

Operators (Continued), Programs with Branching Structure

ESC101: Fundamentals of Computing

Nisheeth

Recap: Opera

Order of evaluation if several operators are present in an expression

Order of evaluation if there are several operators of equal precedence level

- Looked at various operators in C, their **precedence** and **associativity**

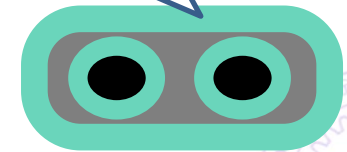
Note: Precedence of brackets () is above every other operator

Precedence ↑

Operators	Description	Associativity
unary + -, ++, --, type, sizeof	Unary plus/minus	Right to left
* / %	Arithmetic: Multiply, divide, remainder	Left to right
+ -	Arithmetic: Add, subtract	Left to right
< > >= <=	Relational operators	Left to right
== !=	Relational operators	Left to right
&&	AND	Left to right
	OR	Left to right
=	Assignment	Right to left

LOW

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



Also see: https://en.cppreference.com/w/c/language/operator_precedence

Plan for today

- **Logical Operators** (started but wasn't finished last time)
- The **Conditional Operator** (didn't see last time)
- Start discussing **conditional statements** (if, if-else, etc) to write C programs that have a **branching structure** and help us **make choices** in our programs

Logical Operators

- There are 3 logical operators in C: AND (&&), OR (||), NOT (!)

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

- Operands can be variables/constants (or expressions in general)
 - Expression-1 && Expression-2 (result = 1 only when **both** expr. are non-zero)
 - Expression-1 || Expression-2 (result = 1 if **at least one** of them is non-zero)
 - !Expression (negates the result of an expression: 0 to 1 or non-zero to 0)

Logical Operators: Some Examples

	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	AND operating on 2 expressions

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: Precedence and Associativity

- NOT has same precedence as unary operators (thus **very high precedence**)
- AND and OR have **lower precedence than relational operators**
- OR has lower precedence than AND (**important**)
- Associativity for logical operators is **left to right**

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



$1 \ \&\& \ 0 \ || \ 1 \ || \ 0$



$0 \ || \ 1 \ || \ 0 \ \longrightarrow \ 1 \ || \ 0 \ \longrightarrow \ 1$

The Conditional Operator

A question being asked here

- The conditional operator is of the form

Expression 1 ? Expression 2 : Expression 3

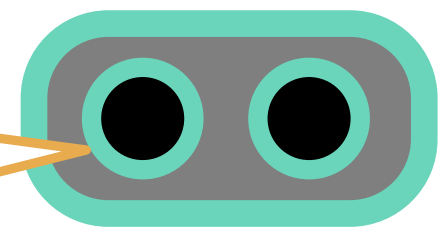
- Meaning: Evaluate expression 1, if it is true (non-zero), evaluate expression 2, otherwise evaluate expression 3
- The operator **generates the value** of expression 2 or expression 3
- Often, we assign the result to another variable ($a = \text{exp1} ? \text{exp2} : \text{exp3}$)
 - Data type of generated value ? Whichever of exp2 or exp3 is of higher type
- Precedence of cond. operator is just above assignment operators
- Associativity of cond. operator is right to left

The Conditional Operator: Some Examples

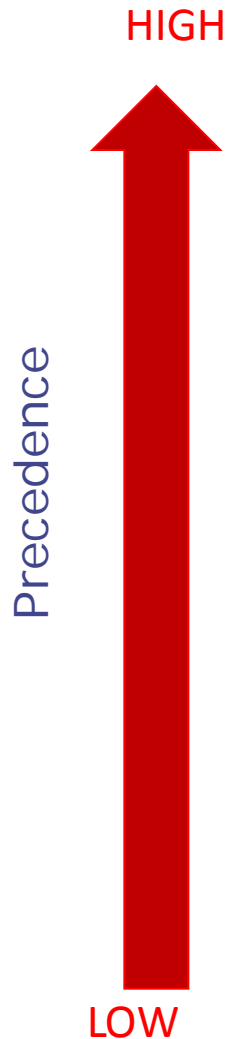
- `a = (i>0) ? 100 : 10;` `/* a will be 100 or 10 depending on i */`
- `a = (i>0)? 10.0 : 5;` `/* RHS result will be a float */`
- A sophisticated example (expression 1 consisting of multiple operators)
 - `c += (a>0 && a<=10) ? ++a : a/b;`
 - The above will first evaluate `a>0 && a<=10` and then choose `++a` or `a/b`
 - Result from RHS will be added to `c` (`c = c + result`)

