Expressions and Operators in C

ESC101: Fundamentals of Computing

Nisheeth

Announcements

- Section number confusion?
  - It seems Pingala shows a changed section number for some students
  - Continue with same section number that you are using right now. We will reconcile with pingala this week
- Week-2 lab graded
  - Apply for regrading only if
    - Your output is almost exactly what is expected in the test cases, but for some reason the test cases are not passing
    - You made a small mistake, fixing which would make your code work. Specify the small mistake in the regrading request, TAs are not obliged to look for it
- Minor Quiz 1 will be graded soon (this week)

Arithmetic on char data type

Note: When printing a char using printf, the quote symbols ‘ ‘ are not shown

- Each char is associated with an integer value (its ASCII
  - Example: char ‘A’ to ’Z’ are associated with integers 65 to 90
  - Refer to the ASCII table for the code (int) of each char (no need to remember by heart). signed char range: -128 to 127, unsigned char range is 0 to 25

Note: When giving char input for scanf, we don’t type the quote symbols ‘ ‘

```
#include <stdio.h>
int main(){
    int x = ‘B’ - ‘A’ + 2;
    printf(“x = %d\n”, a);
    char y = 68;
    printf(“y = %c”,y);
    printf(“y = %d”,y);
    return 0;
}
```

3

D

68

```
#include <stdio.h>
int main(){
    char x = 128;
    printf(“x = %d\n”, x);
    char y = -130;
    printf(“y = %d\n”,y);
    return 0;
}
```

First number from the negative side

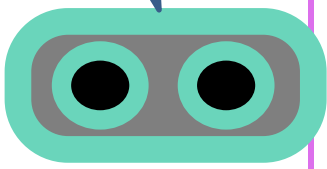
-128

Second number from the positive side

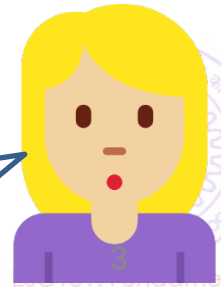
126

128 and -130 are out of the range of signed char

Try in Prutor and see yourself



What if x and y are unsigned char ?



Expressions in C

We use math formulae all the time

```
a = b / 5
x = y * y + z * z
x = (int)(pow((double)y, 2.0) + pow((double)z, 2.0))
```

pow is a function in math.h (power function)
$x = y*y + z*z (= y^2 + z^2)$

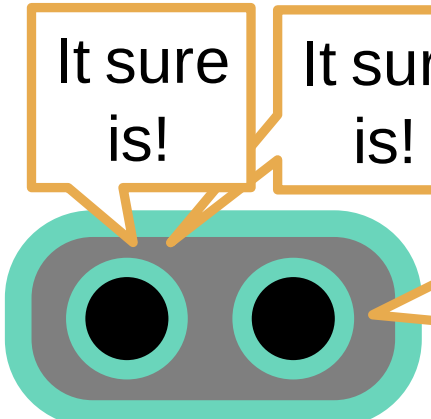
Mr C calls these formulae *expressions*

x = y * y + z * z is an expression for Mr C
y * y + z * z is also an expression for Mr C
y * y is also an expression for Mr C
z * z is also an expression for Mr C

Oh! So two expressions can be added together to get another expression!

So is z an expression?

So is 5 an expression?



It sure is!

It sure is!

