C language

The first C program:

```
#include <stdio.h>          /* Standard I/O library */
int main () {               /* Function main with no arguments */
  printf("Hello, World!\n");  /* call to printf function */
  return 1;                 /* return SUCCESS = 1 */
}

% gcc -o hello hello.c
% hello
Hello, World!
%
```

```
C = (5/9)*(F-32);   Celsius vs Fahrenheit table (in steps of 20F)

#include <stdio.h>
int main() {
  int fahr, celsius, lower, upper, step;
  lower = 0;
  upper = 300;
  step = 20;
  fahr = lower;
  while (fahr <= upper) {
    celsius = 5 * (fahr - 32) / 9;
    printf("%d\t%d\n",fahr, celsius);
    fahr += step;
  }
  return 1;
}


Remarks: 5/9 = 0
         primitive data types: int, float, char, short, long, double
         integer arithmetic: 0F = 17C instead of 17.8C
         %d, %3d, %6d etc for formatting integers
         \n newline  \t tab
```

Next Version: (using float)

```
#include <stdio.h>
int main() {
  float fahr, celsius;
  int lower, upper, step;
  lower = 0;
  upper = 300;
  step = 20;
  fahr = lower;
  while (fahr <= upper) {
    celsius = (5.0 / 9.0) * (fahr - 32.0);
    printf("%3.0f %6.1f \n", fahr, celsius);
    fahr += step;
  }
  return 1;
}
```

Remarks: %6.2f 6 wide; 2 after decimal

Version 3: (using for loop)

```
#include <stdio.h>
int main() {
  int fahr;
  for (fahr=0; fahr <= 300; fahr += 20)
    printf("%3d %6.1f \n", fahr, (5.0 / 9.0) * (fahr - 32.0));
  return 1;
}
```

Version 4: with Symbolic Constants.

```
#include <stdio.h>
#define LOWER 0
#define UPPER 300
#define STEP 20

int main() {
  int fahr;
  for (fahr=LOWER; fahr <= UPPER; fahr += STEP)
    printf("%3d %6.1f \n", fahr, (5.0 / 9.0) * (fahr - 32.0));
  return 1;
}
```

Character I/O:

```
  c = getchar();
  putchar(c);
```

File copying:

```
  #include <stdio.h>
  int main() {
    int c;
    c = getchar();
    while (c != EOF) {
      putchar(c);
      c = getchar();
    }
  }
```

Simpler Version:

```
  #include <stdio.h>
  int main() {
    int c;
    while ((c = getchar())!= EOF)
      putchar(c);
  }
```

```
  c = getchar() != 0  is equivalent to c = (getchar() != EOF)
                      which results in c getting a value of 0 (false)
                      or 1 (true)
```

Character Counting:

```
  #include <stdio.h>
  int main () {
    long nc = 0;
    while (getchar() != EOF)
      nc++;
    printf("%ld\n",nc);
  }
```

```
  #include <stdio.h>
  int main () {
    long nc;
    for (nc=0;getchar() != EOF;nc++);
    printf("%ld\n",nc);
  }
```

Remarks: nc++, ++nc, --nc, nc--
%ld for lond integer

Line Counting:

```
#include <stdio.h>
int main () {
  int c, nl=0;
  while ((c = getchar()) != EOF)
    if (c == '\n')
      nl++;
  printf("%d\n",nl);
}
```

Word Count:

```
  #include <stdio.h>
  #define IN 1
  #define OUT 0
  int 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",nc, nw, nl);
  }
```

Remarks: short-circuit evaluation of || and &&
nw++ at the beginning of a word
use state variable to indicate inside or outside a word.

Arrays:
  (Count #occurrences of each digit, blank, tab, newline, other characters)

```
#include <stdio.h>
int 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);
}
```

Functions:

```
#include <stdio.h>
int power(int, int);          /* function prototype */
int main() {
  int i;
  for (i = 0; i < 10; i++)
    printf("%d %d %d\n", i, power(2,i), power(-3,i));
}

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

Remarks: Call by value mechanism;
Change in n's value not reflected in main
Using pointers, we can achieve Call by reference effect.

```
/* Character Arrays
   Read a set of text lines and print the longest
*/
#include <stdio.h>
#define MAXLINE 1000
int getline(char [],int);
void copy(char [], char []);
int main() {
  int len, max;
  char line[MAXLINE], longest[MAXLINE];
  max = 0;
  while ((len = getline(line,MAXLINE)) > 0)
    if (len > max) {
      max = len;
      copy(longest,line);
    }
  if (max > 0)
    printf("%s",longest);
}
```

```
int getline(char s[], int limit) {
  int c, i;
  for (i=0; i<(limit - 1) && (c=getchar())!=EOF && c != '\n'; i++)
    s[i] = c;
  if (c == '\n') {
    s[i] = c;
    i++;
  }
  s[i] = '\0';
  return i;
}

void copy(char to[], char from[]) {
  int i=0;
  while ((to[i] = from[i]) != '\0')
    i++;
}
```

External Variables: (Global variables)

- All variables declared so far are local to the functions.
- External or Global variables are accessible to all functions
- These are defined once outside of functions.
- Must be defined once more within each function which is going to to use them.
- Previous program rewritten with external variables.

