C Fundamentals and Console I/O

CSC230: C and Software Tools

N.C. State Department of Computer Science

Computer Science NC STATE UNIVERSITY

Outline

- C Coding Style
- **`indent`**
- Executing Java and C Programs
- Platform Independence?
- Just-in-Time Compilation
- C Compilation Steps
- **`gcc`**
- C99 and C89
- Console I/O
- Streams
- Character I/O
- **`printf`**

Computer Science NC STATE UNIVERSITY

C Coding Style (Conventions)

- Universal agreement
  1. clarity and consistency important
  2. indentation, white space, and comments helpful
  3. consistent naming conventions helpful
- See the Style Guidelines for CSC230

Tools (intelligent editors, **`indent`**, etc.) will take care of much formatting for you

Computer Science NC STATE UNIVERSITY

Does it Matter?

- Entries from the International Obfuscated C Code (IOCC) Contest…

Computer Science NC STATE UNIVERSITY

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

                   #define w "Hk~HdA=Jk|Jk~LSyL[{M[wMcxNksNss:"
                  #define r"Ht@H|@=HdJHtJHdYHtY:HtFHtF=JDBIl"\
                 "DJTEJDFIlMIlM:HdMHdM=I|KIlMJTOJDOIlWITY:8Y"
                #define S"IT@I\\@=HdHHtGH|KILJJDIJDH:H|KID"\
               "K=HdQHtPH|TIDRJDRJDQ:JC?JK?=JDRJLRI|UItU:8T"
              #define _(i,j)L[i=2*T[j,O[i=O[j-R[j,T[i=2*\
             R[j-5*T[j+4*O[j-L[j,R[i=3*T[j-R[j-3*O[j+L[j,
            #define t"IS?I\\@=HdGHtGIDJILIJDIItHJTFJDF:8J"

    #define y                yy(4),yy(5),               yy(6),yy(7)
   #define yy(             i)R[i]=T[i],T[i ]            =O[i],O[i]=L [i]
#define Y _(0          ], 4] )_ (1 ], 5] )_ (2      ], 6] )_ (3 ], 7] )_=1
#define v(i)(      (( R[ i ] * _ + T [ i ]) * _ + O [ i ]) * _ + L [ i ]) *2
double b = 32  ,l ,k ,o ,B ,_ ; int Q , s , V , R [8 ], T[ 8] ,O [8 ], L[ 8] ;
#define q( Q,R ) R= *X ++ % 64 *8 ,R |= *X /8 &7 ,Q=*X++%8,Q=Q*64+*X++%64-256,
# define  p       "G\\QG\\P=GLPGTPGdMGdNGtOGlOG"    "dSGdRGDPGLPG\\LG\\LHtGHtH:"
#  define W         "Hs?H{?=HdGH|FI\\II\\GJlHJ"     "lFL\\DLTCMlAM\\@Ns}Nk|:8G"
# define   U           "EDGEDH=EtCElDH{~H|AJk}"       "Jk?LSzL[|M[wMcxNksNst:"
#  define u                 "Hs?H|@=HdFHtEI"             "\\HI\\FJLHJTD:8H"
char   *  x                  ,*X , ( * i )[              640],z[3]="4_",
*Z = "4,8O4.8O4G" r U "4M"u S"4R"u t"4S8CHdDH|E=HtAIDAIt@IlAJTCJDCIlKI\\K:8K"U
 "4TDdWDdW=D\\UD\\VF\\FFdHGtCGtEIDBIDDIlBIdDJT@JLC:8D"t"4UGDNG\\L=GDJGLKHL\
FHLGHtEHtE:"p"4ZFDTFLT=G|EGlHITBH|DIlDIdE:HtMH|M=JDBJLDKLAKDALDFKtFKdMK\
\\LJTOJ\\NJTMJTM:8M4aGtFGlG=G|HG|H:G\\IG\\J=G|IG|I:GdKGlL=G|JG|J:4b"W
S"4d"W t t"4g"r w"4iGlIGlK=G|JG|J:4kHl@Ht@=HdDHtCHdPH|P:HdDHdD=It\
BIlDJTEJDFIdNI\\N:8N"w"4lID@IL@=HlIH|FHlPH|NHt^H|^:H|MH|N=J\\D\
J\\GK\\OKTOKDXJtXItZI|YIlWI|V:8^4mHLGH\\G=HLVH\\V:4n" u t t
"4p"W"IT@I\\@=HdHHtGIDKILIJLGJLG:JK?JK?=JDGJLGI|MJDL:8M4\
rHt@H|@=HtDH|BJdLJTH:ITEI\\E=ILPILNNtCNlB:8N4t"W t"4u"
p"4zI[?Il@=HlHH|HIDLILIJDII|HKDAJ|A:JtCJtC=JdLJtJL\
THLdFNk|Nc|\
:8K"; main (
int C,char**        A) {for(x=A[1],i=calloc(strlen(x)+2,163840);
C-1;C<3?Q=_=       0,(z[1]=*x++)?((*x++==104?z[1]^=32:--x), X =
strstr(Z,z))      &&(X+=C++):(printf("P2 %d 320 4 ",V=b/2+32),
V*=2,s=Q=0,C      =4):C<4?Q-->0?i[(int)((l+=o)+b)][(int)(k+=B)
]=1:_?_-=.5/     256,o=(v(2)-(l=v(0)))/(Q=16),B=(v(3)-(k=v(1)
))/Q:*X>60?y    ,q(L[4],L[5])q(L[6],L[7])*X-61||(++X,y,y,y),
Y:*X>57?++X,  y,Y:*X >54?++X,b+=*X++%64*4:--C:printf("%d "
,i[Q][s]+i[Q ][s+1]+i[Q+1][s]+i[Q+1][s+1])&&(Q+=2)<V||(Q=
0,s+=2)<640
||(C=1));}
```

Purpose of program?


