fahr <= upper) {
```

```
    celsius = (5.0/9.0)*(fahr - 32.0);
    printf("%3.0f %6.1f\n", fahr, celsius);
    fahr = fahr + step;
  }
}
```

- *Concepts:* Floating-point constants & expressions; type conversion

1.3 The For Statement

```
#include <stdio.h>

/* print Fahrenheit-Celsius table */
main()
{
  int fahr;

  for (fahr = 0; fahr <= 300; fahr = fahr + 20)
    printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

- *Concepts:* Complex expressions in place of simple variables
- Shorthand: `fahr = fahr + 20` can be written `fahr += 20`

1.4 Symbolic Constants

```
#include <stdio.h>

#define LOWER 0   /* lower limit of table */
#define UPPER 300 /* upper limit */
#define STEP  20  /* step size */

/* print Fahrenheit-Celsius table */
main()
{
  int fahr;

  for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
    printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

- *Concepts:* preprocessing

1.5 Character Input and Output

1.5.1 File Copying

```
#include <stdio.h>

/* copy input to output; 1st version */
main()
{
  int c;

  c = getchar();
  while (c != EOF) {
    putchar(c);
    c = getchar();
  }
}
```

- *Concepts:* Text streams; `getchar()` & `putchar()`

```
#include <stdio.h>

/* copy input to output; 2nd version */
main()
{
  int c;

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

- *Concepts:* Assignments in expressions

1.5.2 Character Counting

```
#include <stdio.h>

/* count characters in input; 1st version */
main()
{
  long nc;

  nc = 0;
  while (getchar() != EOF)
    ++nc;
  printf("%ld\n", nc);
}
```

- *Concepts:* Increment operator; `long` types
- Increment (`++`) and decrement (`--`) operators can be *prefix* or *postfix*

```
#include <stdio.h>

/* count characters in input; 2nd version */
main()
{
  double nc;

  for (nc = 0; getchar() != EOF; ++nc)
    ;
  printf("%.0f\n", nc);
}
```

- *Concepts:* Null statement

1.5.3 Line Counting

```
#include <stdio.h>

/* count lines in input */
main()
{
  int c, nl;

  nl = 0;
  while ((c = getchar()) != EOF)
    if (c == '\n')
      ++nl;
  printf("%d\n", nl);
}
```

- *Concepts:* Equality operator; character constants

1.5.4 Word Counting

```
#include <stdio.h>

#define IN  1   /* inside a word */
#define OUT 0   /* outside a word */

/* count lines, words, and characters in input */
main()
{
  int c, nl, nw, nc, state;

  state = OUT;
  nl = nw = nc = 0;
  while ((c = getchar()) != EOF) {
    ++nc;
    if (c == '\n')
      ++nl;
    if (c == ' ' || c == '\n' || c == '\t')
      state = OUT;
    else if (state == OUT) {
      state = IN;
      ++nw;
    }
  }
  printf("%d %d %d\n", nl, nw, nc);
}
```

- *Concepts:* Multiple assignment; logical operators; `if/else`
- The other binary logical operator: `&&` (AND)

1.6 Arrays

```
#include <stdio.h>

/* count digits, white space, others */
main()
{
  int c, i, nwhite, nother;
  int ndigit[10];

  nwhite = nother = 0;
  for (i = 0; i < 10; ++i)
    ndigit[i] = 0;

  while ((c = getchar()) != EOF)
    if (c >= '0' && c <= '9')
      ++ndigit[c-'0'];
    else if (c == ' ' || c == '\n' || c == '\t')
      ++nwhite;
    else
      ++nother;

  printf("digits =");
  for (i = 0; i < 10; ++i)
    printf(" %d", ndigit[i]);
  printf(", white space = %d, other = %d\n",
 nwhite, nother);
}
```

- *Concepts:* Arrays; character representation of digits; `if/else-if/else`
- Another multi-branch control statement: `switch`

1.7 Functions

```
#include <stdio.h>

int power(int m, int n);

/* test power function */
main()
{
  int i;

  for (i = 0; i < 10; ++i)
    printf("%d %d %d\n", i, power(2,i), power(-3,i));
  return 0;
}

/* power: raise base to n-th power; n >= 0 */
int power(int base, int n)
{
  int i, p;

  p = 1;
  for (i = 1; i <= n; ++i)
    p = p * base;
  return p;
}
```

- *Concepts:* Definitions, declarations, parameters, arguments & prototypes; the `return` statement
- Shorthand: `p = p * base;` can be written `p *= base;`

1.8 Arguments—Call by Value

```
#include <stdio.h>

int power(int m, int n);

/* test power function */
main()
{
  int i;

  for (i = 0; i < 10; ++i)
    printf("%d %d %d\n", i, power(2,i), power(-3,i));
  return 0;
}

/* power: raise base to n-th power; n >= 0; version 2 */
int power(int base, int n)
{
  int p;

  for (p = 1; n > 0; --n)
    p = p * base;
  return p;
}
```

- *Concepts:* Decrementing loop
- Argument values can be changed inside functions using *pointers*

