C EXPRESSIONS

- Expression in C are formed from operators and variables or constants. We begin with a discussion of variables.
- there are five basic (atomic) variable types in C.

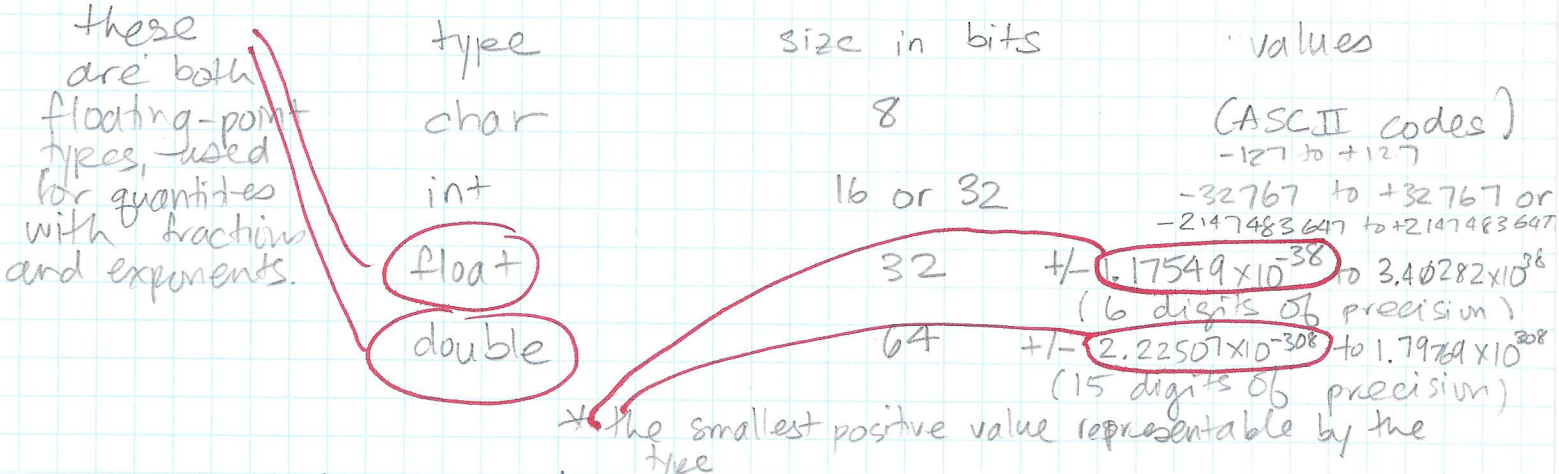
```
char
int
float
double
void   <-- also known as "value less"
```

- there are nine data types in C, but they are based on these types

- recall that types are used to define variables as in:

```
char   ch;
int    num;
float  grade;
double distance;
void   blank;
```

and that a certain number of bytes is taken up by a variable of a given type. The number of bytes depends on compiler/platform you are using. Typically, in a 32-bit environment we have:



- the type void is a generic type indicating that no value is returned or that special objects — called pointers — are created; it has no fixed associated size. To be discussed later...

modifying basic types

- the basic types (except for void) can be modified with special keywords to fit various situations:

signed
unsigned
long
short

→ these can be applied to int, but double only accommodates long

Note that "signed int" is redundant because an integer is already signed

type	size in bits	range
char	8	-127 to 127
unsigned char	8	0 to 256
signed char	8	-127 to 127
int	16 or 32	-32767 to 32767
unsigned int	16 or 32	0 to 65535
short int	16	-32767 to 32767
long int	32	-2147483647 to 2147483647
double	64	
long double	80	

naming variables / functions / labels / etc.

- various user-defined objects are named using identifiers, several characters in length.

Note that:

- the first character must be a letter or underscore
- subsequent letters must be letters, digits, or underscores (other punctuation marks are not allowed)
- the identifier cannot be a C keyword (that is, a reserved word corresponding to a C-language component).