```
          /*                                  ,*/
         #include                           <time.h>
          #include/*                    _  ,o*/  <stdlib.h>
          #define  c(C)/*     -      . */return      ( C); /*    2004*/
           #include   <stdio.h>/*.   Moekan            "'   `\b-'      */
             typedef/*  */char   p;p* u                    ,w       [9
              ][128] ,*v;typedef  int _;_   R,i,N,I,A                ,m,o,e
           [9],  a[256],k    [9], n[              256];FILE*f       ;_ x   (_ K,_ r
     ,_ q){;   for(;                                    r<     q    ; K       =((
   0xffffff)   &(K>>8))^                                  n[255     &        ( K
  ^u[0        +                                  r  ++      ]     ))];c          (K
 )}       _ E                          (p*r,    p*q  ){     c(          f          =
        fopen                        (r  ,q))}_   B(_ q){c(     fseek       (f,      0
      ,q))}_ D(){c(  fclose(f ))}_  C(  p    *q){c(  0-    puts(q    )   )}_/*   /
    */main(_ t,p**z){if(t<4)c(   C("<in"       "file>"    "\40<l"   "a"  "yout> "
   /*b9213272*/"<outfile>"    ) )u=0;i=I=(E(z[1],"rb")) ?B(2)?0 :   (((o   =ftell
  (f))>=8)?(u     =(p*)malloc(o))?B(0)?0:!fread(u,o,1,f):0:0)?0:  D():0      ;if(
 !u)c(C("      bad\40input  "));if(E(z[2],"rb" )){for(N=-1;256> i;n[i++] =-1   )a[
 i]=0;       for(i=I=0;   i<o&&(R  =fgetc(   f))>-1;i++)++a[R] ?(R==N)?( ++X>7)?(n[
N]+1       )?0:(n [N   ]=i-7):0:   (N=R)    |(I=1):0;A =-1;N=o+1;for(i=33;i<127;i++
 )(        n[i   ]+  1&&N>a[i])?    N= a     [A=i]     :0;B(i=I=0);if(A+1)for(N=n[A];
I<       8&&   (R  =fgetc(f ))>    -1&& i   <o        ;i++)(i<N||i>N+7)?(R==A)?((*w[I
]           =u [i])?1:(*w[I]=   46))?(a             [I++]=i):0:0:0;D();}if(I<1)c(C(
           "  bad\40la" "yout  "))for(i           =0;256>(R=  i);n[i++]=R)for(A=8;
          A  >0;A --)   R  = ( (R&1)==0)          ?(unsigned int)R>>(01):((unsigned
         /*kero  Q'       ,KSS  */)R>>         1)^        0xedb88320;m=a[I-1];a[I
         ]=(m           <N)?(m=   N+8):        ++        m;for(i=00;i<I;e[i++]=0){
        v=w          [i)+1;for(R                    =33;127  >R;R++)if(R-47&&R-92
        &&       R-(_)* w[i])*(                   v++)=    (p)R;*v=0;}for(sprintf
              /*'_  G*/  (*w+1,                "%0"     "8x",x(R=time(i=0),m,o)^~
          0)   ;i<       8;++           i)u      [N+ i]=*(*w+i+1);for(*k=x(~
            0,i=0     ,*a);i>-       1;      ){for (A=i;A<I;A++){u[+a [ A]
          ]=w[A       ][e[A]]  ;     k      [A+1]=x (k[A],a[A],a[A+1]
        );}if     (R==k[I])      c(      (E(z[3  ],"wb+"))?fwrite(
       /* */   u,o,1,f)?D          ()|C("  \n     OK."):0    :C(
      "  \n  WriteError"            ))  for  (i  =+I-
     1  ;i >-1?!w[i][++              e[+ i]]:0;
       ) for( A=+i--;                 A<I;e[A++]
       =0); (i <I-4                  )?putchar
       ((_   )  4G)                   | fflush
      /*'      ,*/                   ( stdout
      ):       0&                     0;}c(C
     ("       \n                     fail")
     )      /*                      dP' /
           dP                      pd  '
          '                        zc
                                   */
                                  }
```

