ECS 36A, April 10, 2023

Logical Constants and Operators

- In C, 0 is false and anything non-zero is true
- Operators
  - greater than: *x* > *y*
  - greater than or equal to : *x* >= *y*
  - equal to: *x* == *y*
  - less than: *x* < *y*
  - less than or equal to : *x* <= *y*
  - not equal to: *x* != *y*
- Example: x = 7; y = 19; z = (x >= y); *here z is 0 (false)*
- Example: x = 7; y = 19; z = (x != y); *here z is 1 (true)*

Combination Operators

Logical and: *x* && *y* (1 if both *x*, y are true)
Logical or: *x* || *y* (1 if either(or both *x*, y are true)
Logical not: !*x* (1 if *x* is false, 0 if *x* is true)

x	y	x && y	x || y	!x
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Precedence and Associativity

- ! has highest precedence, associates right to left
- && comes next, associates left to right
- || comes next, associates left to right

- ! comes before the arithmetic operators
- && and || come after

Lazy Evaluation

- C evaluates logical operators left to right
- It stops *as soon as it can determine the result*
- Examples: let x = 12; y = 29; z = -1; then
  - (x < y || y < z && x < z) = 1 [y < z, && is false, then x < y, || is true, stop]
  - (x > y || y > z && x > z) = 1 [y > z, x > z, && is true, so || is true, stop]
  - x > y && y > z = 0 [x > y, && is false, stop]

Conditional Branching: if

if (*condition*){
    statements
}

- Test *condition*
- If true, execute the *statements*
- If false, do not execute the *statements*
- Note: if there is only one *statement*, you can omit the { }

Example

```
x = 12;
if (x == 12)
    printf(“x is 12!”);
if (x < 12)
    printf(“x is less than 12!”);
```

- x is indeed 12, so print “x is 12!”
- x is not less than 12, so the second if prints nothing

Conditional Branching: if/else

if (*condition*){
    if_statements
}
else {
    else_statements
}

- Test *condition*
- If true, execute the *if_statements*
- If false, do not execute the *else_statements*
- Note: if there is only one statement in the if or else, you can omit the { }

Examples

```
x = 12;
if (x == 12)
    printf("x is 12!");
else
    printf("x is not 12!");
```

- x is indeed 12, so print "x is 12!"

```
x = -3;
if (x == 12)
    printf("x is 12!");
else
    printf("x is not 12!");
```

- x is not 12, so print "x is not 12!"

Conditional Branching: Nested ifs

```
if (condition1){
    if1_statements
}
else {
        if (condition2){
            if2_statements
        }
        else {
            else_statements
    }
}
```

- Test *condition1*
- If true, execute the *if1_statements*
- If false, go to else and test *condition2*
- If true, execute the *if2_statements*
- If false, execute the *else_statements*

Conditional Branching: A Cleaner Way

```
if (condition1){
    if1_statements
}
else if (condition2){
    if2_statements
}
else {
    else_statements
    }
}
```

- Test *condition1*
- If true, execute the *if1_statements*
- If false, go to else and test *condition2*
- If true, execute the *if2_statements*
- If false, execute the *else_statements*

Example

```
if (x == 12)
    printf(“x is 12!”);
else if (x == 11)
    printf(“x is 11!”);
else if (x == 10)
    printf(“x is 10!”);
else
    printf(“x is not 10, 11, or 12!”);
```

- If x is 12, prints “x is 12!”
- If x is 11, prints “x is 11!”
- If x is 10, prints “x is 10!”
- If x is 28, prints

  “x is not 10, 11, or 12!”

Conditional Branching: switch Statement

```
switch(expression){
case case1:
        statements1;
        break;
case case2:
        statements2;
        break;
default:
        statementsd;
        break;
}
```

- Evaluate *expression*
- If it evaluates to *case1*, execute *statements1* and leave the switch
- If it evaluates to *case2*, execute *statements2* and leave the switch
- Otherwise, execute *statementsd* and leave the switch
- Each of the *caseis* must be different

Example

```
switch(x){
case 12:
        printf(“x is 12!”);
        break;
case 11:
        printf(“x is 11!”);
        break;
case 10:
        printf(“x is 10!”);
        break;
default:
        printf(“x is not 10, 11, or 12!”);
        break;
}
```

- If x is 12, prints “x is 12!”
- If x is 11, prints “x is 11!”
- If x is 10, prints “x is 10!”
- If x is 28, prints
  “x is not 10, 11, or 12!”

Example, But Omitting break

```
switch(x){
case 12:
        printf("x is 12!");
case 11:
        printf("x is 11!");
        break;
case 10:
        printf("x is 10!");
        break;
default:
        printf("x is not 10, 11, or 12!");
}
```

- If x is 12, prints "x is 12!x is 11"
- If x is 11, prints "x is 11!"
- If x is 10, prints "x is 10!"
- If x is 28, prints
  "x is not 10, 11, or 12!"

Note: leaving off the "break" at the end works, but is *very bad form* (because someone may add a case after it and not notice there is no break in the one above