1.9 Character Arrays

```
#include <stdio.h>
#define MAXLINE 1000   /* maximum input line size */

int getline(char line[], int maxline);
void copy(char to[], char from[]);

/* print longest input line */
main()
{
  int len;   /* current line length */
  int max;   /* maximum length seen so far */
  char line[MAXLINE];      /* current input line */
  char longest[MAXLINE];   /* longest line saved here */

  max = 0;
  while ((len = getline(line, MAXLINE)) > 0)
    if (len > max) {
      max = len;
      copy(longest, line);
    }
  if (max > 0)   /* there was a line */
    printf("%s", longest);
  return 0;
}

/* getline: read a line into s, return length */
int getline(char s[], int lim)
{
  int c, i;

  for (i=0; i<lim-1 && (c=getchar())!=EOF && c!='\n'; ++i)
    s[i] = c;
  if (c == '\n') {
    s[i] = c;
    ++i;
  }
  s[i] = '\0';
  return i;
}

/* copy: copy 'from' into 'to'; assume to is big enough */
void copy(char to[], char from[])
{
  int i;

  i = 0;
  while ((to[i] = from[i]) != '\0')
    ++i;
}
```

- *Concepts:* The `void` type; the null character; bounds checking

1.10 External Variables and Scope

```
#include <stdio.h>

#define MAXLINE 1000   /* maximum input line size */

int max;               /* maxumum length seen so far */
char line[MAXLINE];    /* current input line */
char longest[MAXLINE]; /* longest line saved here */

int getline(void);
void copy(void);

/* print longest input line; specialized version */
int main(void)
{
  int len;
  extern int max;
  extern char longest[];

  max = 0;
  while ((len = getline()) > 0)
    if (len > max) {
      max = len;
      copy();
    }
  if (max > 0)   /* there was a line */
    printf("%s", longest);
  return 0;
}

/* getline: specialized version */
int getline(void)
{
  int c, i;
  extern char line[];

  for (i=0; i<MAXLINE-1 && (c=getchar())!=EOF && c!='\n'; ++i)
    line[i] = c;
  if (c == '\n') {
    line[i] = c;
    ++i;
  }
  line[i] = '\0';
  return i;
}

/* copy: specialized version */
void copy(void)
{
  int i;
  extern char line[], longest[];

  i = 0;
  while ((longest[i] = line[i]) != '\0')
    ++i;
}
```

- *Concepts:* Automatic *vs* external variables

A.1 Pointers

A pointer is a variable that contains the address of a variable. For example:

```
int i,j; /* simple integer variables */
int *pi; /* pointer to an integer variable */

i = 10;  /* assign the value 10 to i */
pi = &i; /* assign the address of i to pi */
j = *pi; /* assign the contents of address pi to j */
```

The type of the pointer, e.g. `int *`, indicates to the compiler the type of the data stored at the address. Pointers allow a function to change objects in the function that called it. For example, compare:

```
void swap(int x, int y)
{
  int temp;

  temp = x;
  x = y;
  y = temp;
}
```

with

```
void swap(int *px, int *py)
{
  int temp;

  temp = *px;
  *px = *py;
  *py = temp;
}
```

The version on the left fails because C passes arguments by value. But by passing the *addresses* of the arguments, e.g. `swap(&a, &b)`, the *contents* of the addresses can be interchanged for the desired effect.

There is an elegant relationship in C between pointers and arrays: an array name is in fact a pointer. The first element of an array `x[10]` of `doubles` is `x[0]`, which is equivalent to `*x`. The second element is `x[1]`, equivalent to `*(x + 1)` (the compiler knows that `x` is a pointer to `double`, so `x + 1` is an address one `double` further along in memory). This fact can be used to allocate arrays dynamically using `malloc()`:

```
double *x = (double *) malloc(10*sizeof(double));
```

A.2 Math Functions & Macros

A.2.1 Common Math Functions (defined in `math.h`)

In the following table, `x` and `y` are `doubles` and `n` is an `int`.

Category	Function(s)	Notes
Trigonometric	`cos(x)`, `sin(x)`, `tan(x)`, `acos(x)`, `asin(x)`, `atan(x)`	Use radians
Hyperbolic	`cosh(x)`, `sinh(x)`, `tanh(x)`, `acosh(x)`, `asinh(x)`, `atanh(x)`	
Exponential & Log	`exp(x)`, `log(x)`	Use base e
Power Functions	`pow(x,y)`, `sqrt(x)`	
Bessel Functions	`j0(x)`, `j1(x)`, `jn(n,x)`, `y0(x)`, `y1(x)`, `yn(n,x)`	

A.2.2 Math Macros

C does not provide an exponeniation operator. Nor are there functions for simple operations such as taking the absolute value of a number of arbitrary type, determining the sign of a number, etc. The following are simple macros for these types of operations. They should be used with care however because they are inefficient (complex arguments may be evaluated more than once) and can have unexpected side effects.

```
#define SQ(x)    ((x)*(x))                            /* square of x */
#define ABS(x)   ((x) < 0 ? -(x) : (x))               /* absolute value of x */
#define SGN(x)   ((x) < 0 ? -1 : (x) > 0 ? 1 : 0)     /* sign of x */
#define MIN(x,y) ((x) < (y) ? (x) : (y))              /* minimum of x and y */
#define MIN(x,y) ((x) > (y) ? (x) : (y))              /* maximum of x and y */
```

Note also that there is no built-in C support for complex numbers. The best strategy in this case is to define a *structure* that consists of the real and imaginary parts of the number, then write simple functions to perform basic math on these structures.

A.3 Recommended Reading

Numerical Recipes in C, Chapter 1.2