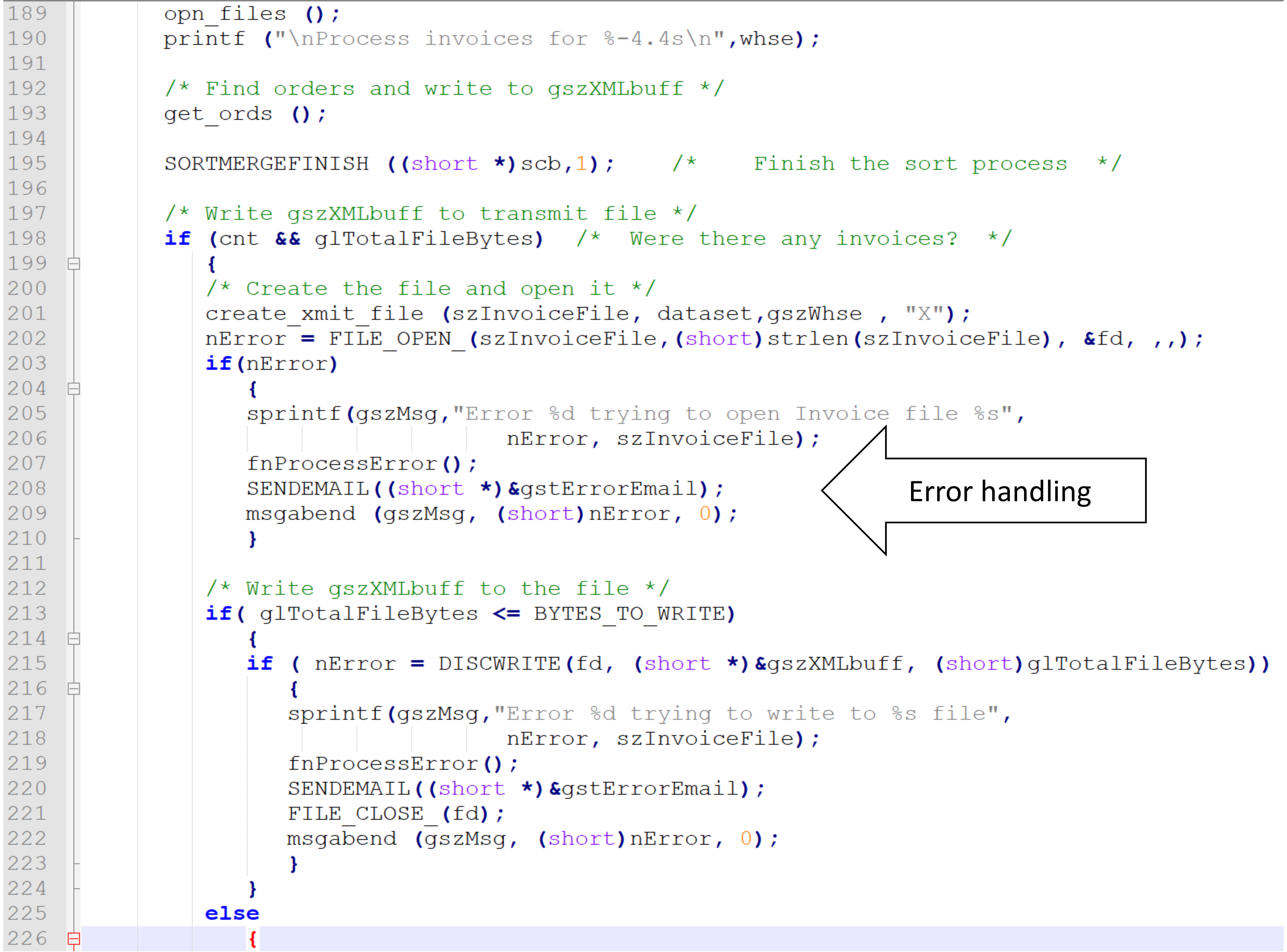
CSE 1320

Week of 01/21/2019

Instructor : Donna French

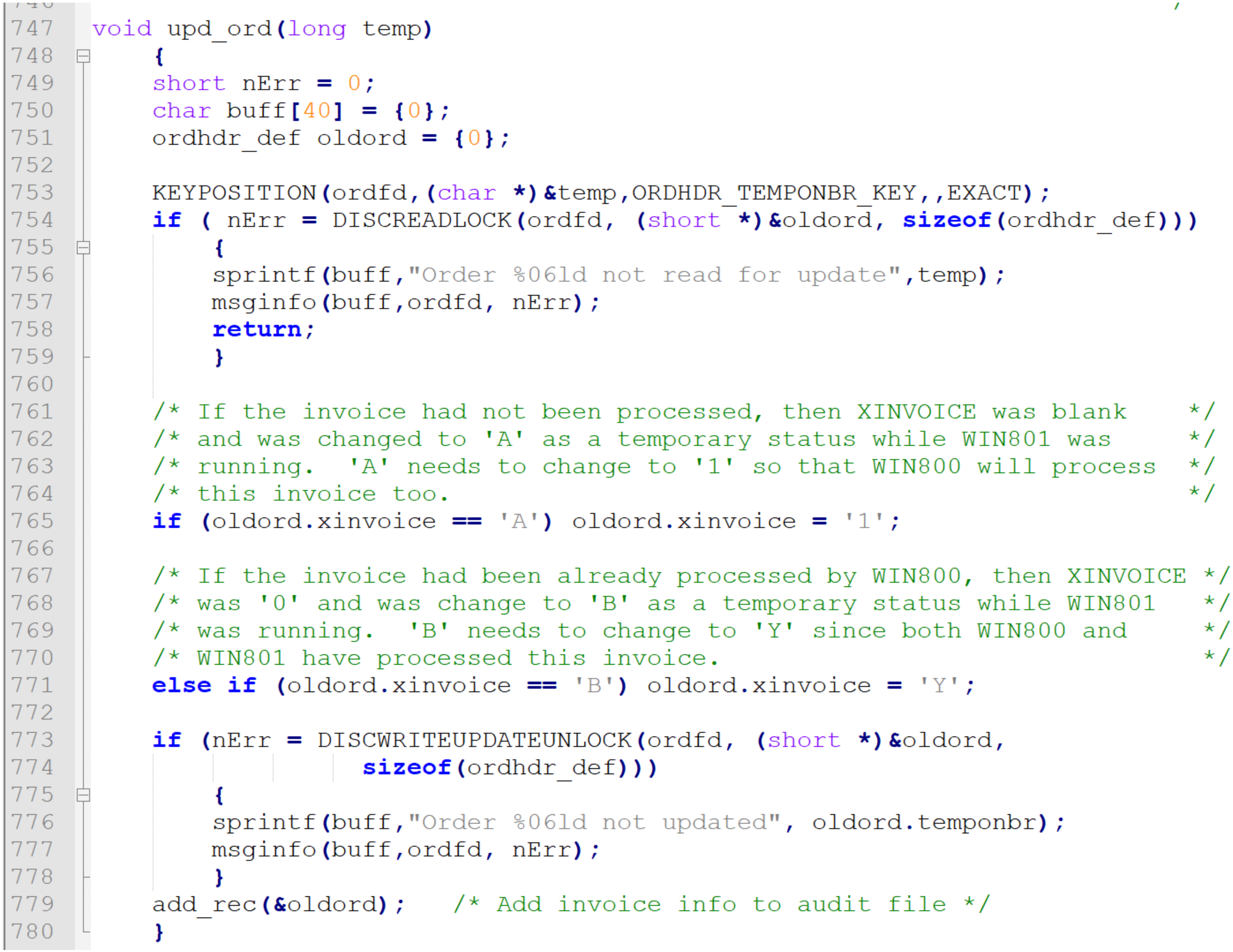
Structured Programming in C

- Write source code that is
    - modular
    - easily modifiable
    - robust (handles errors gracefully)
    - readable
- Write functions that can be used with little or no modification in many programs
- Write functions to do one task that is not too long and can be understood easily



```
opn_files ();
printf ("\nProcess invoices for %-4.4s\n",whse);

/* Find orders and write to gszXMLbuff */
get_ords ();

SORTMERGEFINISH ((short *)scb,1);    /*    Finish the sort process  */

/* Write gszXMLbuff to transmit file */
if (cnt && glTotalFileBytes)  /*  Were there any invoices?  */
   {
   /* Create the file and open it */
   create_xmit_file (szInvoiceFile, dataset,gszWhse , "X");
   nError = FILE_OPEN_(szInvoiceFile,(short)strlen(szInvoiceFile), &fd, ,,);
   if(nError)
      {
      sprintf(gszMsg,"Error %d trying to open Invoice file %s",
                         nError, szInvoiceFile);
      fnProcessError();
      SENDEMAIL((short *)&gstErrorEmail);
      msgabend (gszMsg, (short)nError, 0);
      }

   /* Write gszXMLbuff to the file */
   if( glTotalFileBytes <= BYTES_TO_WRITE)
      {
      if ( nError = DISCWRITE(fd, (short *)&gszXMLbuff, (short)glTotalFileBytes))
         {
         sprintf(gszMsg,"Error %d trying to write to %s file",
                         nError, szInvoiceFile);
         fnProcessError();
         SENDEMAIL((short *)&gstErrorEmail);
         FILE_CLOSE_(fd);
         msgabend (gszMsg, (short)nError, 0);
         }
      }
   else
      {
```

Error handling



```
void upd_ord(long temp)
    {
    short nErr = 0;
    char buff[40] = {0};
    ordhdr_def oldord = {0};

    KEYPOSITION(ordfd,(char *)&temp,ORDHDR_TEMPONBR_KEY,,EXACT);
    if ( nErr = DISCREADLOCK(ordfd, (short *)&oldord, sizeof(ordhdr_def)))
        {
        sprintf(buff,"Order %06ld not read for update",temp);
        msginfo(buff,ordfd, nErr);
        return;
        }

    /* If the invoice had not been processed, then XINVOICE was blank    */
    /* and was changed to 'A' as a temporary status while WIN801 was     */
    /* running.  'A' needs to change to '1' so that WIN800 will process  */
    /* this invoice too.                                                 */
    if (oldord.xinvoice == 'A') oldord.xinvoice = '1';

    /* If the invoice had been already processed by WIN800, then XINVOICE */
    /* was '0' and was change to 'B' as a temporary status while WIN801   */
    /* was running.  'B' needs to change to 'Y' since both WIN800 and     */
    /* WIN801 have processed this invoice.                                */
    else if (oldord.xinvoice == 'B') oldord.xinvoice = 'Y';

    if (nErr = DISCWRITEUPDATEUNLOCK(ordfd, (short *)&oldord,
                  sizeof(ordhdr_def)))
        {
        sprintf(buff,"Order %06ld not updated", oldord.temponbr);
        msginfo(buff,ordfd, nErr);
        }
    add_rec(&oldord);   /* Add invoice info to audit file */
    }
```

