The C Programming Language

“”

Contents

C is both a low level and high level computer language. It is the original 'write once, compile anywhere' language. C is just high enough level to be portable and low enough level to be efficient. C is still the most important language for writing virtual machines and operating systems.

`http://c-faq.com/`
: this is essential reading

`http://www.cs.cf.ac.uk/Dave/C/`
: a badly formatted introduction to c

`nntp://comp.lang.c`
: to be read regularly

`http://www.iu.hio.no/~mark/CTutorial/CTutorial.html`
: a pretty basic tutorial about c, doesnt go into dynamic memory allocation or anything unpleasant like that

`http://ee.hawaii.edu/~tep/EE160/Book/`
: an online book about c

`http://www.acm.uiuc.edu/webmonkeys/book/c_guide/`
: not a bad guide to but pretty old

`http://www.java2s.com/Code/C/CatalogC.htm`
: lots of example c programs

`http://www.lysator.liu.se/c/pikestyle.html`
: rob pikes notes on c

Possibly the simplest possible c program

```
#include <stdio.h>
int main ()
{
  printf ("This is a C program\n");
  return(0);
}
```

Section 1

History

C was invented, circa 1972, by Denis Ritchie et al, at Bell labs, as part of the process of creating the Unix operating system. One of the motivations of c was to aleviate the work of porting the Unix operating system to new computer architectures.

Section 2

Api

`http://www.cppreference.com/wiki/`
: a complete looking api reference oriented to c++

Search for a character within a string

```
#include <wchar.h>
wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
```

Section 3

Internationalization

3.1 Locales

Set the locale for all categories to the environment of the computer

```
setlocale (LC_ALL, ""); ~(the default is the 'c' locale)
```

Find out the current locale

```
#include <locale.h>
#include <stdio.h>
int main() {
  printf("Current locale is: %s\n", setlocale(LC_ALL,NULL));
  return 0;
}
```

Set the locale for formatting numbers (but not money) to 'french';

```
#include <stddef.h>
#include <locale.h>
#include <stdlib.h>
#include <string.h>
int main()
{
  setlocale(LC_NUMERIC, "fr");
}
```

Set the locale for formatting time values to 'spanish'

```
setlocale(LC_TIME, "es");
```

`http://www.chemie.fu-berlin.de/chemnet/use/info/libc/libc_19.html#SEC322`

c locale categories

LC_COLLATE	Collation of strings (eg 'strcoll')
LC_CTYPE	Classification and conversion of characters (multibyte, wide)
LC_MONETARY	Formatting monetary value
LC_NUMERIC	Formatting numbers (not money)
LC_TIME	Formatting date and time values
LC_ALL	All categories
LANG	Take local from the enviroment variable

Set the local to the local computer

```
setlocale(LC_ALL, ""); ~(this must come before any wprintf statement)
```

http://triptico.com/docs/unicode.html
how to convert
http://www.linux.com/archive/feature/51836

In this day and age using the 'char' character type is simply not good enough. The world has become globalised and text with foreign language characters needs to be handled.

http://www.gnu.org/software/libc/manual/html_node/Extended-Char-Intro.html#Extended-Char-I

Loop through a (unicode) text file one character at a time

```
/* ?? maybe have to convert the text file first */
// this is ISO C90 code
wint_t c;
while ((c = wgetc (fp)) != WEOF)
{
}
```

teminology

multibyte character	A character in an encoding like utf8 (variable length)
wide character	A character in ucs-2 or ucs-4 encoding (fixed length)

wide character conversion functions

int wctob(wint_t c)	Converts a wide char to a single byte (if possible)
mbrtowc	Converts next multibyte char in string to a wide char

Converting a multibyte string (utf8 for example) to a wide character string

```
wchar_t * mbstouwcs(const char *s)
{
  size_t len = strlen (s);
  wchar_t * result = malloc((len + 1) * sizeof(wchar_t));
  wchar_t * wcp = result;
  wchar_t tmp[1];
  mbstate_t state;
  size_t nbytes;

  memset (&state, '\0', sizeof (state));
  while ((nbytes = mbrtowc (tmp, s, len, &state)) > 0)
    {
      if (nbytes >= (size_t) - 2)
        /* Invalid input string.  */
        return NULL;
      *wcp++ = towupper (tmp[0]);
      len -= nbytes;
      s += nbytes;
    }
  return result;
}
```