Correct	Incorrect
Sum32	32sum
phidget_7	ph!dget?
_me_myself_I	_me...myself...I

Arrays

- As we have seen, it is possible to store string literals in arrays of characters. Recall our last program contained the definition

```
char user_name[30];
```

meaning that a user name of up to 30 characters may be contained in the variable. This variable is an example of an array. For example the name "Azealia" would be stored as

byte 0	user_name[0]	A
byte 1	user_name[1]	z
byte 2	user_name[2]	e
⋮	user_name[3]	a
	user_name[4]	l
	user_name[5]	i
	user_name[6]	a
	user_name[7]	\0

← note that the first element in an array has index 0

recall that a char variable occupies 8 bits (corresponding to ASCII code)

← most string literals are terminated automatically with a null "\0" character

- Note that we can create arrays of any of the atomic data types or their modified versions, e.g.,

```
short int smallish[11]
long double bignums[23]
unsigned char ucharr[512]
```

- if the array name appears without [·], a pointer to the array is implied. This pointer points to the first element of the array.

- a pointer is a variable that contains a memory address, and we will have much more to say about pointers in the future

Operators

C has a rich set of operators. An operator can be used to modify the values of variables. There are four classes of operator:

- arithmetic
- relational
- logical
- bitwise

An expression in C is composed of one or more operands (variables, constants) that are combined by operators. For example:

```
int x, y, z;
x = 10;   <- assignment operator
y = 20;   <- assignment operator
z = x * y + 23   <- operators: assignment, *, +
```

operands: x, y, 20, 23

this code fragment consists of three expressions. The operands 10, 20 and 23 are integer constants. The operands x, y and z are variables (integer type). The first expression:

```
x = 10;
```

is an assignment. The variable x is set equal to 10.

Operators can be binary (having two operands) or unary (having a single operand)

Arithmetic		
Operator	Function	Use
+	unary signch.	+expr
-	unary signch.	-expr
*	multiplication	expr * expr
/	division	expr / expr
%	remainder	expr % expr
+	addition	expr + expr
-	subtraction	expr - expr

```
x = -17;
y = +3;
z = -x % 3 * z
```

- The following code is valid in C:

```
int a, b, c;
a = b = c = 10;   /* expression with multiple assignments
```

- the assignment operator is "right to left" associative, meaning that the expression can be re-written as

```
a = (b = (c = 10));
```

① the variable c is first set to 10; the assignment operator then returns its left operand, c

② next, b is set equal to c, and b is returned

③ finally, a is set equal to b, a is returned but there is nothing to receive this returned value.

More operators:

Arithmetic		
Operator	Function	Use
++	unary increment	expr++, ++expr
--	unary decrement	expr--, --expr

prefix example:

```
x = 10;
y = ++x;  /* y is set to 11 */
```

postfix example:

```
x = 10
y = x++;  /* y is set to 10 */
```

- in either case, x is set to 11.

Relational	
Operator	Meaning
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal
==	equal
!=	not equal

NOTE: In C, true is interpreted as any value other than ~~zero~~ zero, and false is equal to zero. Expressions that use relational or logical operators return 0 for false and 1 for true.

even more operators

Arithmetic

Operator	Function	Use	Equivalence
+=	binary sum, assignment	expr1 += expr2	expr1 = expr1 + expr2
-=	binary difference, assignment	expr1 -= expr2	expr1 = expr1 - expr2

Logical

Operator	Meaning
&&	AND
||	OR
!	NOT

- what does the expression ((1>0) && (3>2)) || (6!=5) return?
- how about:

```
!1<=0 && 0 || 0
```

NOTE THE PRECEDENCE OF RELATIONAL AND LOGICAL OPERATORS:

HIGHEST (evaluated first)

↓

LOWEST (evaluated last)

```
!
> >= < <=
== !=
&&
||
```

evaluate the following

```
 1010 1111
^0101 1101
```

```
8<<1
1<<3
16>>1
-5 (in binary)
```

BITWISE - USEFUL IN EMBEDDED DRIVERS!