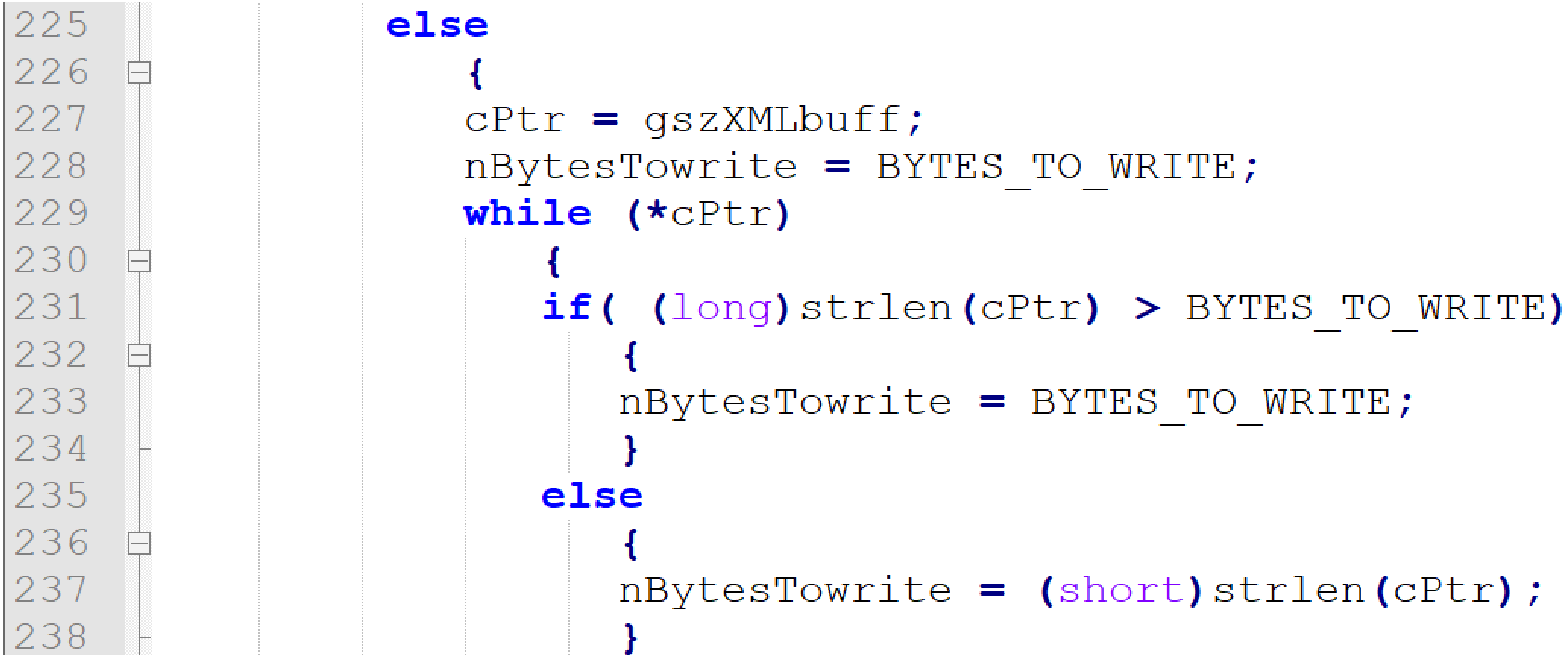
Structured Programming in C

Preprocessor Constants - Defining constants

- adds to a program’s readability
- allows a program to be more easily modified

```
#define SUNDAY    0
#define MONDAY    1
#define TUESDAY   2
#define WEDNESDAY 3
#define THURSDAY  4
#define FRIDAY    5
#define SATURDAY  6
```

```
#define BYTES_TO_WRITE 1096
#define NBR_STATS 3
#define ORDHDR_TEMPONBR_KEY ‘TN’
#define FALSE 0
#define TRUE 1
```



```
else
   {
   cPtr = gszXMLbuff;
   nBytesTowrite = BYTES_TO_WRITE;
   while (*cPtr)
      {
      if( (long)strlen(cPtr) > BYTES_TO_WRITE)
         {
         nBytesTowrite = BYTES_TO_WRITE;
         }
      else
         {
         nBytesTowrite = (short)strlen(cPtr);
         }
```

Expressions vs Statements

Expressions

sequences of tokens that can be evaluated to a numerical quantity

- can be a single number
- can be an identifier
- can be more complicated sequence of tokens
- can contain any of the operators in C
- arguments to functions

Statement

sequence of tokens terminated with a semicolon that can be recognized by the compiler

- may not have values
- purpose might be to select which set of statements to execute in a given circumstance
- purpose might be to cause a sequence of statements to be executed more than once (control statement)
- cannot be an argument to a function

lvalue vs rvalue expressions

lvalue

- an expression that has a location in memory
  - name of a variable
- expressions whose values can be either changed or evaluated
- used on the left hand side of an assignment statement

rvalue

- can be evaluated but cannot be changed
  - single character token ‘5’
- cannot be used on the left hand side of an assignment statement
- may only be used on the right side

An expression can be both an lvalue and a rvalue

```
int x;
int y;
```

Expression	**lvalue**	**rvalue**
`x`	yes	yes
`x + 3`	no	yes
`y`	yes	yes
`2*y – 7`	no	yes
`* (–2/y + 7 % x)`	no	yes

Assignment Expression

```
expr1 = expr2
```

`expr1` is an lvalue

`expr2` is an rvalue

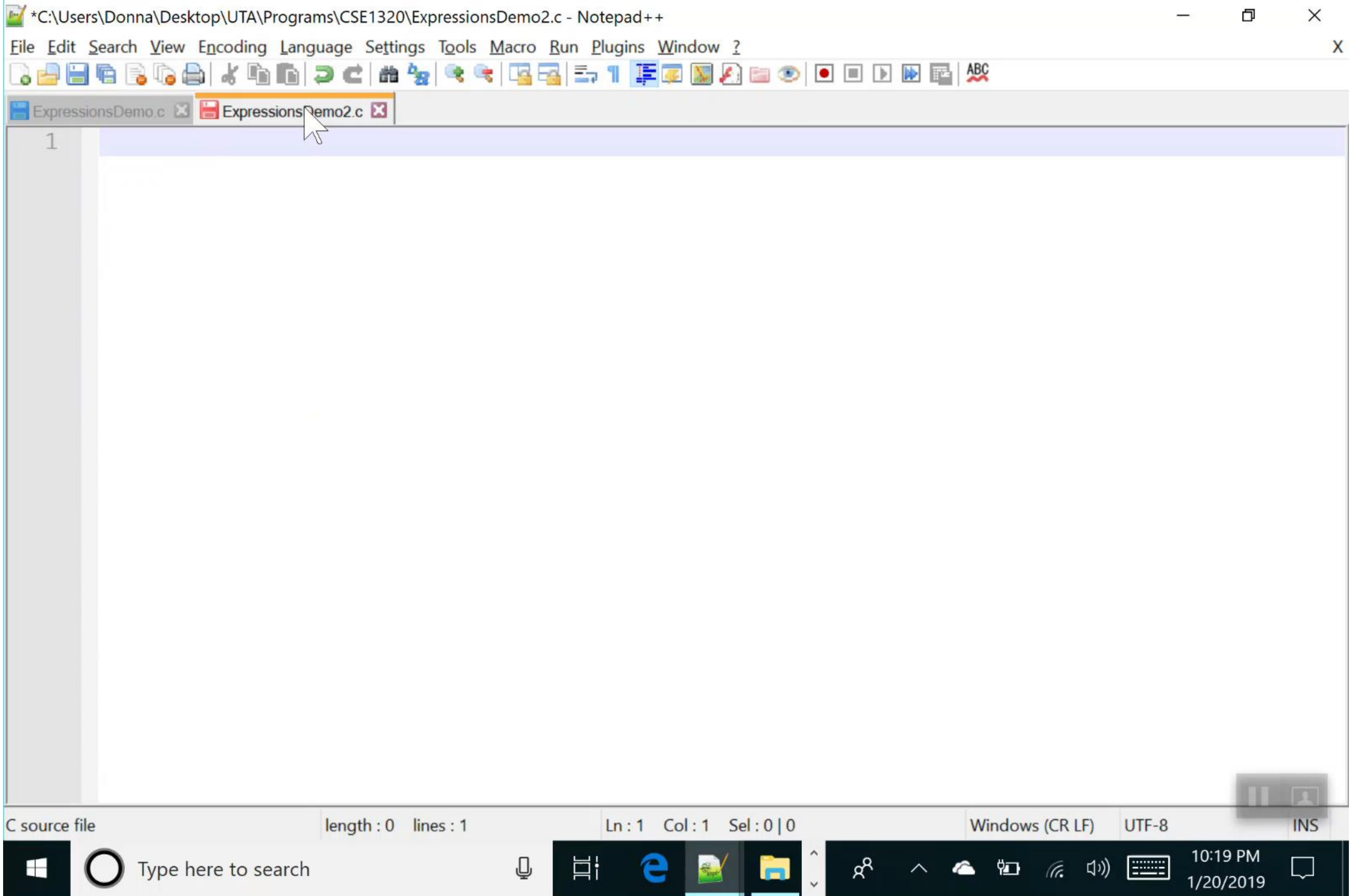
When this assignment expression is evaluated, `expr2` is fully evaluated before the assignment expression itself takes on that value

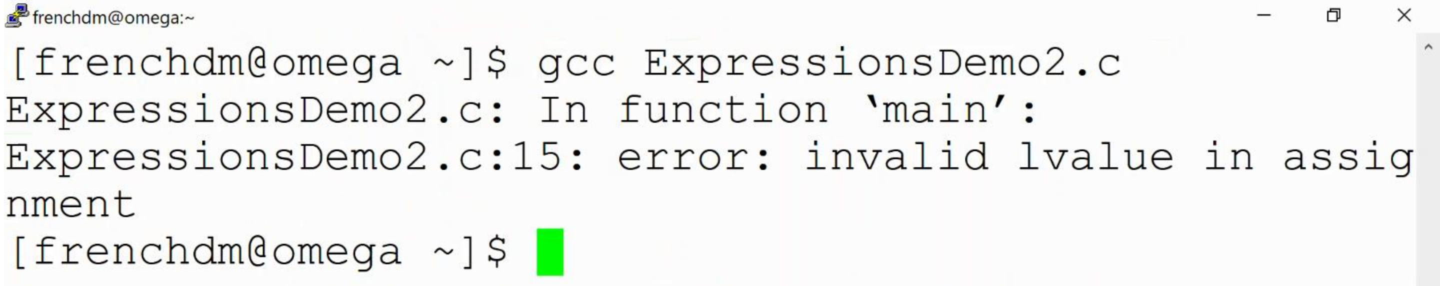
```
x = 5
```

lvalue on left hand side indicates where to store the value obtained from the evaluation of the expression on the right hand side.

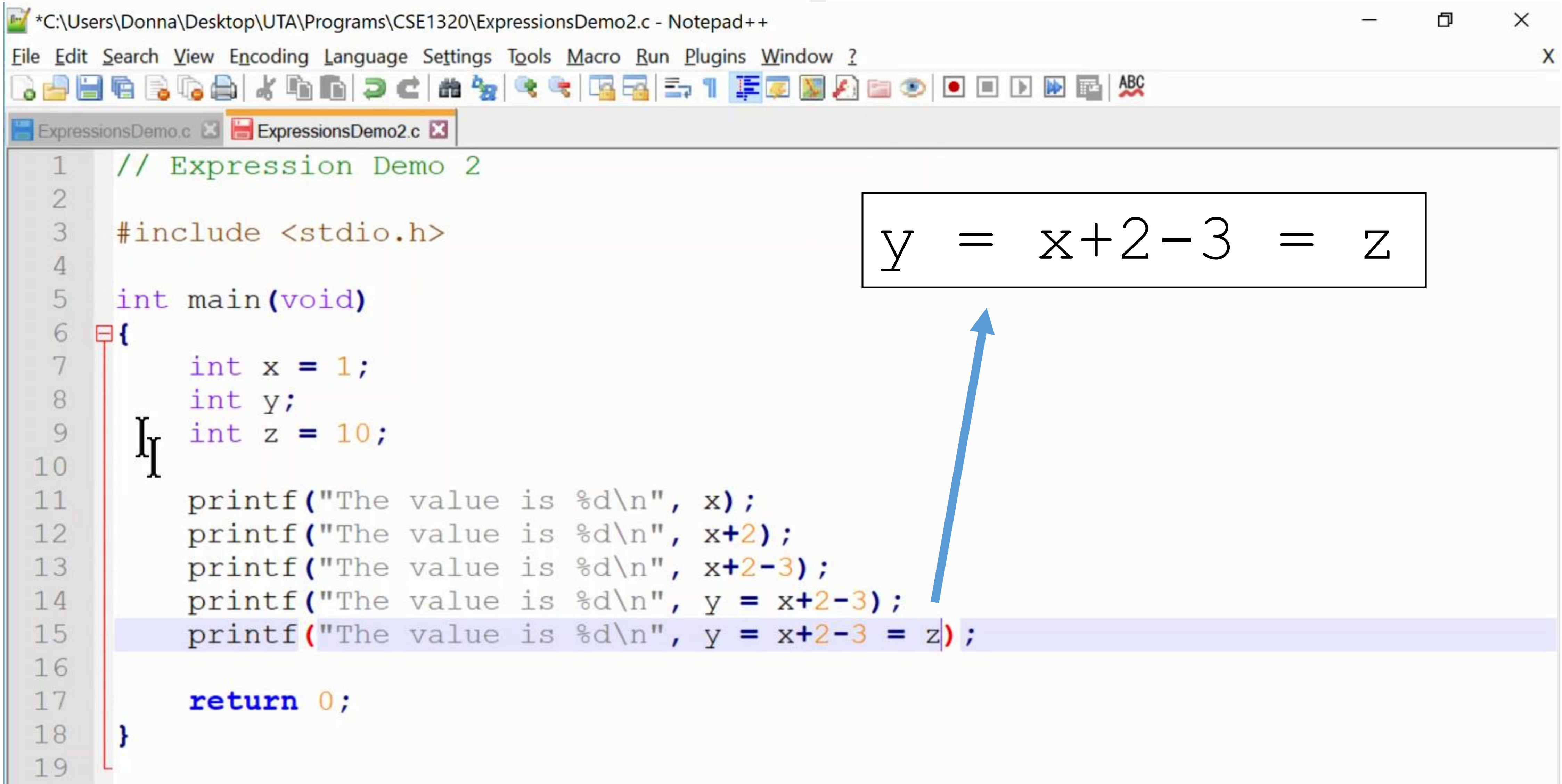
Assignment Expression

```
int x = 1;
int y = 1;
int z = 1;


z = y = 4*x + 5;   ✓


z + 1 = y = 4*x + 5;   ✗
z = y + 1 = 4*x + 5;   ✗
```