```
[I++]=i):0:0:0;D(
);}if(I<1)c(C("
bad\40la”"yout"))
for(i=0;256>(R=i)
;n[i++]=R)for(A=8
;A>0;A--
)R=((R&1)==0)?(un
signed
int)R>>(01):((uns
igned/*kero
Q',KSS
*/)R>>1)^0xedb883
20;m=a[I-
1];a[I]=(m
<N)?(m=N+8):++m;f
or(i=00;i<I;e[i++
]=0){
```

Ex.: Some GNOME Project Guidelines

- "Programmers should strive to write good code so that it is easy to understand and modify by others
- Important qualities of good code
  - clarity
  - consistency
  - extensibility
  - correctness"

Example… (cont'd)

- "It is important to follow a good naming convention for the symbols in your programs
  - Function names should be of the form **`module_submodule_operation`**, for example, **`gnome_canvas_set_scroll_region`**
  - Symbols should have descriptive names: do not use **`cntusr()`**, use **`count_active_users()`** instead
  - Function names are lowercase, with underscores to separate words, like this:

    `gnome_canvas_set_scroll_region()`"

Example… (cont'd)

- "Macros and enumerations are uppercase, with underscores to separate words, like this: **`GNOMEUIINFO_SUBTREE()`** for a macro
- Typedefs and structure names are mixed upper and lowercase, like this: **`GnomeCanvasItem`**, **`GnomeIconList`**
- Very short and terse names should only be used for the local variables of functions; never call a global variable **`x`**; use a longer name that tells what it does"

Computer Science NC STATE UNIVERSITY

Another Ex.: Some Linux Guidelines

- "Tabs are 8 characters, and indentations too
- Put the opening brace last on the line, and put the closing brace first, thusly:

```
if (x is true) {
  we do y
}
```

- Functions have the opening brace at the beginning of the next line, thus:

```
int function(int x)
{
  body of function
}"
```


Computer Science NC STATE UNIVERSITY

Our Guidelines! (These Matter!)

- File level comments
  - Author(s) name and unity id(s)
  - Brief purpose of program or module within program
- Function comments
  - Function's purpose
  - Inputs (global or parameters)
  - Outputs (return values and side effects)
  - Pre-conditions
  - Post-conditions (including side effects)

Our Guidelines! (These Matter!)

- Global Variables
  - Describe purpose
- Magic Numbers
  - Use #define except for obvious numbers (-1, 0, 1, 2)
    - Unless those numbers have a specific named purpose or are an exit code!!!

Our Guidelines! (These Matter!)

- Indentation
  - All indentation must be spaces (except for Makefiles)
  - The number of spaces for indentation must be consistent
    - 2 to 3 spaces
  - Indent:
    - Statements in a function
    - Statements in a control structure
    - Statements in a block { }

Our Guidelines! (These Matter!)

- Curly Braces
  - Functions – opening curly brace on next line
  - Everything else – opening curly brace at end of control structure
- Statements
  - 1 statement per line

Formatting with **indent**

- Many editors and IDEs (emacs, vim, Eclipse, Visual Studio, …) automatically do formatting while you write your code
- Another option: use a standalone tool for formatting, e.g., **indent**
- Warning: remove tabs from your source code before using **indent**

Computer Science NC STATE UNIVERSITY

Example: Code Before **indent**

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

static int  computelength (int, int);
int main (void) {
    typedef struct { int         left; int         right; int         length; }
          linesegment;
    linesegment *seg1, *seg2; seg1 = (linesegment *) malloc (sizeof (linesegment
)); seg2 = (linesegment *) malloc (sizeof (linesegment));
                (void) printf ("Enter left edge of segment 1: ");
(void) scanf ("%d", &(seg1->left)); (void) printf ("Enter right edge of segment
1: "); (void) scanf ("%d", &(seg1->right)); (void) printf ("Enter left edge of s
egment 2: "); (void) scanf ("%d", &(seg2->left)); (void) printf ("Enter right ed
ge of segment 2: ");
                           (void) scanf ("%d", &(seg2->right));
                           seg1->length = computelength (seg1->left, seg1->righ
t);
seg2->length = computelength (seg2->left, seg2->right); if (seg1->length == seg2
->length) printf ("segment lengths are equal\n"); else printf ("segment lengths
are NOT equal\n"); return 0; } int computelength (int left, int right) { return
(right-left); }
```

- A mess!

Computer Science NC STATE UNIVERSITY

Example: Using **indent**

```
indent prog.c
```

- Lots of options, customize to your preference
  - put these options in a file named **.indent.pro**, in your home directory
- Default indent does NOT meet all of our style guidelines!

Computer Science NC STATE UNIVERSITY

```
static int  computelength (int, int);
int
main (void)
{
    typedef struct {
    int         left;
    int         right;
    int         length;
    }           linesegment;
    linesegment *seg1,
                *seg2;

    seg1 = (linesegment *) malloc (sizeof (linesegment));
    seg2 = (linesegment *) malloc (sizeof (linesegment));
    (void) printf ("Enter left edge of segment 1: ");
    (void) scanf ("%d", &(seg1->left));
    (void) printf ("Enter right edge of segment 1: ");
    (void) scanf ("%d", &(seg1->right));
    (void) printf ("Enter left edge of segment 2: ");
    (void) scanf ("%d", &(seg2->left));
    (void) printf ("Enter right edge of segment 2: ");
    (void) scanf ("%d", &(seg2->right));
    seg1->length = computelength (seg1->left, seg1->right);
    seg2->length = computelength (seg2->left, seg2->right);
    if (seg1->length == seg2->length)
    printf ("segment lengths are equal\n");
    else
    printf ("segment lengths are NOT equal\n");
    return 0;
}

int
computelength (int left, int right) {
    return (right - left);
}
```

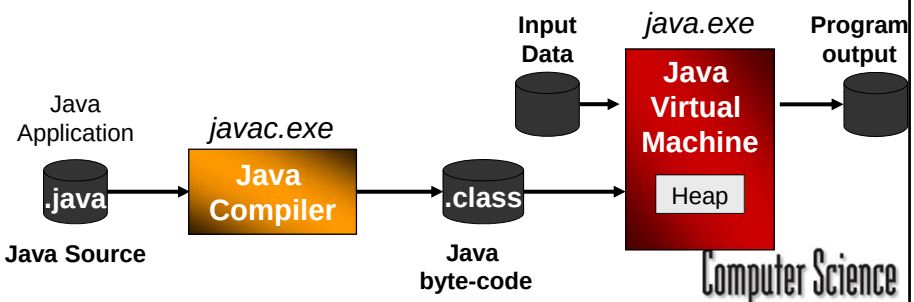
Example: after **indent**

- Much better!