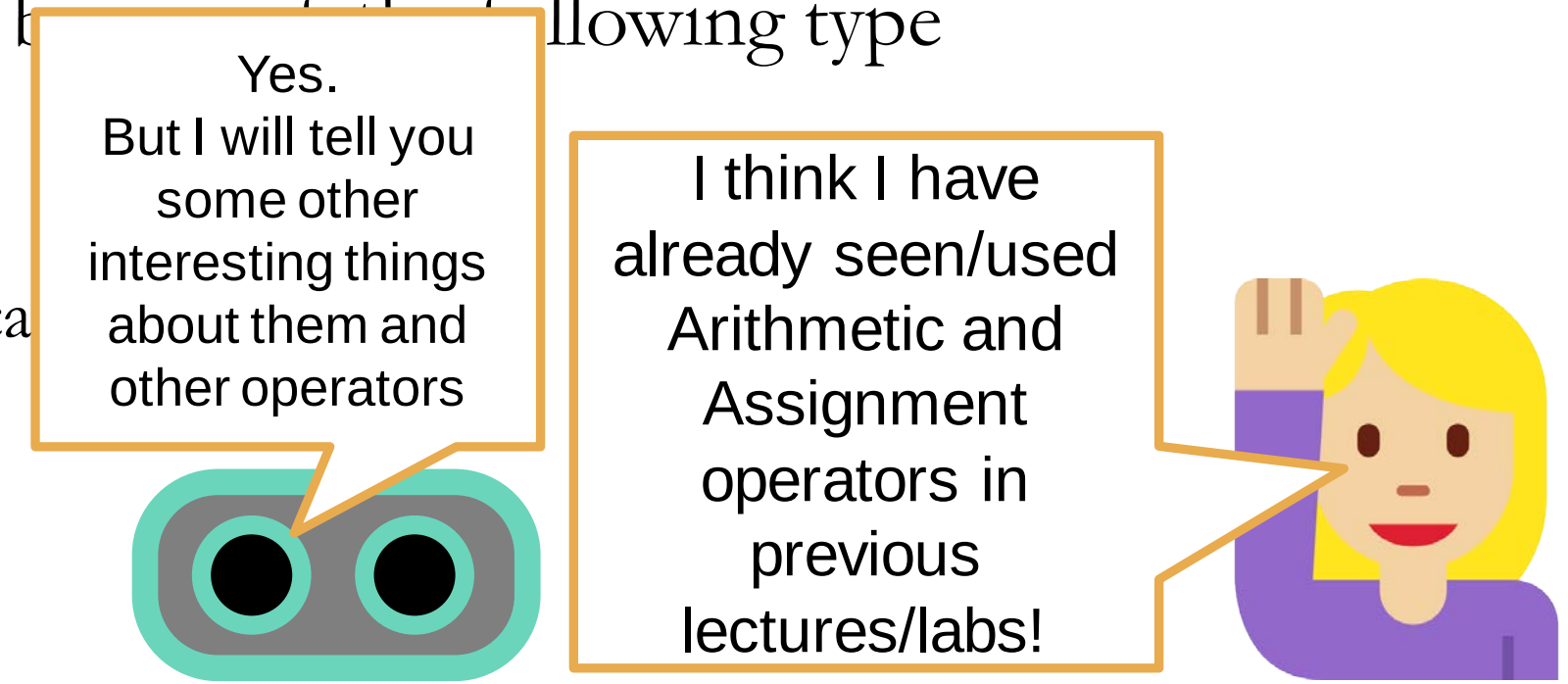
Yes, take two expressions and do operations like addition, multiplication, or assignment (=) with them and a new expression will emerge

Expressions and Operators

- Expressions in C consist of one or more variables/constants
- An expression contains one or more operators, such as

```
c = a + b - 2;
```

- Operators in C can b llowing type
  - Arithmetic
  - Unary
  - Relational and logica
  - Assignment
  - Conditional



Arithmetic operators

- Already seen. Operate on int, float, double (and char)

Op	Meaning	Example	Remarks
+	Addition	9+2 is 11	
		9.1+2.0 is 11.1	
-	Subtraction	9-2 is 7	
		9.1-2.0 is 7.1	
*	Multiplication	9*2 is 18	
		9.1*2.0 is 18.2	
/	Division	9/2 is 4	Integer division
		9.1/2.0 is 4.55	Real division
%	Remainder	9%2 is 1	Only for int