*C:\Users\Donna\Desktop\UTA\Programs\CSE1320\ExpressionsDemo2.c - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
ExpressionsDemo.c
ExpressionsDemo2.c
C source file
length : 0 lines : 1
Ln : 1 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS
Type here to search
10:19 PM
1/20/2019



```
[frenchdm@omega ~]$ gcc ExpressionsDemo2.c
ExpressionsDemo2.c: In function ‘main’:
ExpressionsDemo2.c:15: error: invalid lvalue in assig
nment
[frenchdm@omega ~]$ 
```


*C:\Users\Donna\Desktop\UTA\Programs\CSE1320\ExpressionsDemo2.c - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
ExpressionsDemo.c
ExpressionsDemo2.c

```
1   // Expression Demo 2
2
3   #include <stdio.h>
4
5   int main(void)
6   {
7       int x = 1;
8       int y;
9       int z = 10;
10
11      printf("The value is %d\n", x);
12      printf("The value is %d\n", x+2);
13      printf("The value is %d\n", x+2-3);
14      printf("The value is %d\n", y = x+2-3);
15      printf("The value is %d\n", y = x+2-3 = z);
16
17      return 0;
18  }
19
```

y = x+2-3 = z

Blocks and Compound Statements

Compound Statement

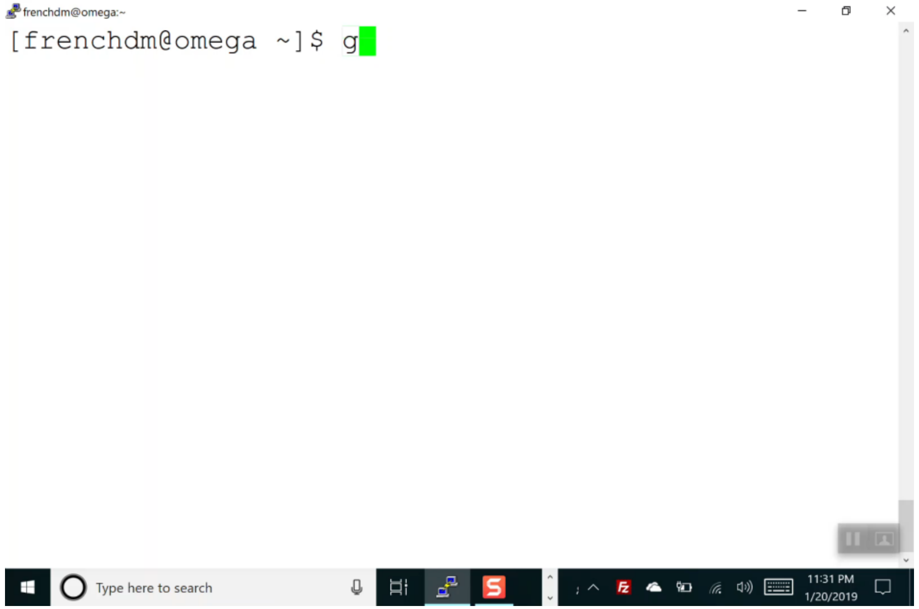
- sequence of statements that can be used anyplace in the syntax that a simple statement can be used
- the construct that implements a compound statement is called **block**

Block

- every function must have a function block
- must begin with an opening brace and terminate with a closing brace

```
int main (void)
{
   return 0;
}
```

```
int VarGlobal = 1;

int main (void)
{
   int VarLocalToMain = 2;

   {
      int VarLocalToBlock = 3;
      printf("Value of VarLocalToMain is %d\n", VarLocalToMain);
      printf("Value of VarLocalToBlock is %d\n", VarLocalToBlock);
      printf("Value of VarGlobal is %d\n", VarGlobal);
   }

   printf("Value of VarLocalToMain is %d\n", VarLocalToMain);
   printf("Value of VarLocalToBlock is %d\n", VarLocalToBlock);
   printf("Value of VarGlobal is %d\n", VarGlobal);

   return 0;
}
```



```
[frenchdm@omega ~]$ g
```

The if and if-else Statements

if

- conditional statement
- allows a program to test a condition and then choose which code to execute next
- the choice depends on the outcome of that test

```
if (expression)
   statement
```

If `expression` evaluates to TRUE, then `statement` will be executed.
If `expression` evaluates to FALSE, then `statement` will not be executed.

The if and if-else Statements

if-else

- conditional statement
- allows a program to test a condition and then choose which code to execute next
- the choice depends on the outcome of that test

```
if (expression)
   statement1
else
   statement2
```

If `expression` evaluates to TRUE, then `statement1` will be executed.
If `expression` evaluates to FALSE, then `statement2` will not be executed.

```
if (DayOfWeek == SUNDAY)
{
   printf("Today is Sunda
}
if (DayOfWeek == MONDAY)
{
   printf("Today is Monda
}
if (DayOfWeek == TUESDAY)
{
   printf("Today is Tuesd
}
if (DayOfWeek == WEDNESDA
{
   printf("Today is Wedne
}
if (DayOfWeek == THURSDAY
{
   printf("Today is Thursday and tomorrow is Friday\n");
}
if (DayOfWeek == FRIDAY)
{
   printf("Today is Friday and tomorrow is Saturday\n");
}
if (DayOfWeek == SATURDAY)
{
   printf("Today is Saturday and tomorrow is Sunday\n");
}
```

ifDemo.c

```
#define SUNDAY    0
#define MONDAY    1
#define TUESDAY   2
#define WEDNESDAY 3
#define THURSDAY  4
#define FRIDAY    5
#define SATURDAY  6
```

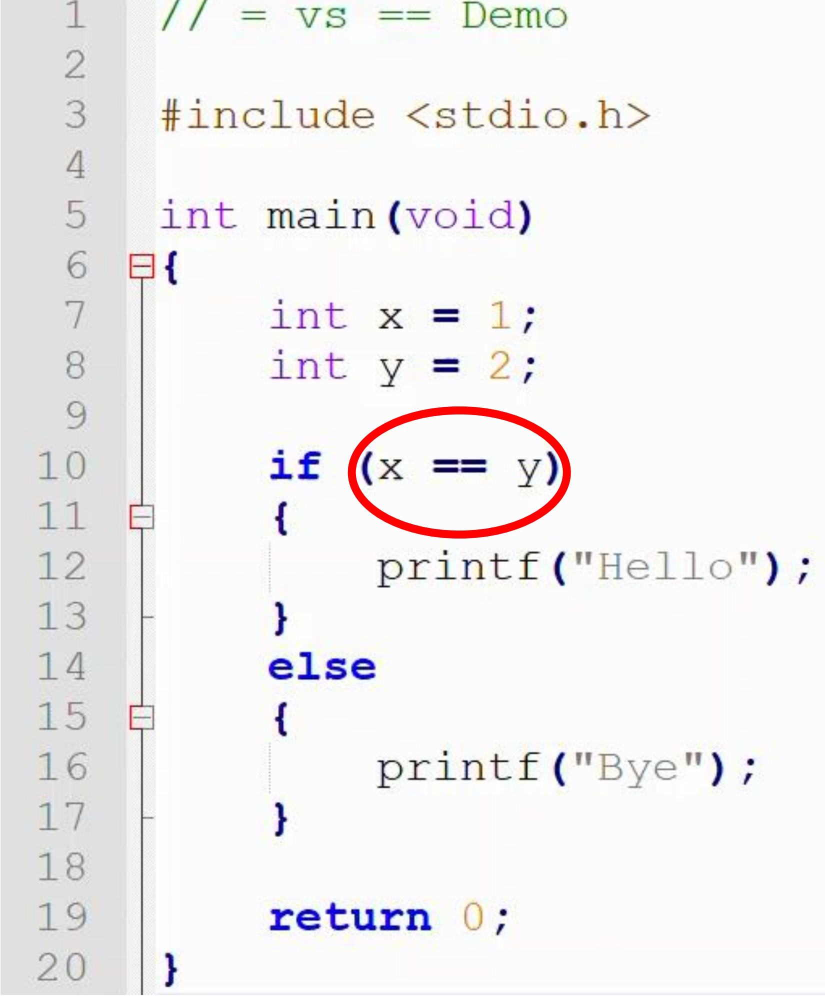
```
if (DayOfWeek == SUNDAY)
{
       ("Today is Sunday and tomorrow is Monday\n");

       DayOfWeek == MONDAY)

       ("Today is Monday and tomorrow is Tuesday\n");

       DayOfWeek == TUESDAY)

       ("Today is Tuesday and tomorrow is Wednesday\n");

       DayOfWeek == WEDNESDAY)

       ("Today is Wednesday and tomorrow is Thursday\n");

       DayOfWeek == THURSDAY)
{
   printf("Today is Thursday and tomorrow is Friday\n");
}
else if (DayOfWeek == FRIDAY)
{
   printf("Today is Friday and tomorrow is Saturday\n");
}
else
{
   printf("Today is Saturday and tomorrow is Sunday\n");
}
```

ifelseDemo.c

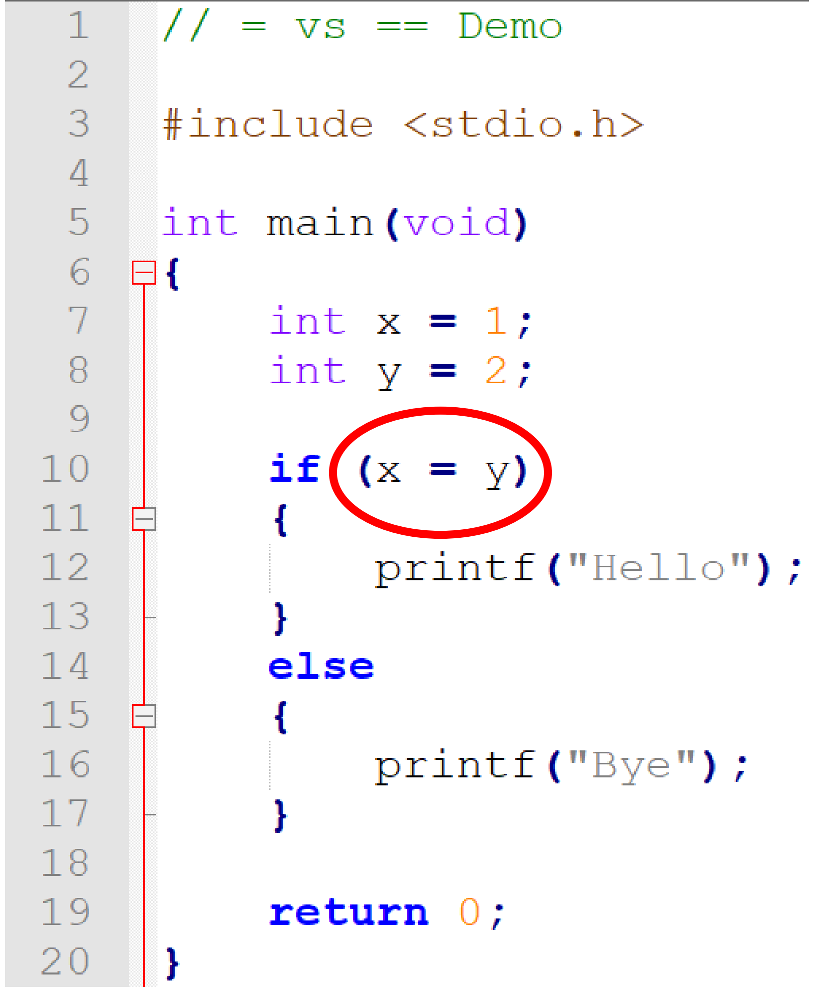
Relational Operators

is less than or equal to	<=
is greater than or equal to	>=
is equal to	==
is not equal to	!=
is greater than	>
is less than	<

