Statements

Last updated 10/27/20

Statements

- These slides introduce 4 types of statements
- Upon completion: You should be able interpret and code using these statement types

Statements

- Statement
  - Causes the processor to do something
  - 11 types of statements
    - Null
    - Expression
    - Return
    - Compound
    - Conditional
    - Labeled
    - Switch
    - Iterative
    - Break
    - Continue
    - Goto

Statements

- Statement
  - Null Statement
    - Causes nothing to happen

```
;

while(1){
    ;
}
```

Statements

- Statement
  - Expression Statement
    - An expression with a semi-colon added
    - Causes the processor to evaluate the expression
    - Causes the processor to complete any side effects
    - Processor discards the expression

    - Special note: the side effect of the assignment operator is to store a value into a variable

Statements

- Statement
  - Expression Statement

aa = 5;

; causes the expression to be evaluated → 5

side effect of the assignment (=) is aa holds the value 5

aa = bb = 5;

same precedence, operate R to L

bb = 5

value is 5, side effect is bb holds the value 5

aa = 5

value is 5 (<u>value</u> of BB), side effect is aa holds the value 5

note: this equals 5 (the value), not bb

Statements

- Statement
  - Expression Statement

```
ab = 5;
```

value is 5
side effect is ab takes the value 5

```
ab++;
```

value is 5
side effect is ab takes the value 6
the value is then discarded (not assigned to anything)

Statements

- Statement
  - Return Statement
    - Terminates all functions (including main)

```
int main(void) {
…
   return 1;
}
```

Statements

- Statement
  - Compound Statement
    - Block of code containing zero or more statements
    - These statements are considered a single entity
    - Defined by {…}

```
int main(void) {
…                              // multiple statements
   return 1;
}
```

Statements

- Statement

  - Pre-processor commands vs statements

```
#define int_rate 0.25          // pre-processor command

#define int_rate 0.25;         // error

payment = int_rate * balance;
```

creates a compiler error at the “payment =” line

but you never see the expansion

```
payment = 0.25; * balance;
```

very difficult to catch