Computer Science NC STATE UNIVERSITY

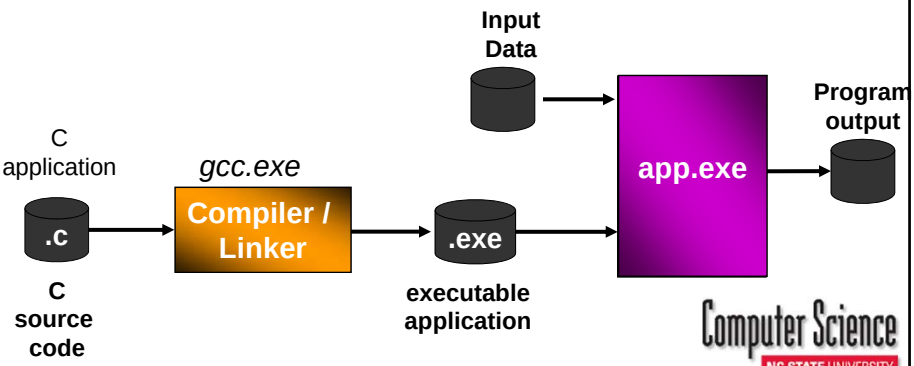
Executing Java Programs

1. Java source code is compiled into platform-independent intermediate form (*bytecode*)
2. This intermediate code is interpreted by the Java Virtual Machine (JVM)



Executing C Programs

1. HLL source code is compiled into the instruction set of the target computer
2. This code is loaded and executed directly by the host



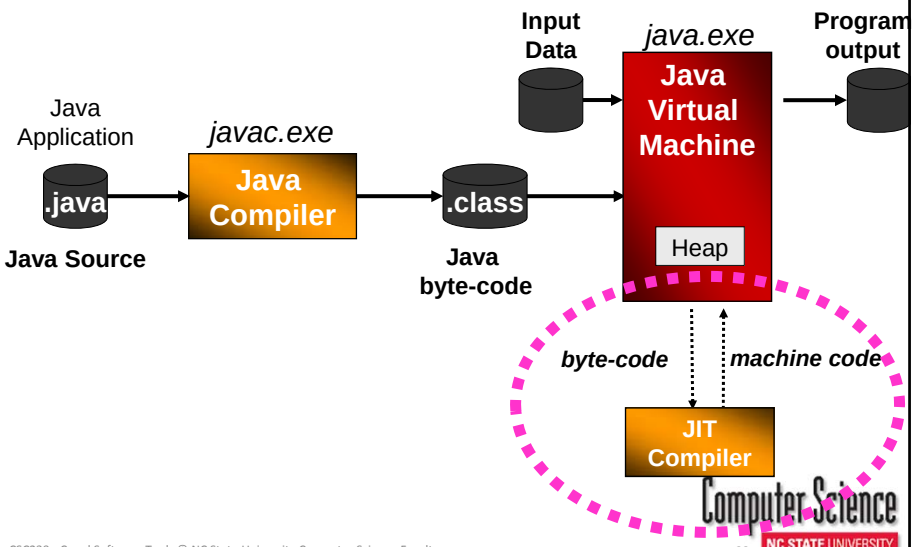
Platform Independence?

- Compiled
  - parts of the compiler (*front end*) are platform-independent
  - parts of the compiler (*back end*) are specific to the platform on which the program will be executed
- Interpreted
  - the Java compiler is platform-independent
  - the JVM is platform-specific

"Just-in-Time" Compiling

- Idea: compile a method to machine code just before first use
  - and reuse that machine code each time the method is invoked
- Benefits of interpreted + speed of compiled

JVM, Again



Computer Science NC STATE UNIVERSITY

Comparison

Property	Better Compiled, or Interpreted?
Execution Speed	?
Error messages, debugging support	?
Platform Independence / Portability	?