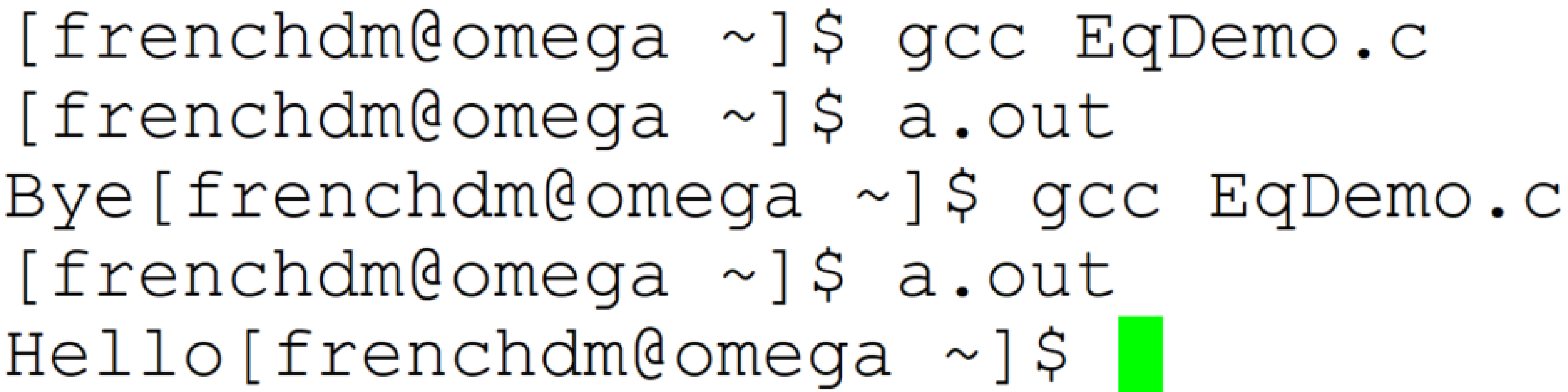
The actual value assigned to an expression formed with a relational operator is `1` if the relation is true and `0` if it is false.



```c
// = vs == Demo

#include <stdio.h>

int main(void)
{
    int x = 1;
    int y = 2;

    if (x == y)
    {
        printf("Hello");
    }
    else
    {
        printf("Bye");
    }

    return 0;
}
```



```c
// = vs == Demo

#include <stdio.h>

int main(void)
{
    int x = 1;
    int y = 2;

    if (x = y)
    {
        printf("Hello");
    }
    else
    {
        printf("Bye");
    }

    return 0;
}
```



```
[frenchdm@omega ~]$ gcc EqDemo.c
[frenchdm@omega ~]$ a.out
Bye[frenchdm@omega ~]$ gcc EqDemo.c
[frenchdm@omega ~]$ a.out
Hello[frenchdm@omega ~]$ █
```

Operator Precedence

Relational operators have a lower precedence than any of the arithmetic operators.

```
4 <= z + 3              4 <= (z + 3)
4 <= x = z + 3          illegal
4 <= (x – z + 3)        legal
3 < x < 7               ((3 < x) < 7)
```

IncDemo.c

Increment/Decrement Operators

++

Increment Operator

Add 1 to a variable

Two forms

```
i++
++i
```

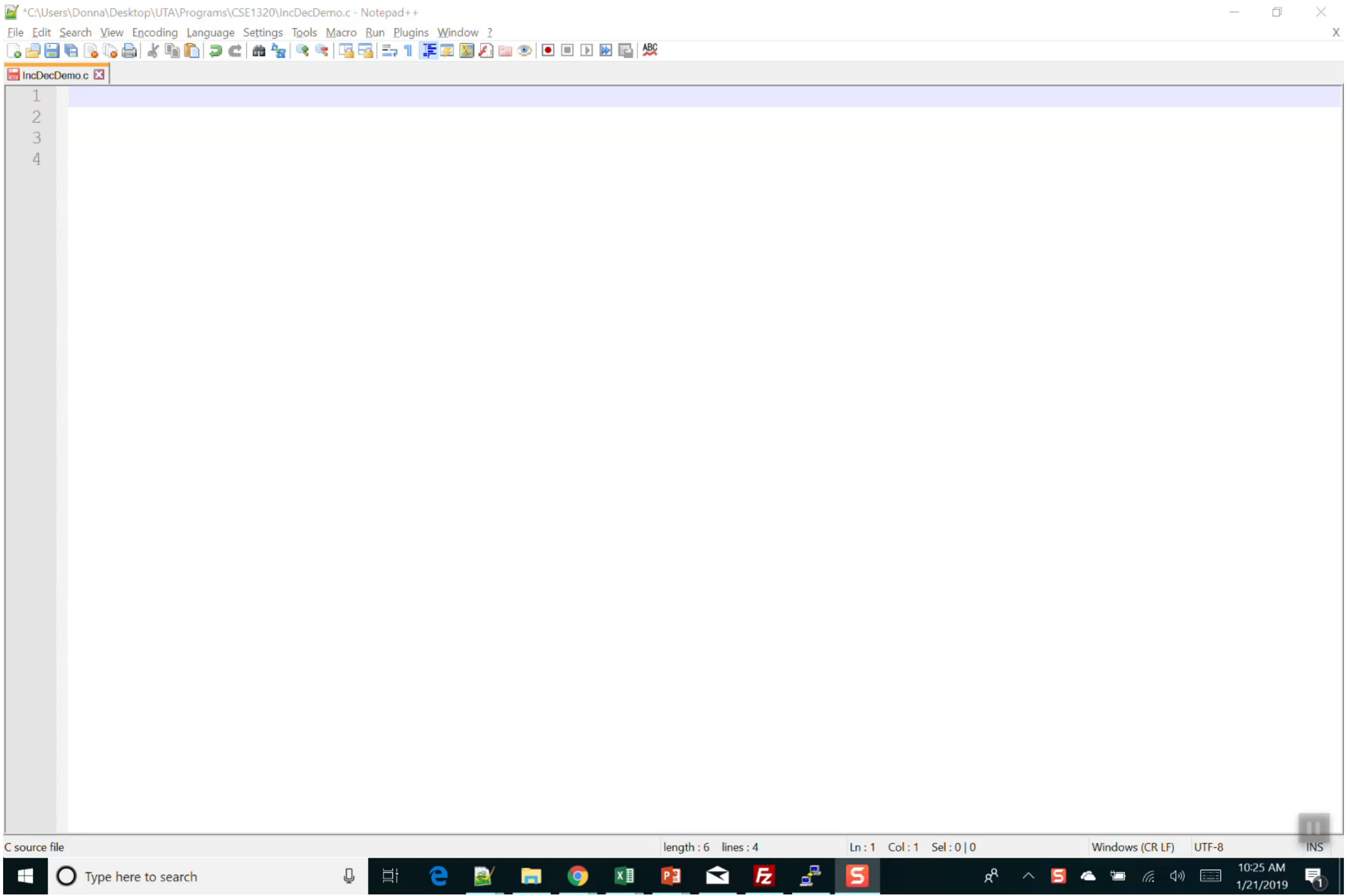
--

Decrement Operator

Subtract 1 from a variable

Two forms

```
i--
--i
```


*C:\Users\Donna\Desktop\UTA\Programs\CSE1320\IncDecDemo.c - Notepad++
File
Edit
Search
View
Encoding
Language
Settings
Tools
Macro
Run
Plugins
Window
?
IncDecDemo.c
C source file
length : 6   lines : 4
Ln : 1   Col : 1   Sel : 0 | 0
Windows (CR LF)
UTF-8
INS
Type here to search
10:25 AM
1/21/2019

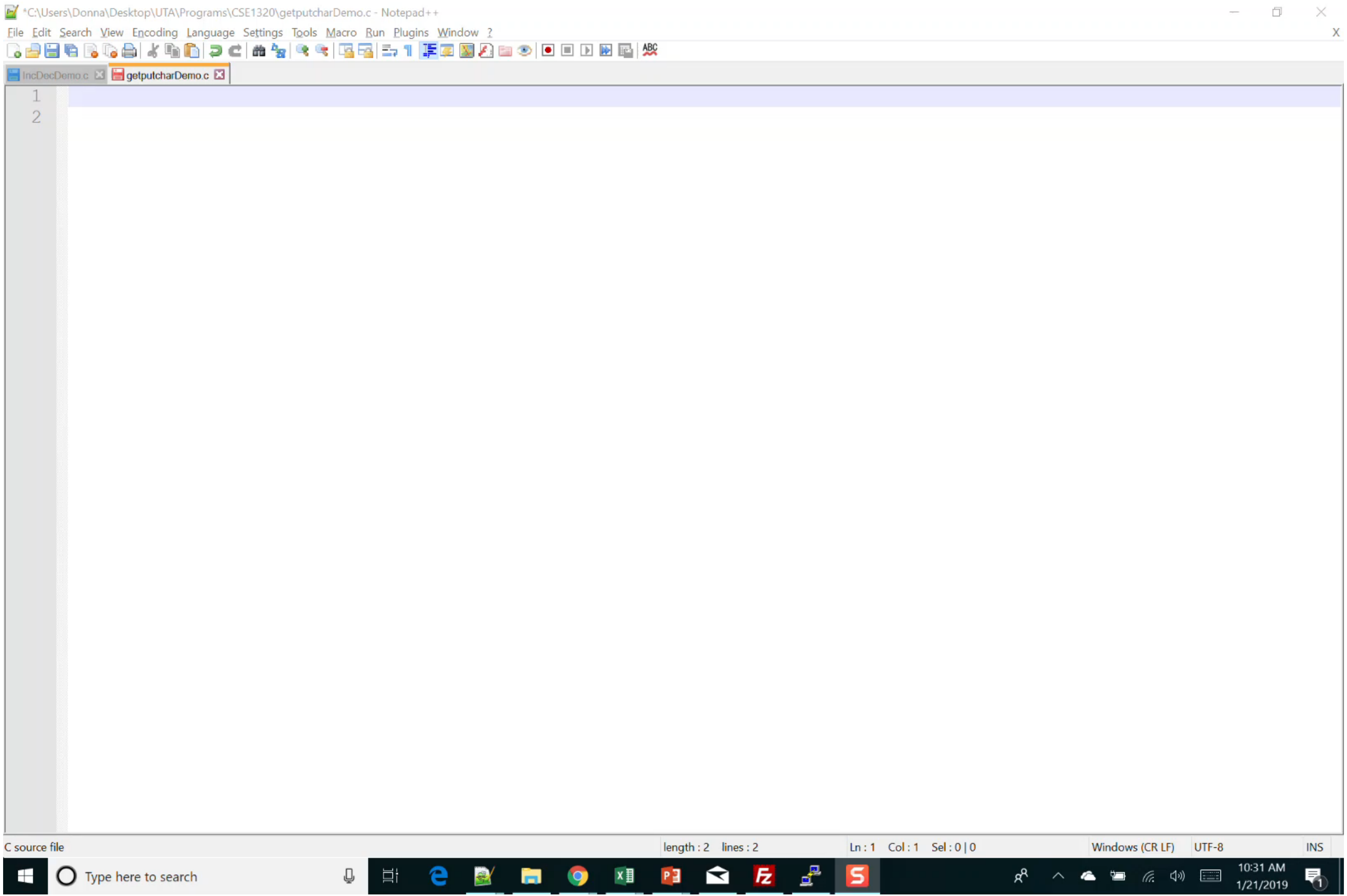
getputcharDemo.c

getchar() and putchar()

```
getchar()

int getchar(void)
```

```
putchar()

int putchar(int c)
```

```
int i;

printf("Enter a character for getchar() ");
i = getchar();
printf("Calling putchar() ");
putchar(i);
printf("\n\n");
```



length : 2 lines : 2 Ln : 1 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

The while Loop

```
while (expression)
    statement
```

Step 1 : `expression` is evaluated

Step 2 : if `expression` is true (nonzero), then `statement` is executed

Step 3 : Return to Step 1

Iteration

executing a code segment more than once

getputwhileDemo.c

```
int main(void)
{
   int iochar;
   int LoopCounter = 0;
   int ENTERCounter = 0;
   int CharCounter = 0;

   iochar = getchar();

   while (iochar != EOF)
   {
      if (iochar == '\n')
         ENTERCounter++;
      else
         CharCounter++;

      putchar(iochar);
      iochar = getchar();
      LoopCounter++;
   }

   printf("You entered EOF - bye!!\n\n");
   printf("The while loop was executed %d times\n\n\n",
            LoopCounter);

   printf("ENTERCounter is %d\nCharCounter is %d\n\n",
           ENTERCounter, CharCounter);

   return 0;
}
```

Ctrl-D from the keyboard makes EOF

```
1Hello
1Hello
2There!
2There!
3How
3How
4are
4are
5you?
5you?
You entered EOF - bye!!

The while loop was
executed 31 times


ENTERCounter is 5
CharCounter is 26
```

Logical Operators and Expressions

<table>
<tr><td>logical-not</td><td>!</td></tr>
<tr><td>logical-and</td><td>&&</td></tr>
<tr><td>logical-or</td><td>||</td></tr>
</table>

```
!expression1

expression1 && expression2

expression1 || expression2
```

logicaloperatorDemo.c

Logical Operators and Expressions

<table>
<tr><td>logical-not</td><td>!</td></tr>
<tr><td>logical-and</td><td>&&</td></tr>
<tr><td>logical-or</td><td>||</td></tr>
</table>

<table>
<tr><th>p</th><th>q</th><th>!p</th><th>!q</th><th>p && q</th><th>p || q</th><th>!(p && q)</th><th>!(p || q)</th><th>!p && !q</th><th>!p || !q</th></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

Precedence of the Logical Operators

Logical-not (!) has higher precedence than the logical-and (&&) which has higher precedence than the logical-or (||)

- logical-not
  - logical-and
    - logical-or

- Left to right evaluation

```
i || !j || !k          (i || (!j)) || (!k)
i && !j && !k          (i && (!j)) && (!k)
i || !j && !k           i || ((!j) && (!k))
```

Logical Operators and Expressions

Caution

C only evaluates as much as necessary to determine the truth value.