(often used with operands of unsigned int type)

Operator	Meaning
&	and
|	or
^	xor
~	one's complement (not)
>>	shift right
<<	shift left

C STATEMENTS

- A statement in C is a part of the program that is executable — it corresponds to a set of assembly language instructions

Statements can fall into the following categories:

- selection
- iteration
- jump
- label
- expression
- block

For example, selection can be performed by a conditional statement, such as if-else, which we have seen.

An expression (as discussed above) can be a statement in standalone form when followed by a semi-colon

The "return 0;" in main() is an example of a jump statement

- One category we have not yet discussed is iteration, which can be of three types:
  - while
  - for
  - do-while

- What do you suppose the following code does?

```
char d;
int x;
x = 7;
d = '*';
while (x != 0)
{
    printf("%c", d);
    --x;
}
```

character constants are enclosed by single quotes

For understanding while loops, it is sometimes helpful to identify an invariant description for it; in this case, the invariant is "x is the number of characters to be printed"

Q: how many times does this loop execute?

- in debugging code — that is, finding the cause of functional errors in our programs — it is useful to be able to tabulate the progress of algorithms

iteration (pass before })	value of x (before --x;)	displayed result
1	7	*
2	6	* *
3	5	* * *
4	4	* * * *
5	3	* * * * *
6	2	* * * * * *
7	1	* * * * * * *
8 → exit occurs because x = 0		

- another variation is the do-while loop: can you chart the progress of the following code?

```
char d;
int x;
x = 7;
d = '*';
do
{
    printf("%c", d);
    --x;
} while (x != 0)
```

How many asterisks are printed now?

- Here is the for-loop version:

```
char d;
d = '*';
for (int x = 7; x != 0; --x)
{
    printf("%c", d);
}
```

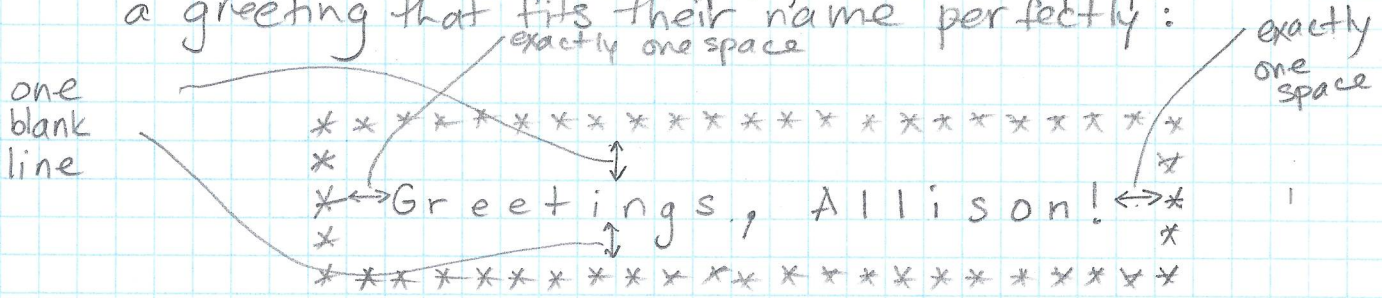
→ after each iteration x is altered based on this expression and also this condition

HOMEWORK

① Look up the strlen() function found in the string.h library. What is its purpose?

Look up the related functions, strcpy() and strcat(). How are they used?

② Using the tools you have learned so far, write a program that takes a user's name from standard input, and displays a greeting that fits their name perfectly:



③ Write a C-program that calculates one's gain on a principal amount of $1000, given an annual compounded interest rate of 6%, over 30 years. Use an approach based on iteration.

④ Can you modify the above program to accept the principal amount, interest rate and investment period, from the user?