- Another (major) benefit of interpreted languages: dynamic typing of variables
  - not supported in Java, however

Computer Science NC STATE UNIVERSITY

Steps in Compiling C Programs

- Source Code

```
#define N 3
a=c+b*N;
```

preprocessing

Expanded Source Code

```
a=c+b*3;
```

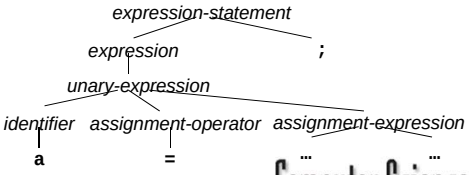
lexical analysis

Tokens

```
a  =  c  +  b  *  3  ;
```

parsing

Parse Tree



code generation

Computer Science NC STATE UNIVERSITY

Steps… (cont'd)

code generation

Assembly Language

```
mov ebx, b
imul ebx, ebx, 3
mov ecx, c
```

assembling

Object Code

```
001110010111
```

linking + other Object Code

Executable Code

```
00111001011101101010…
```


Computer Science NC STATE UNIVERSITY

Using the **gcc** Compiler

- **gcc** is a high-quality, open source compiler available for most platforms
- At the command prompt, type

```
gcc  -Wall -std=c99 <pgm.c>
```

where *<pgm.c>* is the C program source file

  - Creates an executable **a.out**.
- **-std=c99** specifies that C99 standard features are allowed
- **-Wall** turns on all the important warning messages

Computer Science NC STATE UNIVERSITY

Compiler… (cont'd)

- GNOME (and me): "Make sure your code compiles with absolutely no warnings from the compiler. These help you catch stupid bugs."

Computer Science NC STATE UNIVERSITY

Some Useful **gcc** Options

-c	Compile the source code but do not link (i.e., produce only the object file)
-E	Preprocess the source code only (i.e., expand macros, but do not compile the source code)
-o *file*	Put output in file named **file**
--version	Display version number of gcc
-std=c99	Support C99 language features
-Wall	Enable all warnings
-g	Produce information necessary to debug using gdb

gcc Options… (cont'd)

-O, -O1	Various optimization levels
-D *name*	Define name as a macro with value 1 (used for conditional compilation)
-l*lib*	Search named **library** when linking
-I*dir*	Add directory **dir** to the head of the list of directories to search for header files
-L*dir*	Add directory **dir** to the list of directories to search for libraries containing object files (specified using the **-l** option)

A Word About C99

- The generations of C
  - K&R C
  - C89 (or C90)
  - C99

  } ISO standards
- We will use C99 in this course
  - for the most part, C99 adds to / clarifies earlier versions, does not invalidate earlier code

Computer Science NC STATE UNIVERSITY

(Some) Differences C89↔C99

1. *Comments allowed to be C++ style (//)*
2. ***`_Bool`** macro is available*
3. *Additional library functions, and a few new header files*
4. *Variable length arrays*
5. *Variable declarations can appear anywhere in the code block*
6. *Variable declarations in **`for`** loops*
7. *Support for non-ASCII character sets ("wide" characters)*

Computer Science NC STATE UNIVERSITY

(Some) Differences… (cont'd)

8. New **long long** integer data type
9. Functions must declare a return value
10. Macros may have variable number of arguments, denoted by ellipsis (**…**)
11. Functions may be inlined
12. Restricted pointers (prevent aliasing)

Computer Science NC STATE UNIVERSITY

C99… (cont'd)

- gcc 4.4.6 supports most of C99, but you may not be able to use…
  - wide characters
  - variable length arrays
  - complex numbers
  - extended integer types (**long long**)

Computer Science NC STATE UNIVERSITY

Console I/O in C

- I/O is provided by standard library functions
  - available on all platforms
- To use, your program must have

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

- …and it doesn't hurt to also have

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

- *These are preprocessor statements; the .h files define function types, parameters, and constants from the standard library*