&&

The second operand will only be evaluated when the first operator is nonzero.

||

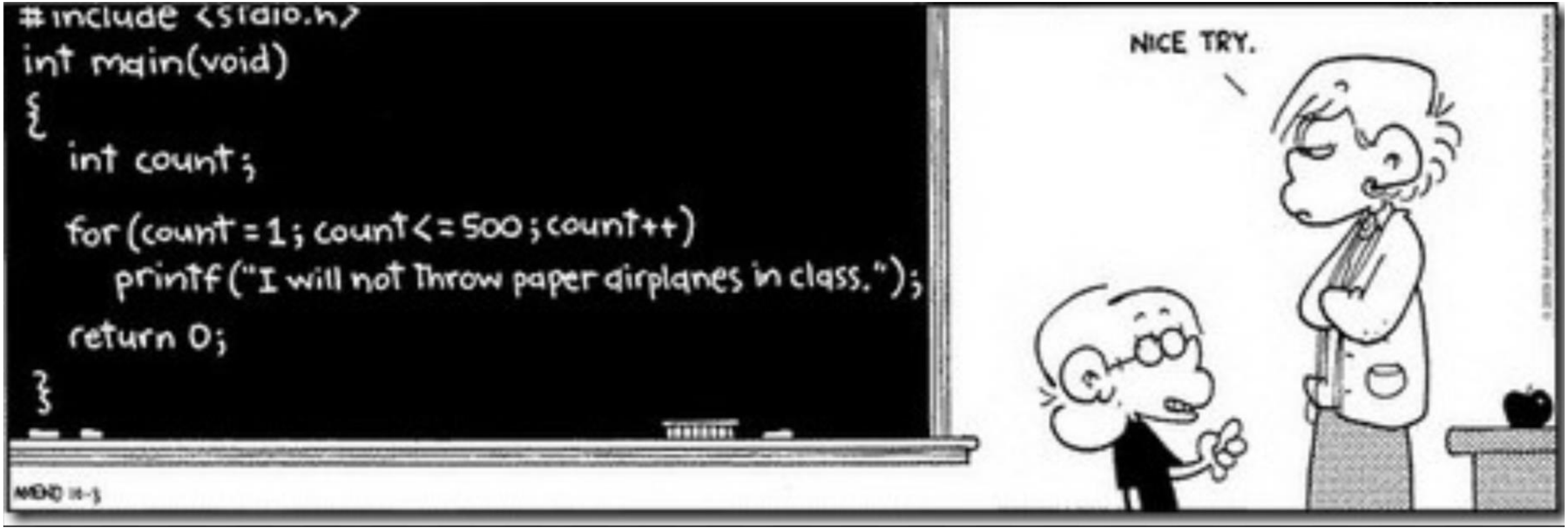
If the first operand is nonzero, then the second operand is not evaluated.

logicalevaluationDemo.c

```c
int main(void)
{
   int i = 0;
   int j = 0;

   printf("i = %d   j = %d\n\n", i, j);
   printf("i && j++ evaluates to %d\n\n", i && j++);
   printf("i = %d   j = %d\n\n", i, j);
   printf("i || j++ evaluates to %d\n\n", i || j++);
   printf("i = %d   j = %d\n\n", i, j);

   printf("Resetting i and j to 0...\n\n");
   i = j = 0;

   printf("i = %d   j = %d\n\n", i, j);
   printf("i && ++j evaluates to %d\n\n", i && ++j);
   printf("i = %d   j = %d\n\n", i, j);
   printf("i || ++j evaluates to %d\n\n", i || ++j);
   printf("i = %d   j = %d\n\n", i, j);
   return 0;
}
```

```
i = 0   j = 0

i && j++ evaluates to 0

i = 0   j = 0

i || j++ evaluates to 0

i = 0   j = 1

Resetting i and j to 0...

i = 0   j = 0

i && ++j evaluates to 0

i = 0   j = 0

i || ++j evaluates to 1

i = 0   j = 1
```

The for Loop



```
#include <stdio.h>
int main(void)
{
   int count;

   for(count=1; count<=500; count++)
       printf("I will not Throw paper airplanes in class.");

   return 0;
}
```

The for Loop

```
for (initialization; test; processing)
    statement
```

`initialization`

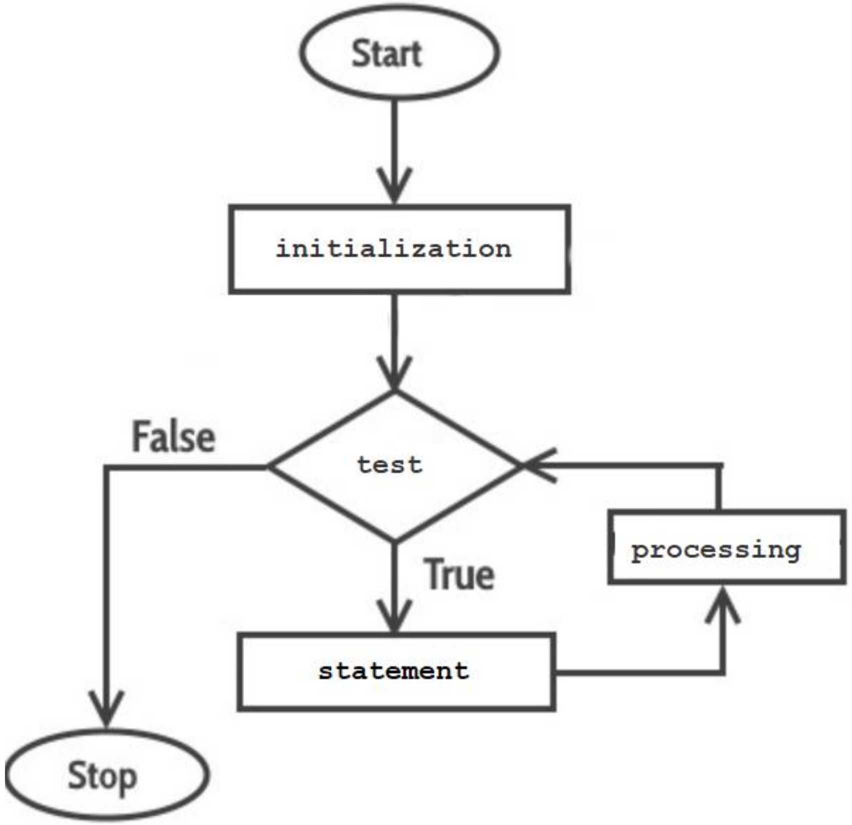
- expression that is evaluated once as the loop is entered

`test`

- expression that is evaluated as a condition for continuing the loop
    - if the value of `test` is nonzero, statement is executed
    - if the value of `test` is zero, the execution of the for loop is terminated

`processing`

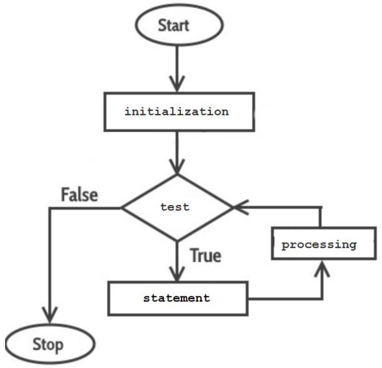
- expression that is evaluated after statement is executed each time through the loop
- does bottom-of-the-loop processing



any or all of the three expressions may be omitted

forloopDemo.c

The `for` Loop

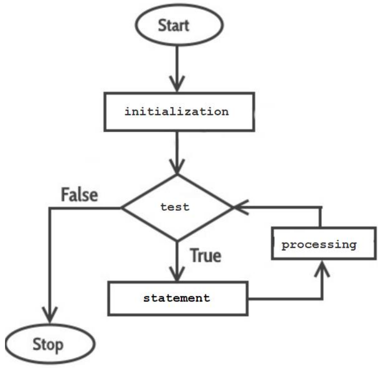


```
int i;

for (i = 0; i <= 3; i++)
   printf("i = %d\n", i);

i = 0
i = 1
i = 2
i = 3
```

```
int i;

for (i = -3; i <= 3; i++)
   printf("i = %d\n", i);

i = -3
i = -2
i = -1
i = 0
i = 1
i = 2
i = 3
```

forloopDemo.c

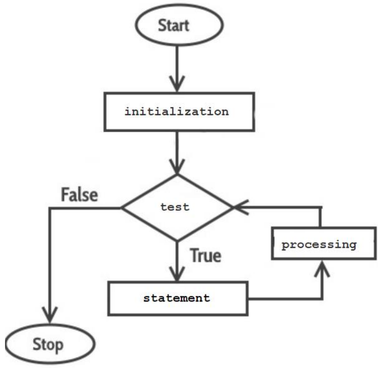
The for Loop



```
int i;

for (i = 4; i > 0; i--)
   printf("i = %d\n", i);

i = 4
i = 3
i = 2
i = 1
```

```
int i;

for (i = 2; i > -2; i--)
   printf("i = %d\n", i);

i = 2
i = 1
i = 0
i = -1
```

forloopDemo.c



The `for` Loop

```
int i;

for (i = 1; i < 16; i+=4)
   printf("i = %d\n", i);

i = 1
i = 5
i = 9
i = 13
```

```
int i;

for (i = 16; i > 0; i/=3)
   printf("i = %d\n", i);

i = 16
i = 5
i = 1
```

The `for` Loop

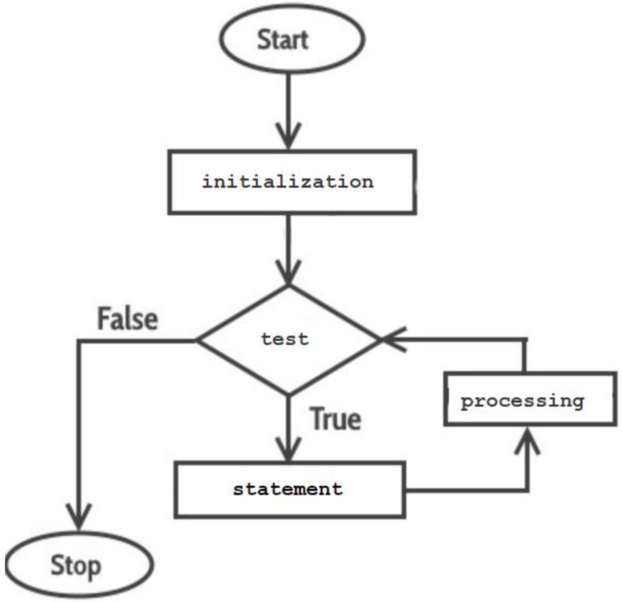
```
for (i = 1; i < 10; i+=3)
printf("i = %d\n", i);
printf("\n\ni = %d\n", i);
```

```
i = 1
i = 4
i = 7
```

```
for (i = 1; i < 10; i+=3)
   printf("i = %d\n", i);
printf("\n\ni = %d\n", i);
```