Section 5

Defining Constant Values

Define a constant variable

```
#define ELEMENTSIZE 10
```

Section 6

Initializing Variables

Declare an integer 'i' and an array of 100 integers 'ex'

```
int i, ex[100];
```

Initialise a wide string to be 'empty'

```
wchar_t s[200] = { L'\0' };
```

Section 7

Switch Statements

A switch statement

```
switch(betty)
{
 case 1:
   printf("betty=1\n");
    break;
 case 2:
   printf("betty=2\n");
   break;
 default:
   printf("Not sure.\n");
}
```

Section 8

For Loops

The body of a for loop does not need to have braces {} around it

```
    for ( n = 1; n < index; ++n )
       if ( var->next == NULL )
           return NULL;
       else
           var = var->next;
```

Section 9

Getting Input From The User

Get a line of text from the user, a maximum of 200 wide characters

```
#include <stdio.h>
#include <wchar.h>

int main(void) {
  wchar_t buffer[201] = {L'\0'};
  wprintf(L"Enter something:");
  fgetws(buffer, 200, stdin);
  wprintf(L"You entered: %ls", buffer);
}
```

Note that fgetws (like fgetwc, fputwc, and fputws) performs an implicit conversion between multibyte and wide characters. How this actually works, I wouldnt know.

Displaying Text

`http://www.cplusplus.com/reference/clibrary/cstdio/printf/`
complete specification for printf

Printf is the traditional way. puts
Print a float specifying the output format

```
printf("%6.3f", 2.8)        ~(prints '2.800')
```

Print a string left justified

```
printf "%-4s", "foo";       ~(prints "foo ")
```

```
int swprintf(wchar_t *wcs, size_t maxlen, const wchar_t *format, ...);
```

Use an inspection set

```
scanf("%[A-Z:]", stdin);
```

```
wscanf(L"%l[A-Z:]", stdin);
```

Print 'pi' to five decimal places

```
#include <math.h>
#include <stdio.h>
fprintf(stdout, "pi = %.5f\n", 4 * atan(1.0));
```


Strings

This section deals almost exclusively with wide character string (wchar_t) not with the old c byte array strings (char). The world has moved on. Ascii just doesnt cut it anymore.
It is not possible to use printf and wprintf in the same program That is, any stream is either wide oriented or byte oriented. 'fwide' can determine which it is.
Query if output stream is wide oriented (>0) or not

```
if (fwide(stdout, 0) == 0) ...
```

print a byte string with wprintf

```
#include <locale.h>
int main () {
  char * s = "test";
  wprintf(L"ascii string: %s\n", s);
}
```

Define, initialize, print a wide character string

```
#include <stdio.h>
#include <locale.h>
#include <wchar.h>

int main () {
  wchar_t s[100];
  setlocale(LC_ALL, "");
  wcscpy (s, L"                    hello\n");
  wprintf (L"wide character string s= %ls\n");
}
```


Multibyte And Wide Strings

A multibyte string is something like utf8. In these encodings a character may occupy 1, 2 or more bytes. This is called a variable length encoding. Wide character characters have fixed length (usually 4 bytes).

12.1 Implicit Conversion

use fgetwc, fgetws, fputwc, fputws.

12.2 Explicit Conversion

Convert some input text from multibyte to wide character

```
fgets(s2, BUFSIZ, stdin);   /* read EUC string from stdin into s2 */
```

```
mbstowcs(s1, s2, BUFSIZ);   /* convert EUC string in s2 to process
```

code string in s1 */

12.3 Screen Width

```
int wcswidth (const wchar_t *s, size_t n);
```

12.4 Initialising String Variables

Define and initialise a wide character string

```
wchar_t text[100] = L"This is unicode, rain    ";
```

Initialise a wide string to be 'empty'

```
wchar_t text[50] = L"";
```

```
wchar_t text[50] = {L'\0'};    ~(the same)
```

Define and initialise as empty a string with a capacity of 19

```
wchar_t s[20]; *s = L'\0';
```

```
wchar_t s[20]; *s = 0;         ~(the same ?)
```

12.5 Comparing Strings

Test if the string variable is "snow"

```
if (wcscmp(word, L"snow") == 0)
```

12.6 Single Byte Strings

Traditionally strings in c are single byte characters. In the modern world this is no long adequate. Use 'wchar_t' instead.
http://www.chemie.fu-berlin.de/chemnet/use/info/libc/libc_5.html#SEC61
declare, initialize and print a string

```
#include <string.h>
#include <stdio.h>
int main (void)
{
  wchar_t text[80];
  wcscpy (text, L"hello");
  wprintf(L"%ls", text);
  return(0);
}
```

search a string for a particular character

```
#include <stdio.h>
#include <string.h>
int main (void)
{
  char *p; char c = 'x'
  p = strchr("some text to be searched", c);
  if (p == NULL)
    { printf("character not found"); }
  printf(p);
  return(0);
}
```