Now our table is..

Whenever unsure, use brackets to ensure the expression does what YOU want



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



Operators	Description	Associativity
(unary) + -, !	Unary plus/minus, logical NOT	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
&&	Logical And	Left to right
	Logical Or	Left to right
? :	Conditional	Right to left
=	Assignment	Right to left

Note: Ensure Your Expressions Say What You Mean

```
0 <= 10 <= 4
```

```
(0 <= 10) <= 4
```

```
1 <= 4
```

```
1 /* True */
```

```
0 <= 10 && 10 <= 4
```

```
(0 <= 10) && (10 <= 4)
```


```
(1) && (10 <= 4)
```

```
1 && (0)
```

```
0 /*False*/
```

Some Useful Tips on using correct Data Types

- Double and float are both happy with %f for printf
- However, in scanf, double insists on %lf (%f gives junk)
- Don't use a float/double for long integers



Why? What would be the problem?

Range of float is larger. What if I store it as a float?

When you say `long a = 3213213210`, since the number is within range of long, I will preserve every digit of it carefully

When you say `float a = 3213213210`, I will store 3213213184.00

The number is like 3.2×10^9 and my error was just 26. Don't blame me!

- Choice between float or double: If you don't want your digits after decimal to be rounded off, use double instead of float

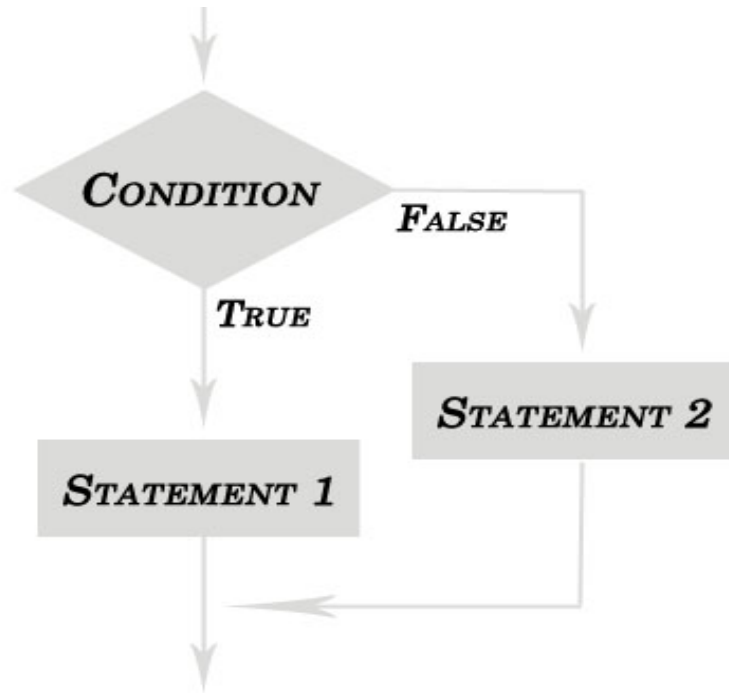
Precision

- There are infinite real numbers between any two real numbers
- We can represent only 2^{32} numbers in 32 bits
- So we can store only a vanishingly small number of decimal numbers precisely
- All others are approximated to 8 (float) or 16 (double) decimal places

```
0 01111111 000000000000000000000000000000002 = 3f80 000016 = 1 (one)
```

