C By Example

The assumption is that you know Java and need to extend that knowledge so you can program in C.

1. Hello world
2. declarations
3. pass by value/reference
4. arrays & structures
5. pointers
6. library functions
7. Compiling and Debugging – gcc and gdb
8. Shell Commands

1. “Hello World” – The Simplest Program

```
#include      <stdio.h>

int    main( int argc, char *argv[] )
{
    printf( "Hello, world\n" );
}
```

This is an include file. It contains definitions of constants, structures, and functions

This is the form used to declare and pass parameters into a program or subroutine.
This says argc is an int and argv is a char string.
If I run this program by saying “prog1 - C is fun”
Then argc is 5 and argv[0] is “-”, argv[1] is “C”, etc.

Every C program must have a main. It’s where the program starts execution.

Printf is a library subroutine that you call to output data to a screen or file.

Declarations

```
#include        <stdio.h>

int     main( int argc, char *argv[] )
{
    int        var_A;
    int       *ptr_var_A;
    double    pi = 3.14159;
    char      string[32];

    var_A = 17;
    if ( argc > 1 )
        var_A = atoi( argv[0] );

    ptr_var_A = &var_A;
    strcpy( string, "Hello, world");

    printf( "String->%s   var_A->%d    ptr_var_A-> %X   pi-> %f\n",
                string, var_A, ptr_var_A, pi );
}
```

Var_A is declared to be an integer. ptr_var_A is declared to be a pointer (or address) of an integer.

If there is an input argument, then use it for the value of var_A.

The "&" says to take the address of the variable. Strcpy is a library routine to copy strings.

Pass by Value and Pass by Reference

```
#include        <stdio.h>

void    subroutineA( int , int * );
short   functionB( int , int  );

int     main( int argc, char *argv[] )
{
    int         var_A;
    int          var_B;

    var_A = 17;
    var_B = 33;
    subroutineA( var_A, &var_B );
    printf( "Return from SubA:  %d  %d\n", var_A, var_B );
    printf( "Return from FunB:  %d\n", functionB( var_A, var_B ) );
}                                   // End of main
void    subroutineA( int A, int *B )
{
    *B = A;   // B is a pointer to an int
}                                   // End of subroutineA
short   functionB( int A, int B )
{
    return( A + B );
}                                   // End of functionB
```

These are called "prototypes".

Var_A is passed by value. Note the & on var_B which means that its address is passed in.

Note how int and ptr to int are declared.

Arrays and Structures (1)

```
#define          ARRAY_LEN       32

#include         <stdio.h>

typedef struct
{
    int          arr1[ARRAY_LEN];
    char         string[ARRAY_LEN];
} MY_STRUCT;

void    subroutineA( MY_STRUCT * );
```

This is called a “define”. Used to parameterize constants.

Here we define the structure MY_STRUCT. It’s laid out in memory as 32 ints (4 bytes each) plus 32 bytes arranged as a string.

We’ll be passing the address of this structure to the subroutine.

Arrays and Structures (2)

```
int     main( int argc, char *argv[] )
{
    int         index;
    MY_STRUCT   my_struct;
    for ( index = 0; index < ARRAY_LEN; index++ )
    {
        my_struct.arr1[index]   = index;
        my_struct.string[index] = (char)index;
    }
    subroutineA( &my_struct );
    printf( "Return of SubA:  %d  %c\n",
        my_struct.arr1[0], my_struct.string[0] );
}                                   // End of main
void    subroutineA( MY_STRUCT *xx )
{
    xx->arr1[0]   = 17;
    xx->string[0] = (char)33;
}                                   // End of subroutineA
```

Declare an instance of the structure.

Here we're fillling in the structure

Pass the structure by reference.

Note the use of "->" to reference the structure.

What is printed out by this program?

Pointers

```
#include        <stdio.h>

int     main( int argc, char *argv[] )
{
    int         a, b;
    int         *p;

    a = b = 7;
    p = &a;                         // p points to a
    printf( "p = %d\n", *p );       // 7 is printed
    *p = 3;
    printf( "a = %d\n", a );        // 3 is printed
    p = &b;
    *p = 2 * *p - a;
    printf( "b = %d\n", b );        // 11 is printed
    p = &a;
    printf( "Input an integer  ");
    scanf( "%d", p );
}                                    // End of Main
```

a and b are ints. p is a pointer to an int.

Library Functions (1)

These pages show include files and the library function prototypes that are contained in them.

```
#include        <ctype.h>
int  isalnum(int c);   // returns TRUE if char is alpha or numeric
int  isspace(int c);   // returns TRUE if char is white space
int  tolower(int c);    // returns the conversion of c to lower case

#include        <math.h>
double  cos(double x);
double  exp(double x);

#include        <stddef.h>
#typedef  unsigned    size_t;
#define   NULL  ((void *) 0)
#define   offsetof(s_type, m) \
             ((size_t) &(((s_type *) 0) ->m ))

#include        <stdio.h>
#define EOF     (-1)            // End of File
#define NULL    0
```

Library Functions (2)

```
#include       <stdlib.h>
int   atoi( const char *s );
int   rand(void);
void  *malloc( size_t size ); // Allocates "size" bytes of memory

#include       <string.h>
void  *memcpy(void *to, void *from, size_t n);
char  *strcat( char *s1, char *s2);   // Concatenates
size_t strlen( char *s);
```

Compilation And Debugging

```
Babbage gcc -g prog4.c -o prog4
babbage% gdb prog4
(gdb) l
(gdb) b 24
(gdb) r
(gdb) p index
(gdb) s
(gdb) c
(gdb) p my_struct
$4 = {arr1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
    10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    20, 21, 22, 23, 24, 25,
    26, 27, 28, 29, 30, 31},
  string = "\000\001\002\003\004\005\006\a\b\t\n
            \013\f\r\016\017\020\021\022\023\024
            \025\026\027\030\031\032\e\034
            \035\036\037"}
(gdb) q
babbage %
```

-g says prepare for debugging. –o is the name of the executable.

l says list lines of the program

b says set a breakpoint (at line 24 in this case)

r means run the program.

p says to print a variable.

s == single step
c == continue to next breakpoint.

q == quit

Shell Commands

```
babbage% vi make_all
babbage% chmod 700 make_all
babbage% ls -l make_all
-rwx------    1 jbreeche users
babbage% more make_all
gcc -g prog1.c -o prog1
gcc -g prog2.c -o prog2
gcc -g prog3.c -o prog3
gcc -g prog4.c -o prog4
gcc -g prog5.c -o prog5
gcc -g prog6.c -o prog6

babbage% mkdir foo
babbage% cd foo
babbage% echo "This is a line" > bar
babbage% cat bar
This is a line
babbage% rm bar
```

You need an editor – either emacs or vi.

Change privileges

ls –l == list directory contents

more == print out file, 1 screen at

mkdir == create a directory

cd == change directory
echo == print a line. In this case “>” puts that line in a file.
rm == delete a file (rmdir == delete a directory.)

“man” and “apropos” are my favorites.