Statements

Last updated 9/4/21

Statements

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

Statements

- Null Statement
  - Causes nothing to happen

```
;

while(1){
  ;
}
```

Statements

- 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

- Expression Statement

aa = 5;

; causes the expression to be evaluated → 5
side effect of the assignment (=) is aa holds the value 5
the value 5 is then discarded

Statements

- Expression Statement

aa = bb = 5;

same precedence, operate R to L

; causes the expression to be evaluated → 5
side effect is bb holds the value 5
the value 5 is then discarded

aa = bb

value is 5 (value of expression bb)
side effect is aa holds the value 5
the value 5 is then discarded

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

Statements

- Expression Statement

ab = 5;

; causes the expression to be evaluated → 5
side effect is ab takes the value 5
the value 5 is discarded

ab++;

; causes the expression to be evaluated → 5
side effect is ab takes the value 6
the value 5 is then discarded

Statements

- Return Statement
  - Terminates all functions (including main)

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

value can be a variable, expression or a constant

If type of value does not match the return type in the function definition, it will typically be cast to the proper type when returned (some compilers throw an error)

Statements

- Return Statement

Functions that return nothing (void) terminate with just return (no value)

```
void foo(void) {
…
  return;
}
```

Statements

- 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

- 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

You see: payment = INT_RATE * balance;

The compiler uses: payment = 0.25; * balance;

very difficult to catch