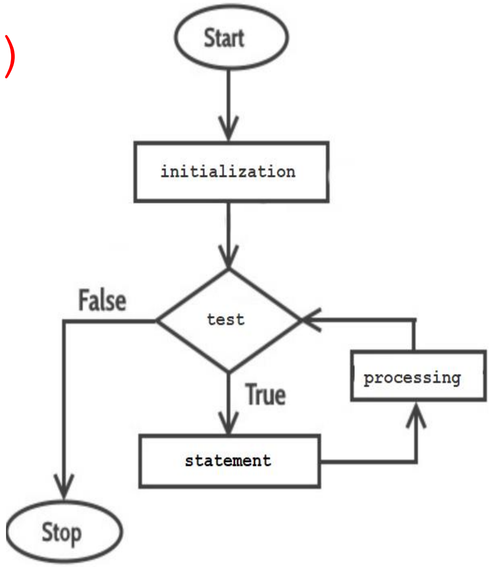
```
i = 10
```

```
for (i = 1; i < 10; i+=3)
{
   printf("i = %d\n", i);
}
printf("\n\ni = %d\n", i);
```



```
int forloopCounter = 0;
int rock = 1, paper = 45, scissors = 122, lizard= -10, Spock = 100;

for (rock = 20; paper > 3; scissors/=39, lizard++, Spock-=3, paper /=2)
{
   forloopCounter++;
}

printf("\n\n\nrock(%d)\tpaper(%d)\tscissors(%d)\tlizard (%d)\tSpock(%d)\n",
        rock, paper, scissors, lizard, Spock);
```

```
rock(20)  paper(45)  scissors(122) lizard (-10) Spock(100)
rock(20)  paper(22)  scissors(3)   lizard (-9)  Spock(97)
rock(20)  paper(11)  scissors(0)   lizard (-8)  Spock(94)
rock(20)  paper(5)   scissors(0)   lizard (-7)  Spock(91)

rock(20)  paper(2)   scissors(0)   lizard (-6)  Spock(88)
```



Variables in C

Rules of the Variable

- Must be declared
- Must be assigned a type
- Compiler reserves space in memory – amount depends on type

```
int x;
long y;
short z;
char a;
```

Variable Types in C

- Scalar types
  - enumerated
  - pointer
  - arithmetic – integer types and floating point types
- Aggregate types
- Function types
- Union types
- Void type
  - Type when a function does not return a value

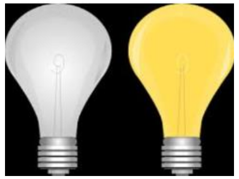
Variables in Computer Memory

Bit vs Byte vs Word

Binary Digit – Bit

0	1

holds 0 or 1 – TRUE or FALSE – ON or OFF



Byte

8 bits

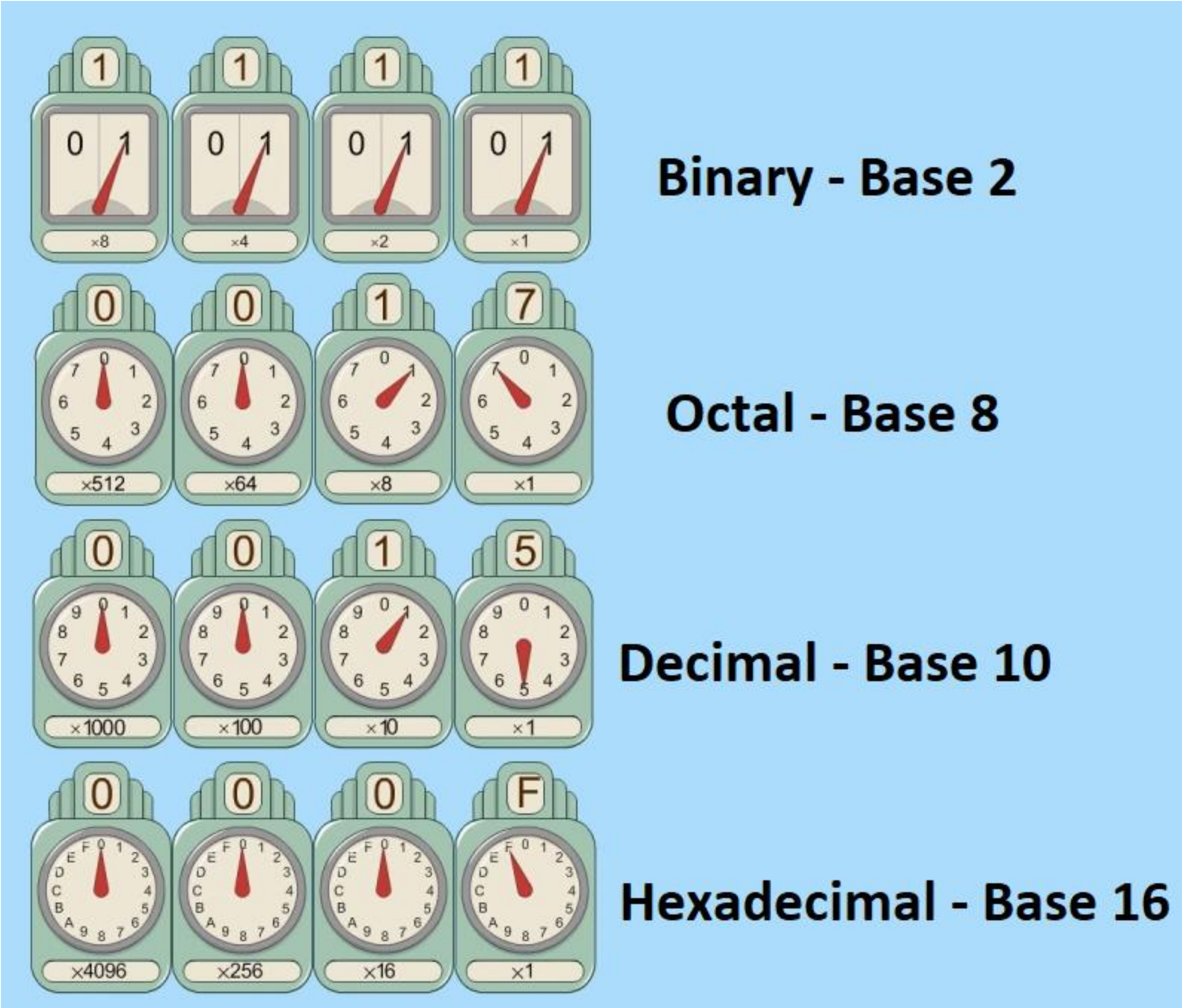
0	1	0	1	0	1	0	0

Word

one or more bytes

0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	0

Integers and Different Integer Bases



Integers and Different Integer Bases

Convert binary to decimal

Convert 11001011_2 to decimal

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	0	1	0	1	1
128	64			8		2	1

$128 + 64 + 8 + 2 +1 = 203$

$11001011_2 = 203_{10}$

Integers and Different Integer Bases

Convert decimal to binary

Convert 203_{10} to binary

Divide in half and ignore the remainder

1 ← 3 ← 6 ← 12 ← 25 ← 50 ← 101 ← 203

Write 1 for odd numbers and 0 for even numbers

1 1 0 0 1 0 1 1

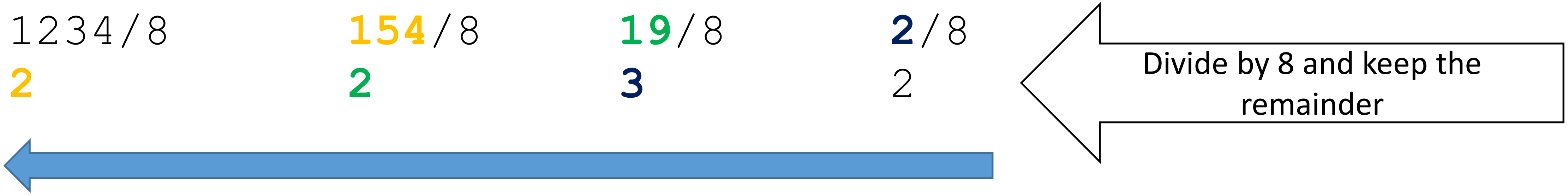
$203_{10} = 11001011_2$

Integers and Different Integers Bases

Octal

Used when the number of bits in one word is a multiple of 3

Convert 1234_{10} to octal



1234/8	**154**/8	**19**/8	**2**/8
2	**2**	**3**	2

Divide by 8 and keep the remainder

$1234_{10} = 2322_8$

Integers and Different Integers Bases

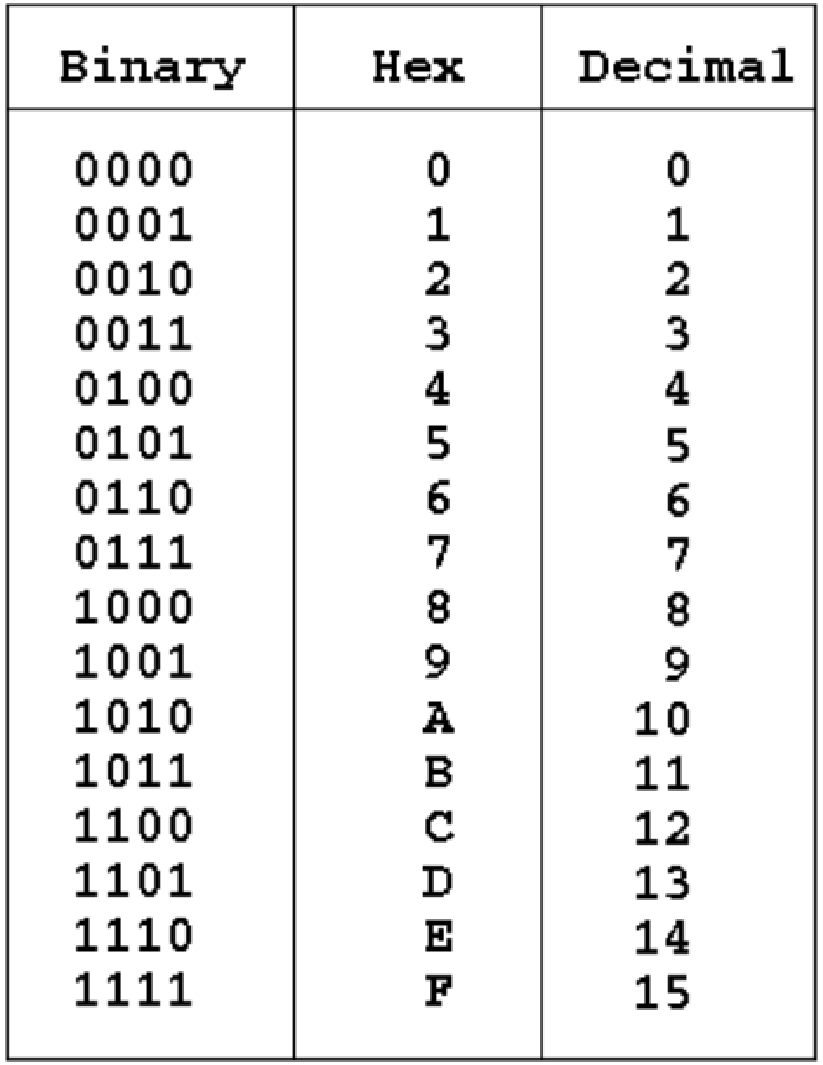
Hexadecimal

Used when the number of bits in one word is a multiple of 4

0001111100111010

0001	1111	0011	1010
1	F	3	A

$0001111100111010_2 = 1F3A_{16}$



Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

The Integer Types

`int`

- scalar type
- usually equivalent to a word
- handled more efficiently than the other types in C

- Issues
  - the size of a word varies with different hardware
    - 16 bits on one computer and 32 bits on another
  - creates portability problems
  - largest value can vary

The Integer Types

`short int`	also referred to as `short`
`long int`	also referred to as `long`

- used to avoid issues with `int`
- behave like `int` with arithmetic operators
- major difference is the number of bytes used to store each value

The Integer Types

Conversion Specifications

`%ld`	a `long` in decimal
`%lo`	a `long` in octal
`%lx` or `%1X`	a `long` in hexadecimal
`%hd`	a `short` in decimal
`%ho`	a `short` in octal
`%hx` or `%hX`	a `short` in hexadecimal