Unary operators

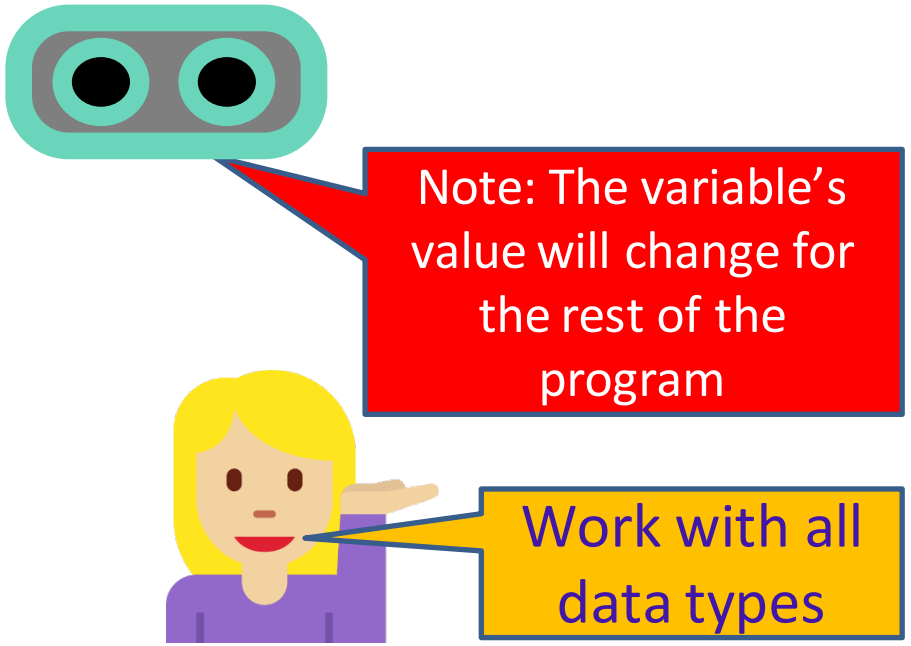
Operator	
-	Negative of an expression
++/--	Increment/decrement a variable
sizeof	Output memory box size for a variable
type (examples: int, float, double, etc)	Type-casting

Unary Operators - Negative

- Operators that take only one argument (or operand)
  - -5
  - -b
- Observe that – is both an arithmetic and unary operator
  - Meaning depends on context
  - This is called overloading

Unary operators – increment and decrement

- Increment (++) increases a variable by 1
- Decrement (--) decreases a variable by 1
- ++*variable* is the pre-increment operator
  - Means increment, then use
- *variable*++ is the post-increment operator
  - Means use, then increment
- Likewise, the -- can be pre/post decrement




```
int main(){
        char a = 'A';     float b = 3.31;
        printf("%c\t%f\n",++a,b++);
        printf("%c\t%f",--a,b--);
        return 0;
}
```

```
B    3.31
A    4.31
```

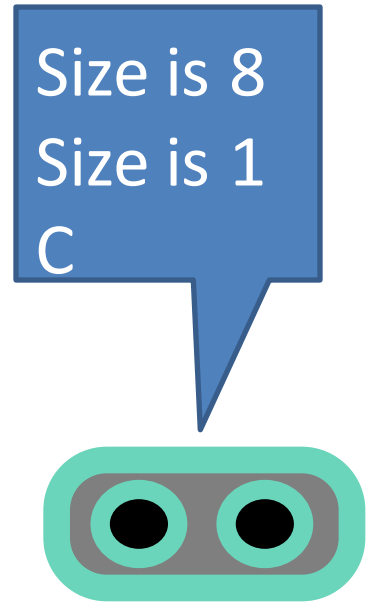
Unary operators - sizeof

- Syntax
  - sizeof var
  - sizeof(type)
- Returns size of the operand in bytes
  - sizeof(char) will return 1
  - sizeof(float) will (mostly) return 4
- Very useful when you are porting programs across computers

Unary operators - typecast

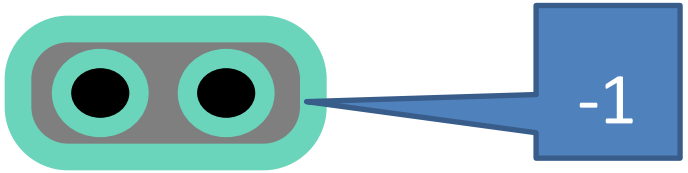
- Syntax
  - (type) var, for example – (int) a, (float) a, etc
- We have already seen this
- What will be the output of this program?

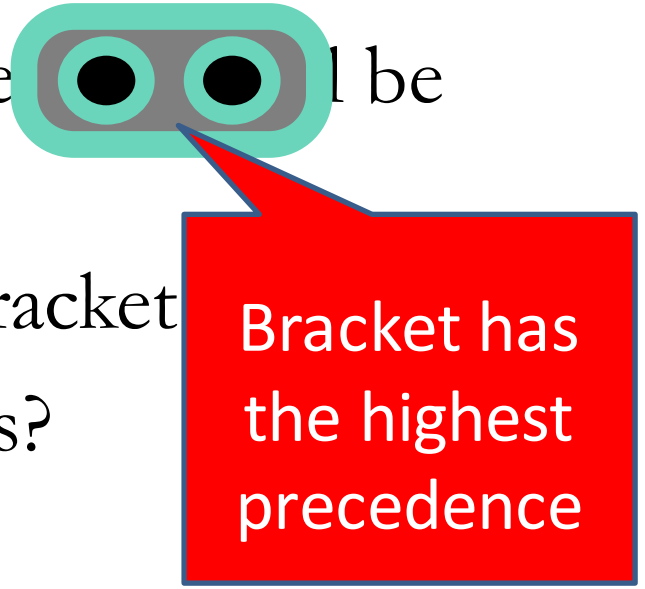
```
int main(){
        double a = 67.2;
        printf("size is %d\n", sizeof a);
        printf("size is %d\n", sizeof((char) a));
        printf("%c", (char) a);
    return 0;
}
```