```
/* Longest line program with line, longest, max as external variables */
#include <stdio.h>
#define MAXLINE 1000
int max;
char line[MAXLINE];
char longest[MAXLINE];
int getline(void);      /* no parameters */
void copy(void);        /* no parameters */
int main() {
  int len;
  extern int max;
  extern char longest[];
  max = 0;
  while ((len = getline()) > 0)
    if (len > max) {
      max = len;
      copy();
    }
  if (max > 0)
    printf("%s",longest);
}
```

```
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;
}

void copy(void) {
  int i=0;
  extern char line[], longest[];
  while ((longest[i] = line[i]) != '\0')
    i++;
}
```

Data Types:

char, int, float, double
long int (long), short int (short), long double
signed char, signed int
unsigned char, unsigned int

1234L is long integer, 1234 is integer, 12.34 is float,
  12.34L is long float

'a', '\t', '\n', '\0', etc. are character constants
strings: character arrays (see <string.h> for string functions)
  "I am a string"
  always null terminated.

'x' is different from "x"

Type Conversions:

- narrower types are converted into wider types
  ex. in f + i integer i is converted into float

- characters <---> integers

```
int atoi(char s[]) {
  int i, n=0;
  for (i=0; s[i] >= '0' && s[i] <= '9'; i++)
    n = 10 * n + (s[i] - '0');
  return n;
}
```

- <ctype.h> library contains conversion functions such as
  tolower(c) isdigit(c) etc.

- Boolean values: true : >= 1 false: 0

- Bitwise operations: applied to cgar, int, short, long
  & : and | : or ^ : ex-or << : left-shift >> : right-shift
  ~ : one's complement

ex. bit count

```
int bitcount (unsigned int x) {
  int b;
  for (b=0; x != 0; x >> 1)
    if (x & 01)     /* octal 1 = 000000001 */
      b++;
  return b;
}
```

- conditional expressions:
  expr1? expr2:expr3;
  if expr1 is true then expr2 else expr3
  ex.

```
for (i=0; i<n; i++)
  printf("%6d %c",a[i],(i%10==9||i==(n-1))?'\n':' ');
```

Control Flow:

```
blocks: { ... }
if expr stmt;
if expr stmt1 else stmt2;
switch expr {case ... default }
while expr stmt;
for (expr1;expr2;expr3) stmt;
do stmt while expr;
break; continue (only for loops);
goto label;
```

Scope rules:

- Automatic/Local Variables: These are declared at the beginning of functions; scope is the function body.
- External/Global Variables: These are declared outside functions; scope is from the point where they are declared till end of file (unless prefixed by extern)
- Static Variables: use static prefix on functions and variable declarations to limit scope;
  * static prefix on external variables will limit scope to the rest of the source file (not accessible in other files)
  * static prefix on functions will make them invisible to other files.
  * static prefix on internal variables will create permanent private storage; retained even upon function exit.
- Variables can be declared within blocks too; scope is until end of the block

Pointers and Arrays:

Pointer variable contains the address of another variable.

- unary operator & applied to variables gives the address of variable

```
char c;
p = &c; /* address of c is assigned to variable p */
```

- unary operator * applied to pointer accesses the variable pointer points to

```
int x=1, y=2, z[10];
int *p;
p = &x;
y = *p;
*p = 0;
p = &z[0];
```

Using pointers to achieve Call-By-Reference effect:

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

int a=10,b=20;
swap(&a,&b);
```

```
Pointers and Arrays:
  int a[10], *pa, x;
  pa = &a[0];
  x = *pa;
  x = *(pa+1);
  x = *(pa+5);
  pa = a; /* same as pa = &a[0];

  Main difference between arrays and pointers (even though both
    contain addresses):

    array name is not a variable; so
     pa = a; pa++ OK
     a = pa; a++; NOT OK

  When an array name is passed as a parameter to a finction,
    actually the address of the first element is passed;
    so arrays are always passed as pointers)
```

```
int strlen(char *s) {
  int n;
  for (n=0; *s!='\0';s++)
    n++;
  return n;
}
```

```
int strlen(char s[]) {
  int n;
  for (n=0; s[n]!='\0';n++);

  return n;
}
```