Streams

- A *stream* is a file or a device from which data is read, and/or to which data is written
- By default, every C program automatically has 3 open streams, called
  - the *standard input*
  - the *standard output*
  - the *standard error*
- If you do not override them…
  - standard input = the keyboard
  - standard output & error = the terminal window

Streams… (cont'd)

- Note: the EOF character on your keyboard is either **`ctrl-d`** (Unix, Linux, Mac OS X) or **`ctrl-z`** (Windows)
- You can redirect the standard input from a file, e.g.,

```
pgm99 < infile.txt
```

- You can redirect the standard output to a file, e.g.,

```
pgm99 > outfile.txt
```


Reading One Character from Standard Input

- Definition (from `stdio.h`):

```
int getchar(void)
```

```
int c;

c = getchar();
if (c == EOF)
    …
```

Notes

- **EOF** defined in **stdio.h**
- declaring **c** as type **char** and then comparing to **EOF** may fail ☹

Writing One Character to Standard Output

Definition (from `stdio.h`):

```
int putchar(int c)
```

```
char c;
int b;
…
b = putchar((int) c);
if (b == EOF)
    …
```


Program **echochar.c**

```
#include <stdio.h>

int main ( void )
{
    int c;
    c = getchar();
    while (c != '\n') {
        (void) putchar(c);
        c = getchar();
    }
    putchar('\n');

    return 0;
}
```

Example: `echochar.c`

Demo…

- Keyboard input vs. input from a file
  - use editor to type the input in a file called **`in.txt`**
  - then run **`echochar`** with input redirected from the file

```
% ./echochar < in.txt
```

- No changes to the program!

Demo…

Computer Science NC STATE UNIVERSITY

The `printf()` function

- **`putchar()`** is too cumbersome to use for extensive, formatted output
- **`printf()`** is a much more convenient library function for formatted output, with built-in conversions of input parameters to printable form
- Def: **`int printf(const char * format, …)`**
  - variable number of arguments
- **`format`** specifies how input arguments must be converted/formatted for output

Computer Science NC STATE UNIVERSITY

Parts of `format`

1. % (mandatory)
2. 0 or more flags (infrequently used)
3. Minimum output field width (pad with spaces) (useful for making things line up)
4. .Precision (minimum number of digits to right of decimal point) (optional, default is 6 digits)
5. type of format conversion (mandatory)

Precision Matters

- `printf` the number 33.3:

Format Specifier	Output
`%7.1f`	`  33.3`
`%14.10f`	`33.3000000000`
`%.20f`	`33.29999999999999715783`

Some Types of Conversions

Print as Type…	Specifier
`char`	`%c`
`unsigned int`	`%u` (in decimal) `%o` (in octal) `%x`, `%X` (in hex) (`%lu`, `%lo`, `%lx` for long)
`signed int`	`%d`, `%i` (in decimal) (`%ld`, `%li` for long)
`float`	`%f`
`float`	`%e`, `%E` (use scientific notation)
(string)	`%s`

Example

- Program

```
char c = 'a';
int i = 9999;
float f = 3.1415926535897932;

printf("c = %c (%o in octal)\n", c, c);
printf("i = %6d (%x in hex)\n", i, i);
printf("f = %8.5f (%e in sci. notation)\n",
        f, f);
```

Output:

```
c = a (141 in octal)
i =   9999 (270f in hex)
f =  3.14159 (3.141593e+00 in sci. notation)
```

Reminder

- Base 16 (“hex”):
- $2F3_{16} = 2 * 16^2 + 15 * 16^1 + 3 = 755_{10}$
- Base 8 (“octal”):
- $463_8 = 4 * 8^2 + 6 * 8^1 + 3 = 307_{10}$

Computer Science NC STATE UNIVERSITY

Exercise 02.03: Basic I/O

- Write a program that
  - Reads 3 characters from standard input (all on one line, no spaces)
  - Outputs the characters in reverse order to standard output
- Make sure it compiles cleanly with the **`-Wall -std=c99`** options
- Make sure it is formatted cleanly and consistently
- Submit through Google Form

Computer Science NC STATE UNIVERSITY