Precedence Rules for Unary Operators

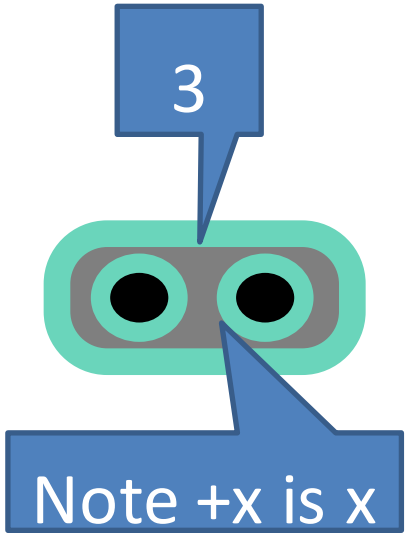
- Precedence rules tell us the order in which the ope l be applied in any C expression
- Unary ops are above arithmetic ops, only below bracket
- If a is 1 and b is 2, what will a + -b be evaluated as?





- What about this program?

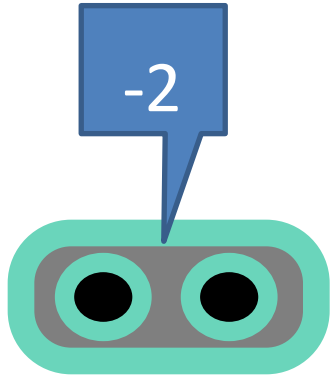
```
int main(){
        int a = 1;    int b = 2;
        printf("%d", a + - + - b);
        return 0;
}
```



Associativity Rules for Unary Operators

- Associativity rules tell us how the operators of same precedence are grouped (e.g., a+b+c will be evaluated as (a+b)+c, not a+(b+c))
- For unary operators, the associativity is from right to left
  - Important to remember this
  - Most other operators' associativity is left to right (e.g., + operator)
- What will this program print?

```
int main(){
        int a = 1;
        printf("%d", - ++a);
       return 0;
}
```

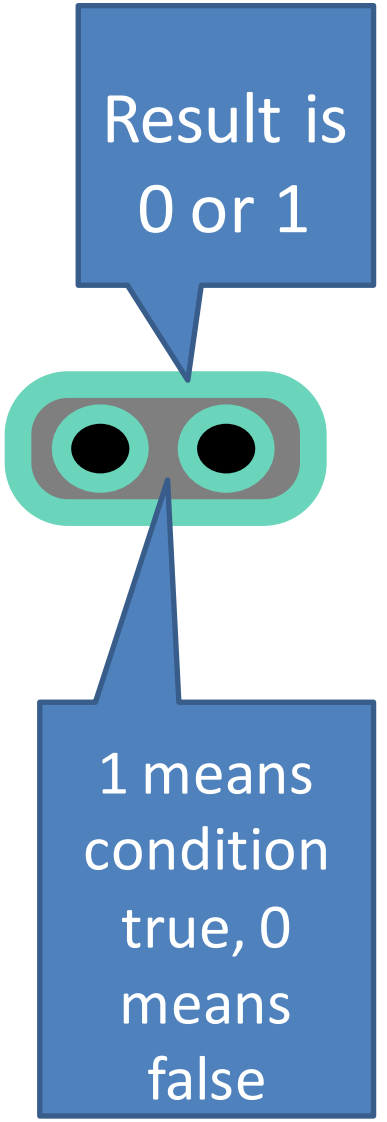


Relational Operators

- Compare two quantities

Operator	Function
>	Strictly greater than
>=	Greater than or equal to
<	Strictly less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

- Work on int, char, float, double…



Relational Operators: Some Examples

Rel. Expr.	Result	Remark
3>2	1	
3>3	0	
‘z’ > ‘a’	1	ASCII values used for char
2 == 3	0	
‘A’ <= 65	1	'A' has ASCII value 65
‘A’ == ‘a’	0	Different ASCII values
(‘a’ – 32) == ‘A’	1	
5 != 10	1	
1.0 == 1	AVOID	May give unexpected result due to approximation

Avoid mixing int and float values while comparing.
Comparison with floats is not exact!