strings = array of characters

```
void strcpy(char *s, char *t) {
  int i=0;
  while ((s[i]= t[i]) != '\0')
    i++;
}
```

```
while ((*s = *t) != '\0') {
  s++; t++;
}
```

```
  % reverse -c fileName
  reverses all lines in fileName
  the -c option reverses each line;

#include <fcntl.h> /* For file mode definitions */
#include <stdio.h>
#include <stdlib.h>

/* Enumerator */
enum { FALSE, TRUE }; /* Standard false and true values */
enum { STDIN, STDOUT, STDERR }; /* Standard I/O channel indices */

/* #define Statements */
#define BUFFER_SIZE    4096   /* Copy buffer size */
#define NAME_SIZE      12
#define MAX_LINES      100000 /* Max lines in file */

/* Function Prototypes */
void parseCommandLine (int argc, char* argv[]);
void processOptions (char* str);
void usageError ();
void pass1 ();
void trackLines (char* buffer, int charsRead);
void pass2 ();
void processLine (int i);
void reverseLine (char* buffer, int size);
void fatalError ();
```

```
/* Globals */
char *fileName = NULL; /* Points to file name */
char tmpName [NAME_SIZE];
int charOption = FALSE; /* Set to true if -c option is used */
int standardInput = FALSE; /* Set to true if reading stdin */
int lineCount = 0; /* Total number of lines in input */
int lineStart [MAX_LINES]; /* Store offsets of each line */
int fileOffset = 0; /* Current position in input */
int fd; /* File descriptor of input */
/*****************************************************************/

int main (int argc, char* argv[]) {
  parseCommandLine (argc,argv); /* Parse command line */
  pass1 (); /* Perform first pass through input */
  pass2 (); /* Perform second pass through input */
  return (/* EXITSUCCESS */ 0); /* Done */
}

/*****************************************************************/
```

```
void parseCommandLine (int argc, char* argv[]) {
/* Parse command line arguments */
  int i;

  for (i= 1; i < argc; i++) {
    if(argv[i][0] == '-')
      processOptions (argv[i]);
    else if (fileName == NULL)
      fileName= argv[i];
    else
      usageError (); /* An error occurred */
  }
  standardInput = (fileName == NULL);
}
```

```
/*****************************************************************/
void processOptions (char* str) {
/* Parse options */
  int j;

  for (j= 1; str[j] != NULL; j++) {
    switch(str[j]) /* Switch on command line flag */ {
      case 'c':
        charOption = TRUE;
        break;
      default:
        usageError ();
        break;
    }
  }
}

/*****************************************************************/

void usageError () {
  fprintf (stderr, "Usage: reverse -c [filename]\n");
  exit (/* EXITFAILURE */ 1);
}
```

```
/*****************************************************************/

void pass1 () {
/* Perform first scan through file */
  int tmpfd, charsRead, charsWritten;
  char buffer [BUFFER_SIZE];

  if (standardInput) /* Read from standard input */ {
    fd = STDIN;
    sprintf (tmpName, ".rev.%d",getpid ()); /* Random name */
    /* Create temporary file to store copy of input */
    tmpfd = open (tmpName, O_CREAT | O_RDWR, 0600);
    if (tmpfd == -1) fatalError ();
  }
  else /* Open named file for reading */ {
    fd = open (fileName, O_RDONLY);
    if (fd == -1) fatalError ();
  }

  lineStart[0] = 0; /* Offset of first line */
```

```
  while (TRUE) /* Read all input */ {
    /* Fill buffer */
    charsRead = read (fd, buffer, BUFFER_SIZE);
    if (charsRead == 0) break; /* EOF */
    if (charsRead == -1) fatalError (); /* Error */
    trackLines (buffer, charsRead); /* Process line */
    /* Copy line to temporary file if reading from stdin */
    if (standardInput) {
      charsWritten = write (tmpfd, buffer, charsRead);
      if(charsWritten != charsRead) fatalError ();
    }
  }

  /* Store offset of trailing line, if present */
  lineStart[lineCount + 1] = fileOffset;

  /* If reading from standard input, prepare fd for pass2 */
  if (standardInput) fd = tmpfd;
}
```

```
/*****************************************************************/

void trackLines (char* buffer, int charsRead) {
/* Store offsets of each line start in buffer */
  int i;

  for (i = 0; i < charsRead; i++) {
    ++fileOffset; /* Update current file position */
    if (buffer[i] == '\n') lineStart[++lineCount] = fileOffset;
  }
}

/*****************************************************************/

void pass2 () {
/* Scan input file again, displaying lines in reverse order */
  int i;

  for (i = lineCount - 1; i >= 0; i--)
    processLine (i);

  close (fd); /* Close input file */
  if (standardInput) unlink (tmpName); /* Remove temp file */
}
```

```
/*****************************************************************/

void processLine (int i) {
/* Read a line and display it */
  int charsRead;
  char buffer [BUFFER_SIZE];

  lseek (fd, lineStart[i], SEEK_SET); /* Find the line and read it */
  charsRead = read (fd, buffer, lineStart[i+1] - lineStart[i]);
  /* Reverse line if -c option was selected */
  if (charOption) reverseLine (buffer, charsRead);
  write (1, buffer, charsRead); /* Write it to standard output */
}

/*****************************************************************/
```

```
void reverseLine (char* buffer, int size) {
/* Reverse all the characters in the buffer */
  int start = 0, end = size - 1;
  char tmp;

  if (buffer[end] == '\n') --end; /* Leave trailing newline */

  /* Swap characters in a pairwise fashion */
  while (start < end) {
    tmp = buffer[start];
    buffer[start] = buffer[end];
    buffer[end] = tmp;
    ++start; /* Increment start index */
    --end; /* Decrement end index */
  }
}

/*****************************************************************/

void fatalError () {
  perror ("reverse: "); /* Describe error */
  exit (1);
}
```