The sizeof() Operator

```
sizeof()
```

gives the number of bytes associated with a specified type or variable

The argument to `sizeof()` can be a

- type name
- variable
- expression

sizeofDemo.c

```
printf("The sizeof(short)    is %d\n", sizeof(short));
printf("The sizeof(int)      is %d\n", sizeof(int));
printf("The sizeof(long)     is %d\n", sizeof(long));
printf("The sizeof(char)     is %d\n", sizeof(char));
```

```
short shortVar;
int   intVar;
long  longVar;
```

```
printf("The sizeof(shortVar) is %d\n", sizeof(shortVar));
printf("The sizeof(intVar)   is %d\n", sizeof(intVar));
printf("The sizeof(longVar)  is %d\n\n", sizeof(longVar));

shortVar = intVar = longVar = MAX_INT;
printf("Assigning %d to shortVar, intVar, longVar\n\n", MAX_INT);

printf("The sizeof(shortVar) is %d\n", sizeof(shortVar));
printf("The sizeof(intVar)   is %d\n", sizeof(intVar));
printf("The sizeof(longVar)  is %d\n\n", sizeof(longVar));

printf("The sizeof(intVar+3/2+3*7-4)   is %d\n", sizeof(intVar+3/2+3*7-4));
```

sizeofDemo.c

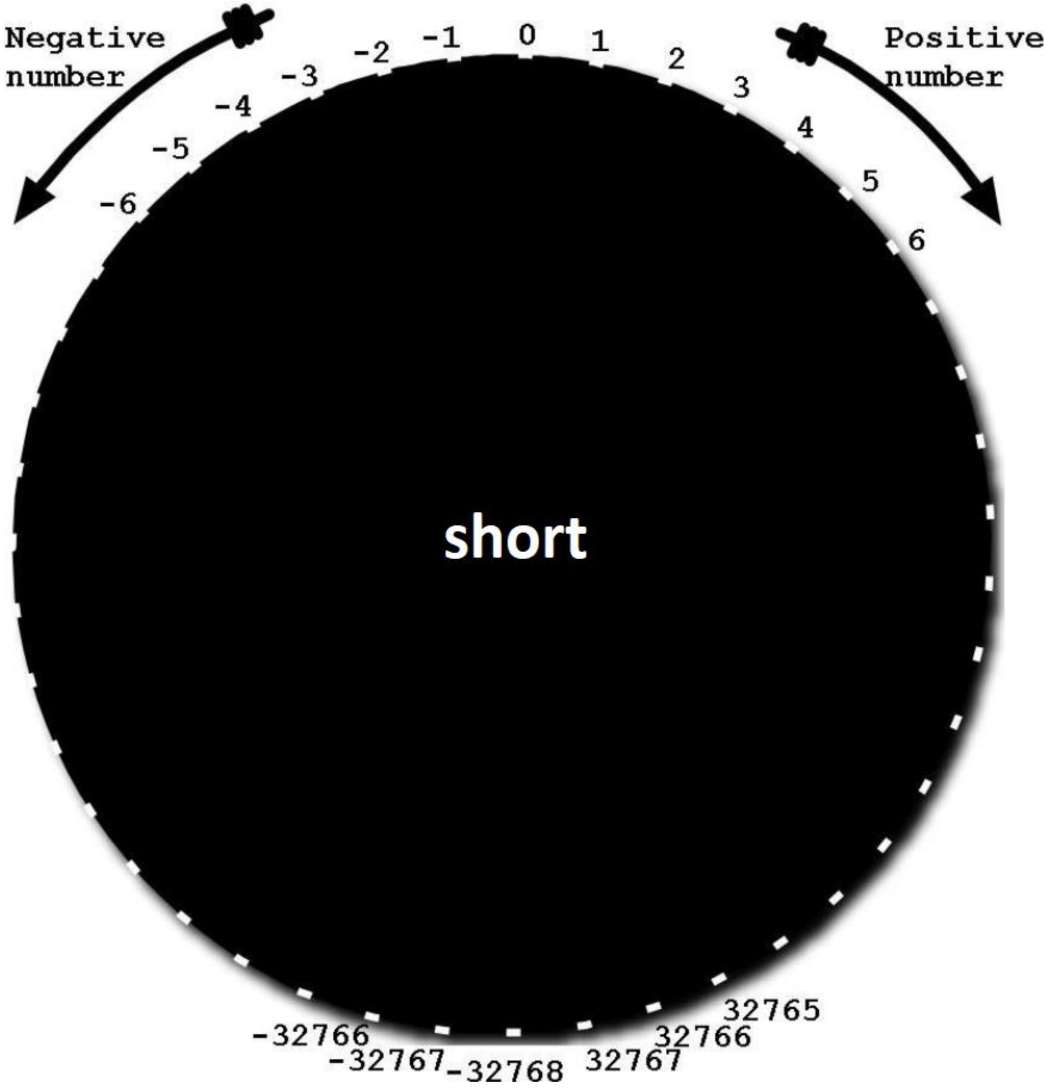
```
The sizeof(short)    is 2
The sizeof(int)      is 4
The sizeof(long)     is 8
The sizeof(char)     is 1

The sizeof(shortVar) is 2
The sizeof(intVar)   is 4
The sizeof(longVar)  is 8

Assigning 32767 to shortVar, intVar, longVar

The sizeof(shortVar) is 2
The sizeof(intVar)   is 4
The sizeof(longVar)  is 8

The sizeof(intVar+3/2+3*7-4)   is 4
```

Overflow

When an arithmetic operation attempts to create a numeric value that is outside of the range that can represented with a given number of bits, we get

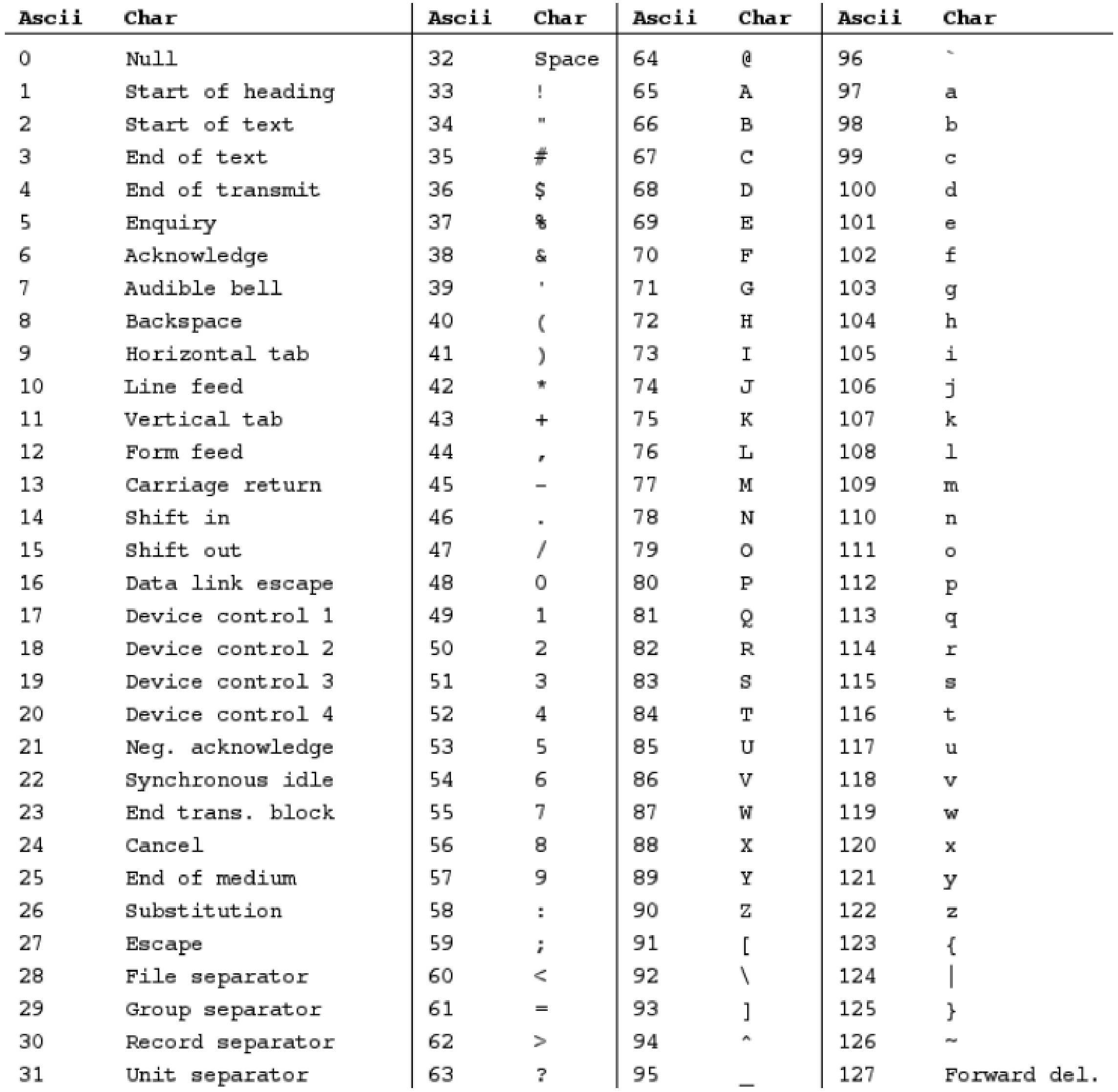
overflow

Each type has its own range



Character Variables

- ASCII character set
  - 128 characters
  - each character has an integer value between 0 and 127
  - C provides an integer type named `char` to represent characters
  - `char` is stored in one byte of memory



Ascii	Char	Ascii	Char	Ascii	Char	Ascii	Char
0	Null	32	Space	64	@	96	`
1	Start of heading	33	!	65	A	97	a
2	Start of text	34	"	66	B	98	b
3	End of text	35	#	67	C	99	c
4	End of transmit	36	$	68	D	100	d
5	Enquiry	37	%	69	E	101	e
6	Acknowledge	38	&	70	F	102	f
7	Audible bell	39	'	71	G	103	g
8	Backspace	40	(	72	H	104	h
9	Horizontal tab	41	)	73	I	105	i
10	Line feed	42	*	74	J	106	j
11	Vertical tab	43	+	75	K	107	k
12	Form feed	44	,	76	L	108	l
13	Carriage return	45	-	77	M	109	m
14	Shift in	46	.	78	N	110	n
15	Shift out	47	/	79	O	111	o
16	Data link escape	48	0	80	P	112	p
17	Device control 1	49	1	81	Q	113	q
18	Device control 2	50	2	82	R	114	r
19	Device control 3	51	3	83	S	115	s
20	Device control 4	52	4	84	T	116	t
21	Neg. acknowledge	53	5	85	U	117	u
22	Synchronous idle	54	6	86	V	118	v
23	End trans. block	55	7	87	W	119	w
24	Cancel	56	8	88	X	120	x
25	End of medium	57	9	89	Y	121	y
26	Substitution	58	:	90	Z	122	z
27	Escape	59	;	91	[	123	{
28	File separator	60	<	92	\	124	|
29	Group separator	61	=	93	]	125	}
30	Record separator	62	>	94	^	126	~
31	Unit separator	63	?	95	_	127	Forward del.

char1Demo.c

Character Variables

```c
#include <stdio.h>

int main(void)
{
    int i;

    for (i = 33; i <= 126; i++)
    {
        printf("%d\t\tis character %c\n", i, i);
    }

    return 0;
}
```

```
33              is character !
34              is character "
35              is character #
36              is character $
37              is character %
38              is character &
39              is character '
.
.
.
120             is character x
121             is character y
122             is character z
123             is character {
124             is character |
125             is character }
126             is character ~
```

char2Demo.c

Character Variables

```
char a = 'a';
char b = 'b';
char c = 'c';
char em = '!';

printf("%d %d %d %d\n\n", a, b, c, em);
printf("a + b + c = %d\n\n", a+b+c);
printf("! + ! = %d\n\n", em + em);
printf("! + ! = %c\n\n", em + em);
printf("%c %c %c %c\n\n", a, b, c, em);
```

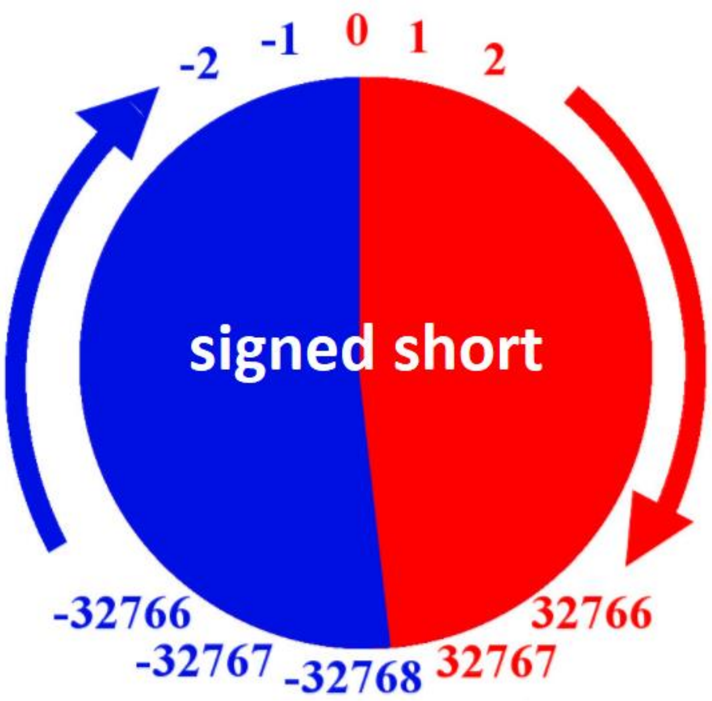
```
97 98 99 33

a + b + c = 294

! + ! = 66

! + ! = B

a b c !
```