Relational Operators: Another Example

- Problem: Input 3 positive integers. Print the count of inputs that are even and odd.
  - Do not use if-then-else

INPUT
10
5
3

OUTPUT
Even=1
Odd=2

```
int a; int b; int c;
int cEven; // count of even inputs
scanf("%d%d%d", &a,&b,&c); // input a,b,c

// (x%2 == 0) evaluates to 1 if x is Even,
//                          0 if x is Odd
cEven = (a%2 == 0) + (b%2 == 0) + (c%2 == 0);
printf("Even=%d\nOdd=%d", cEven, 3-cEven);
```

Assignment Operator

- Basic assignment (*variable = expression*)

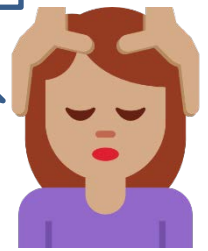
Variant	Meaning
Var += a	Var = Var + a
Var -= a	Var = Var – a
Var *=a	Var = Var *a
Var /=a	Var = Var/a
Var %=a	Var = Var%a

Precedence of Assign Operators

- Always the last to be evaluated
  - x *= -2 *(y+z)/3
  - x = x*(-2*(y+z)/3)
- Seldom need to worry about it

Operator Precedence

Earlier the ASCII table. Now this table? Have to memorize this??

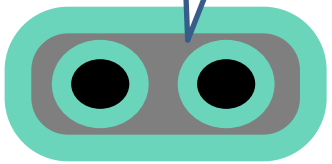


HIGH

LOW

Operators	Description	Associativity
(unary) + -	Unary plus/minus	Right to left
* / %	Multiply, divide, remainder	Left to right
+ -	Add, subtract	Left to right
< > >= <=	less, greater comparison	Left to right
== !=	Equal, not equal	Left to right
=	Assignment	Right to left

No. Write it in your notebook



Example: a + b – c * d % e / f

(a+b) - (((c *d) % e) / f)

Logical Operators

Logical Op	Function	Allowed Types
&&	Logical AND	char, int, float, double
||	Logical OR	char, int, float, double
!	Logical NOT	char, int, float, double

- Remember
- value 0 represents false.
- any other value represents true.
  Compiler returns 1 by default

Logical Operators: Truth Table

"E" for expression

E1	E2	E1 && E2	E1 || E2
0	0	0	0
0	Non-0	0	1
Non-0	0	0	1
Non-0	Non-0	1	1

E	!E
0	1
Non-0	0

Logical Operators: Some Examples

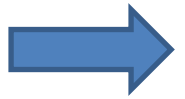
Expr	Result	Remark
2 && 3	1	
2 || 0	1	
‘A’ && ‘0’	1	ASCII value of ‘0’≠0
‘A’ && 0	0	
‘A’ && ‘b’	1	
! 0.0	1	0.0 == 0 is **guaranteed**
! 10.05	0	Any real ≠ 0.0
(2<5) && (6>5)	1	Compound expr

Logical Operators: Precedence and Associativity

- NOT has same precedence as equality operator
- AND and OR are lower than relational operators
- OR has lower precedence than AND
- Associativity goes left to right

2 == 2 && 3 == 1 || 1==1 || 5==4

1 && 0 || 1 || 0

0 || 1 || 0  1 || 0  1

Operator Precedence for various operators

Note: Precedence of brackets () are above every other operator

HIGH

INCREASING

LOW

Operators	Description	Associativity
unary + unary -	Unary plus/minus	Right to left
* / %	Multiply, divide, remainder	Left to right
+ -	Add, subtract	Left to right
< > >= <=	Relational operators	Left to right
== !=	Equal, not equal	Left to right
&&	And	Left to right
||	Or	Left to right
=	Assignment	Right to left

Note: This list doesn't include some other operators that we have not yet seen