Duplicate a string

```
strdup
```

Add a character to a string

```
sprintf(s, "%s%c", s, c);
```

Section 13

Library Functions

View c library functions (on a unix-like operating system)

```
man 3 func
```

Section 14

Arrays

Declare and initialize an integer array.,

```
#include <stdio.h>
int main ()
{
  int items[] = {32, 43, 29, 0};
  printf ("The first item is %d\n", items[0]);
  printf ("The second item is %d\n", items[1]);
  return(0);
}
```

Make an array of float types

```
#include <stdio.h>
#define SIZE 3
int main (void)
{
  float x[SIZE];
  float *fp; int i;
  for (i = 0; i < SIZE; i++)
    { x[i] = 0.5 * (float) i; }
  for (i = 0; i < SIZE; i++)
    { printf("  %d  %f \n", i, x[i]); }
  fp = x;
  for (i = 0; i < SIZE; i++)
    printf("  %d  %f \n", i, *(fp + i));
}
```

Section 15

Reading And Writing Files

legal modes for opening files

`r`	Open text file for reading
`w`	Create a text file for writing
`a`	Append to text file
`rb`	Open binary file for reading
`wb`	Create binary file for writing
`ab`	Append to a binary file
`r+`	Open text file for read/write
`w+`	Create text file for read/write
`a+`	Open text file for read/write
`rb+ or r+b`	Open binary file for read/write
`wb+ or w+b`	Create binary file for read/write
`ab+ or a+b`	Open binary file for read/write

Read a text file character by character.,

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
  FILE *fp;
  char ch;
  if ((fp = fopen (argv[1], "r")) == NULL)
  {
    printf ("Cannot open file.\n");
    exit (1);
  }
  while ((ch = fgetc (fp)) != EOF)
    { printf ("%c", ch); }
  fclose (fp);
  return 0;
}
```

Open the text 'somefile' file for writing.,

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
  FILE *fp;
  if ((fp = fopen ("somefile.txt", "w")) == NULL)
  {
    printf ("Cannot open file.\n");
    exit (1);
  }
  fclose (fp);
}
```

Section 16

Functions

Define a function before the main method

```
#include <stdio.h>
int sum (int i1, int i2, int i3) {
  return (i1 + i2 + i3);
}

int main() {
  printf ("Sum is %d\n", sum (1, 2, 3));
  return (0);
}
```

Define a function with a prototype.,

```
//prototypes allow functions to be defined after they are used
#include <stdio.h>
int f (void);
int main (void) {
  int number;
  number = f(); printf ("%d", number);
```

```
  return 0;
}
int f (void)
 { return 10; }
```

Section 17

Structures

Structures allow data within a program to be organized in a logical manner. Structures are essentially the precursor of 'objects' in modern object oriented languages. C was written to implement Unix in a more portable fashion. Possibly the most important features of the C language are its inclusion of structures and pointers which make it an effective language. Structures and dynamic memory allocation can be used to create standard data structures such as linked lists, binary trees and dynamic stacks.

Defining and using a normal structure

```
#include <stdio.h>
struct point {
  int x;
  int y;
};

int main() {
  struct point p;
  p.x = 3; p.y = 4;
  printf("The point is (%d, %d)\n", p.x, p.y);
  return(0);
}
```

Define a structure and initialise an array of them

```
struct song
{
  wchar_t title[100];
  wchar_t singer[100];
} songlist[] = {
  {L"one too many mornings", L"dylan"},
  {L"money for nothing", L"knoffler"}
  {L"somewhere", L"anon"}
};
,,,,

* define and use a typedef structure.,
\begin{lstlisting}
  typedef struct point {
    int x;
    int y;
  };

  point p, q;
  int main() {
    p.x = 3; p.y = 4;
    printf ("The point is (%d, %d)\n", p.x, p.y);
    return (0);
  }
```

Define a structure type and variables in one go

```
struct student {
  char name[30];
  float number;
} rob, sarah;
```

Allocate memory for a structure

```
struct person * p;
p = (person *) malloc(sizeof(struct person);
```

Section 18

Data Structures

18.1 Linked Lists

A linked list implementation..

```
struct person
{
  wchar_t name[20];
  int age;
  struct person *next;
};

int main()
{
  struct person * someone;
  struct person * someoneelse;
}
```

The above linked list doesnt work very well because it is difficult to dynamically create list items.
A better linked list implementation..

```
struct person
{
  wchar_t * name;
  int age;
  struct person *next;
};

struct person * person();

int main()
{
  struct person * someone;
  struct person * someoneelse;
}
```

Section 19

Object Oriented Programming In C

??