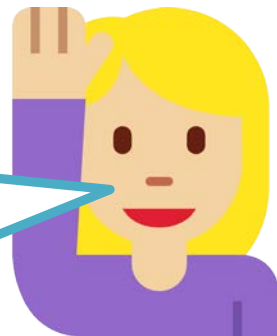
```
0 01111111 000000000000000000000000000000012 = 3f80 000116 = 1 + 2-23 ≈ 1.0000001192  
(smallest number larger than one)
```

Programs with Conditional Statements

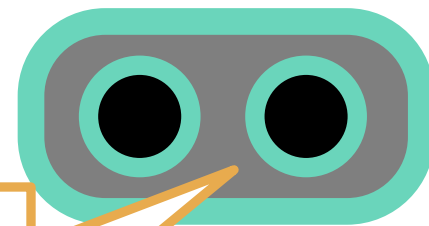


If condition true
do abc
Otherwise
do xyz

But didn't you just teach me about **conditional operators** ?



Yes, but they are usually for small expressions. For more complex programs, I have something different (and better) for you 😊



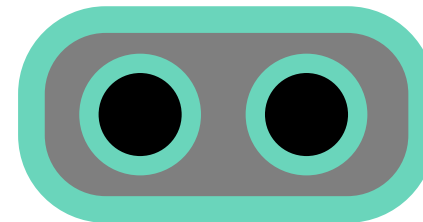
Branching using `if` statement

```
int main(){  
    int salary, loan = 0; // 0 means not approved, 1 means approved (initialize with 0)  
    float interest_rate;  
    scanf("%d",& salary);  
    if (salary >= 400000) {  
        loan = 1; // 1 means loan approved  
        interest_rate = 10.0;  
    }  
    // other stuff in the program..  
}
```

Testing condition is an expression that gives 0 or 1 value

Braces required only when there are multiple statements within the if block

Will execute this block of code only if the condition (`salary > 400000`) is true (1)



Branching using **if-else** statement

```
int main(){
    int salary, loan_amount;
    float interest_rate;
    scanf("%d",& salary);
    if (salary > 400000) {
        loan_amount = 1000000;
        interest_rate = 10.0;
        printf("Congratulations! Your loan amount is %d, interest rate is %d",loan_amount,interest_rate);
    }
    else {
        printf("Sorry! Your loan cannot be approved");
    }
    // do other stuff in the program
}
```

The if block (can have one or more statements)

The else block (can have one or more statements)

Various ways of using **if** and **else**

```
if (condition) {  
}
```

```
if (condition) {  
}  
else {  
}
```

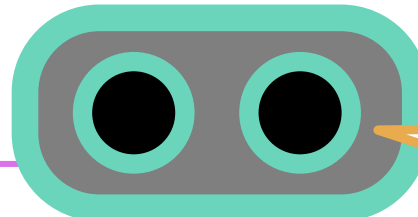
```
if (condition-1) {  
}  
else if (condition-2) {  
}  
else {  
}
```

```
if (condition-1) {  
}  
else if (condition-2) {  
}  
else if (condition-3) {  
}  
⋮  
else if (condition-N) {  
}  
else {  
}
```

```
if (condition-1) {  
  if (condition-2) {  
  }  
  else {  
  }  
}  
else {  
  if (condition-3) {  
  }  
  else {  
  }  
}
```

“nested” if

Note: Each else must have a matching if (also, **number of if** must be **equal to** or **more than number of else**)



Be Careful with Braces when using if-else

- If you do not put curly braces, Mr. C will try to put them for you (and maybe in a way that you don't want him to)

If you write like this....

```
if((a != 0) && (b != 0))
    if(a * b >= 0)
        printf("Positive product");
else
    printf("One number is zero");
```

Mr. C will treat it like this internally

```
if((a != 0) && (b != 0)){
    if(a * b >= 0){
        printf("Positive product");
    }else{
        printf("One number is zero");
    }
}
```

If you do not put brackets, I will match else to closest if

I will not care how you did indentation

But that is not what I meant