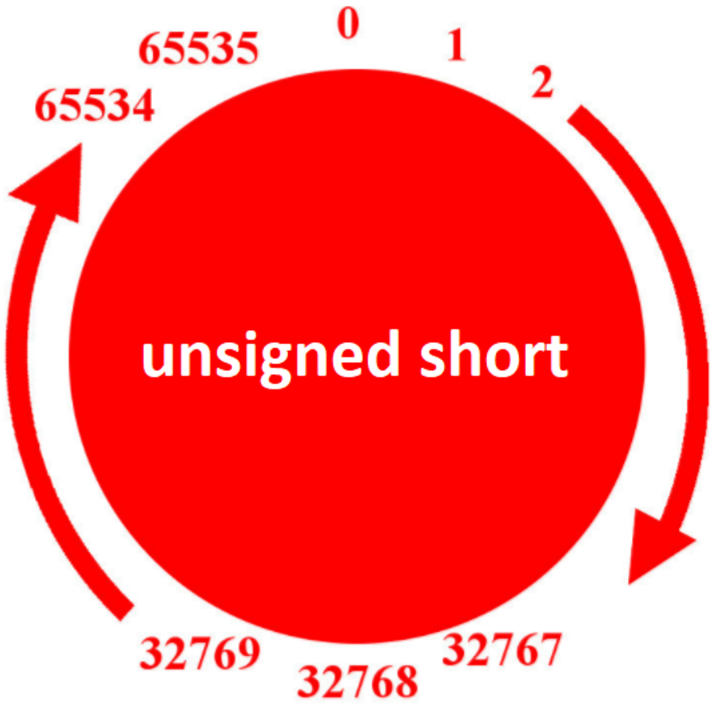
uoverflowDemo.c

Unsigned Types

```
short
int
long
```

```
unsigned short
unsigned int
unsigned long
```





```
The sizeof(short)     is 2           The sizeof(unsigned short) is 2
The sizeof(int)       is 4           The sizeof(unsigned int)   is 4
The sizeof(long)      is 8           The sizeof(unsigned long)  is 8

The sizeof(shortVar)  is 2           The sizeof(ushortVar) is 2
The sizeof(intVar)    is 4           The sizeof(uintVar)   is 4
The sizeof(longVar)   is 8           The sizeof(ulongVar)  is 8

Assigning 32767 to                   Assigning 65535 to
shortVar, intVar, longVar            ushortVar, uintVar, ulongVar

The sizeof(shortVar)  is 2           The sizeof(ushortVar) is 2
The sizeof(intVar)    is 4           The sizeof(uintVar)   is 4
The sizeof(longVar)   is 8           The sizeof(ulongVar)  is 8
```

usizeofDemo.c

Unsigned Types

Conversion Specifications for `unsigned`

`%hu`	an `unsigned short` in decimal
`%u`	an `unsigned int` in decimal
`%lu`	an `unsigned long` in decimal

The `%x` (hexadecimal) and the `%o` (octal) conversion specifications indicate unsigned conversion.

ANSI C and Integer Types

`limits.h`

```
/usr/include/limits.h
```

Contains defines that set the sizes of integer types

```
/* Minimum and maximum values a 'signed int' can hold.  */
#  define INT_MIN       (-INT_MAX - 1)
#  define INT_MAX       2147483647

/* Maximum value an 'unsigned int' can hold.  (Minimum is 0.)  */
#  define UINT_MAX      4294967295U
```

`printf()` – field width specifier

```
printf(control_string, args, ...)
```

```
% [flag] [field width] [.precision] [size] conversion
```

field width

- optional
- a decimal integer constant specifying the minimal field width
- output will be right justified and blanks will be used to pad on the left
- will use more space than designated if more space is necessary to output expression

printfwDemo.c

```
int addend1;
int addend2;
int a;

printf("Enter first  addend ");
scanf("%d", &addend1);
printf("\nEnter second addend ");
scanf("%d", &addend2);

printf("\n\t%5d\n", addend1);
printf("\t\b+%5d\n\t", addend2);


for (a = 0; a < 5; a++)
{
   printf("=");
}

printf("\n\t%5d\n", addend1 + addend2);
```

```
Enter first  addend 12

Enter second addend 1234

           12     12
       + 1234    +1234
        =====     =====
         1246     1246
```

```
Enter first  addend 12345

Enter second addend 0

        12345      12345
       +    0     +0
        =====      =====
        12345      12345
```

Floating Point Types

- `float` – single precision
- `double` – double precision
- `long double` – extra precision

```
float floatVar = 3.14;
double doubleVar = 3.14159;
long double longdoubleVar = 3.1415926535897L;
```

`float.h determines the limits of each type`

For more details on floating point, check out this video

https://www.youtube.com/watch?v=PZRI1IfStY0

fsizeofDemo.c

```
float       floatVar;
double      doubleVar;
long double longdoubleVar;

The sizeof(float)          is 4
The sizeof(double)         is 8
The sizeof(long double)    is 16

The sizeof(floatVar)      is 4
The sizeof(doubleVar)     is 8
The sizeof(longdoubleVar) is 16

floatVar      = FLT_MAX;
doubleVar     = DBL_MAX;
longdoubleVar = LDBL_MAX;
```

fsizeofDemo.c

```
Assigning
340282346638528859811704183484516925440.000000
to floatVar


Assigning
17976931348623157081452742373170435679807056752584499659891747680315726078002
85387605895586327668781715404589535143824642343213268894641827684675467035375
16986049910576551282076245490090389328944075868508455133942304583236903222948
16580855933212334827479782620414472316873817718091929988125040402618412485836
8.000000
to doubleVar


Assigning
11897314953572317650212638530309702051690633222946242004403237338917370055229707226164102903365288828535456978074955773144274 43153670288434198125573853743678673593200706973263201915918282961524365529510646791086614311
79063216977883889613478656060039914875343321145491116008867984515486651285234014977303760000912547939396622315138362241783854 27439178381387178058894875405751682263476592355769748051137256490208848552224947913993775850
26011773549180099796226026859508558883608159846900235645132346594476384939859276456284579661772930407806609229102715046085388 08795932778162298682754783076808004015069494230341172895777710033571401055977524212405734700
73862516601108283791196230084692772009651535002084744707924438485459128867230006190851264721119513614675276335195629275979572 50278002980795904193139603021470997035276467445530922022679656280991498232083329641241038509
23918473478612192169721054348428704835340811304257300221642134891734717423480071488075100206439051723424765600472176809648610 79949434157034763206435586242074435044243805661360176088374781653890278095769759772868600714
87028287955567141404632615832623602762896316173978484254486860609948270867968048078702511858930838546584223040908805996294594 58620190376604844679092600222541053077590106576067134720012584640695703025713896098375799892
69545530523685607586831792231136395194688508807718721047052039575874800131431314442549439199401757531693393923668818561891299 31729104252921236835159922322050998001677102784035360140829296398115122877768135706045789343
53545169653956125404884644716978689321167108722908808277835051822885764606221873970285165508372099234948333443522898475123275 37266360662139022812647062340753520717240586650795182173034637826313533937067749019501978416
90441824738063162828586857741432581165364040218402724913393320949219498422442730427019873044536620350262386957804682003601447 29199712309553005720614186697485284685618651483271597448120312194675168637934309618961510733
00655524214851952017628585950910518394725028638716324941676138049963197914418702543027067584951920088379151694015817400467114 77877201459644461175204059453504764721807975761111720846273639279600339670470037613374509553
18415007379641260504792325166135484129188421134082301547330475406707281876350361733290800595189632520707167390454777712968226 52062256514399193768044002923809031124379126147762559646942219813751469670794468703580043925
07659451618379811859392049544036114915310782251072691486979809240946772142727012404377187409216756613634938900451232351668146 08932240069799317601780533819184998193300841098599393876029260139091141452600372028487213241
19554242821018312042161044674046216353369005836646065911562987647455250681450039329414041314954006776029510059622530228230036 31473824681059648442441324864573137437595096416168048024129351876204668135636877532814675538
79887177183651289394719533506188500326760735438867336800207438784965701457609034985757124304510203873049485425670247933932280 91105260415385289948492039910919461299124916332899179980943803378795220931314669461497059396
64152375949285890960489916121944989986384837022486672249148924678410206183364627416969576307632480235587975245253737035433882 96086275342774001633343405508353704850737454481975472222897528108302089868263302028525992308
41680545396879114182976299889645764827652875045628549242651652177507995162596692291149777889623566709566271384820181913483216 87995863652637620978285070099337294396784639879024914514222742527006363942327998483976739987
15441855420156224415492665301451550468548925862027608576183712976335876121538256512963353814166394951655600026415918655485005 70526114319529199188079545223946496276356301785808966922264062353828985358675959906470083856
87123810329591926494846250768992258419305480763620215089022149220528069842018350840586938493815498909445461977893029113576516 77540623227829831403347327660395223160342282471752818181884430488092132193355086987339586127
60736708666523755556758031714901084773200964243187800700087973460329062789435537435644488519071916164551411557619393996907674 15156402826543664026760095087523945507341556135867933066031744720924446513532366647649735400
85196704077110364053815007348689179836404957060618953500508984091382686953509006678332447257871219660441528492484004185093281 19089636341757398971665960007594878006191640948543387585206571165410722609962881501231443779
44008749301944744330784388995701842710004808305012177123560622895076269042856800047718893158089358515593863176652948089031267 74702966254511086154895839508779675546413794489596052797520987481383976257859210575628440175
93493241621483395653501891968113890918437957347032694063428900878058469403524534793980806742732362978871008671758025315613023 56064878709259865288416350972529537091114317204887747405539054009425375424119317944175137064
68964386151771884986701034153254238591108962471088538580868883777725864856414593426212108664758848926003176234596076950884914 9662444156604419552086811989770240.000000
to longdoubleVar
```

fsizeofDemo.c

```
The sizeof(floatVar)      is 4
The sizeof(doubleVar)     is 8
The sizeof(longdoubleVar) is 16
```

The contents of a variable do not change the `sizeof()` that variable.

Floating Point Types

Using operators with floating point types.

arithmetic	+	-	*	/		
relational	==	!=	<	<=	>	>=
logical	!	&&	||			

Expression	Value	Type
`2.5 + 5.7`	`8.2`	`double`
`2.5 <= 3.62`	`1` (true)	`int`
`2.5 == 3.62`	`0` (false)	`int`
`2.5 / 3.62`	`0.6906`	`double`
`2.5 && 3.62`	`1` (true)	`int`
`!2.5`	`0` (false)	`int`
`!0`	`1` (true)	`int`

Input and Output of Floating Point Values

Conversion Specifications for `scanf()`

`%e`	`%f`	`%g`	`float`
`%le`	`%lf`	`%lg`	`double`
`%Le`	`%Lf`	`%Lg`	`long double`

Conversion Specifications for `printf()`

`%e`	`%f`	`%g`	`%E`	`%G`	`float, double`
`%Le`	`%Lf`	`%Lg`	`%LE`	`%LG`	`long double`

For more about scientific notation

https://www.youtube.com/watch?v=Hmw0wJVud0k

perconDemo.c
psconversionDemo.c

```
Enter a float value for %e              12.3456
Value entered using %e   is             1.234560e+01
Value entered using %.2e is             1.23e+01

Enter a float value for %f              12.3456
Value entered using %f   is             12.345600
Value entered using %.3f is             12.346

Enter a double value for %le            12.3456
Value entered using %le is              1.234560e+01
Value entered using %.4le is            1.2346e+01

Enter a float value for %g              12.3456
Value entered using %g is               12.3456
Value entered using %.2g is             12

Enter a double value %lg                12.3456
Value entered using %lg is              12.3456
Value entered using %.3lg is            12.3

Enter a double long value for %Lg       12.3456
Value entered using %Lg is              12.3456
Value entered using %.4LG is            12.35
```

printf() – precision specification

```
printf(control_string, args, ...)
```

```
% [flag] [field width] [.precision] [size] conversion
```

.precision

- optional
- a period followed by a decimal integer specifying the number of digits to be printed in a conversion of a floating point value after the decimal point

```
float f1 = 1;
float f3 = 3;

double d1 = 1;
double d3 = 3;

long double ld1 = 1L;
long double ld3 = 3L;

printf("float version       of 1/3 %.65f\n\n",
        f1/f3);
printf("double version      of 1/3 %.65f\n\n",
        d1/d3);
printf("long double version of 1/3 %.65Lf\n\n",
        ld1/ld3);

printf("sum = %.65Lf\n\n",
        f1/f3 + d1/d3 + ld1/ld3);
```

```
float version       of 1/3
0.3333333432674407958984375
000000000000000000000000000
0000000000000


double version      of 1/3
0.3333333333333333148296162
562473909929394721984863281
2500000000000


long double version of 1/3
0.3333333333333333333423683
514373792036167287733405828
4759521484375


sum =
1.0000000099341073885863759
307390807862248038873076438
9038085937500
```

floatpreDemo.c