Section 20

Organising C Code

`http://tldp.org/HOWTO/Program-Library-HOWTO/more-examples.html`

c files can be organised in modules, with header file.,

```
/* file: test.c */
#include <stdio.h>
#include "header.h"
char * AnotherString = "Hello Everyone";
```

```
main ()
{
  printf("running test.c...\n");
  ffTest(TEXT);
  printf("Finished.\n");
}

/* file: ffTest.c */
#include <stdio.h>
extern char *AnotherString;
void ffTest(char * s)
{
  printf("%s\n", s);
  printf("Global Variable = %s\n", AnotherString);
}

/* file header.h: */

#define TEXT "Hello World"
void ffTest();
```

20.1 Header Files

use ifndef to not include header files twice

```
/* File foo.h  */
#ifndef FOO_H_SEEN
#define FILE_FOO_SEEN
  ... head file contents
#endif
```

Declarations can be without parameter names

```
int isLeapYear(int);
```

Section 21

One Liners

Execute a shell or system command

```
system(...);
```

Section 22

The Gnu C Compiler

header file search folders

```
                /usr/local/include
libdir/gcc/target/version/include
             /usr/target/include
                    /usr/include
```

Add to the header include folders

```
gcc -Idir
```

Add folders for quoted included header file eg: #include "header.h"

```
gcc -iquote folder
```

Create object files functionA.o and functionB.o

```
gcc -c functionA.c functionB.c
```

Create an executable file from object file linking the math library

```
gcc function.o test.o -o test -lm ~(links lib/libm.a)
```

Create an executable called 'test' from test.c and the function 'func.c'

```
gcc func.c test.c -o test
```

22.1 Using Make To Compile Programs

The principle behind make it to only compile code if it or one of its dependencies has been modified. This was supposed to save time with large projects.

A very simple makefile called 'Makefile'

```
test: test.o ffunction.o
   gcc ffunction.o test.o -o test
test.o: test.c
   gcc -c test.c
ffunction.o: ffunction.c
   gcc -c ffunction.c
clean:
  rm *.o
```

(the command lines have to begin with a tab)

Run only the rule 'clean' from a makefile

```
make clean
```

Compile a program with a make file

```
make
```

22.2 Making Static Libraries

"ar": <ar>chive c code files

- ⋆ it is a convention that static library files are named with a '.a' extention
- ⋆ libraries make it easier to distribute code since only one file has to be installed

Show the contents (files in) of the library 'somelibrary.a'

```
ar t somelibrary.a
```

```
ar tv somelibrary.a ~(the same but more verbose output)
```

Create or update a library file 'somelibrary.a' with 3 files

```
ar r somelibrary.a functionA.o functionB.o functionC.o
```

Add a file to a library file

```
ar q library.a functionA.o
```

Extract files from a library archive

```
ar x somelibrary.a
```

Use a library to create an executable file called 'test'

```
gcc test.c somelibrary.a -o test
```

A makefile to build a static library

```
library.a: functionA.o functionB.o
  ar rv libsome.a functionA.o functionB.o
  rm functionA.o functionB.o
```

Add a table of contents to an archive file

```
ar s libsome.a
```

(this can be run with 'make'. or 'make clean' to remove .o files)

Use a make file which is called 'create' not 'Makefile'

```
make -f create
```

22.3 Shared Libraries

Shared libraries have names like 'libc.so.6' and are loaded dynamically when a program is run, instead of being compiled into the executable binary of a program, as is the case of static libraries made with the 'ar' program.

Check what shared libraries the program 'wc' needs to run

```
ldd /usr/bin/wc
```

22.4 Gcc Error Messages Decifered

An undefined type or variable in a header file function declaration

```
error: expected ) before * token
```

error message: file.c:6: error: conflicting types for aFunction header.h:49: error: previous declaration of aFunction was here explanation:

Section 23

Books

`C` in a nutshell (O'Reilly) This has some useful information about the wide character string functions such a wcscmp etc