Announcements

- Program #1A
  - due Thursday
- Reading
  - Chapter 3 & 7 (skip 7.6)
  - Chapter 10 (Thursday, skip 10.2)

Storage Class and Linkage Example

```
extern int b;
static int c;
int func1(int e)
{
    static int longLived;
    int funcLifetime;

    func2(42);
}
```

Statements

- Assignment is just an expression:
  - Legal statements:
    - X = 3;
    - Y = 5 + (x = 3);
    - Y + 3;
    - x == y;
- If Statement

```
if (expression) {
        statement
} else {
        statement
}
```

  - () are required
  - Good style is to **Always** use { } around statements in an if
  - Expression is of type int
    - any non-zero means the true statement is evaluated

Examples of If Statements

```
if (x > 43) {
    printf("X is bigger than 43\n");
}

if (!x) {
    printf("x is zero");
}

if (x == y) {
    x = 42;
} else {
    x = y;
}
```

```
if (x > 43) {
    printf("X is bigger than 43\n");
} else if (x > 10) {
    printf("X greater than 10\n");
} else {
    printf("X is 10 or less\n");
}
```

While Statement

```
while (expression) {
    body
}
```

- Similar to while in Java
- Expression is (like if) an integer expression
  - Loop terminates when the expression is 0
- Forced loop termination
  - break – exit **one** innermost loop level
  - break does **not** take a label like Java
- Forced next iteration
  - continue – return to loop head and evaluate expression
    - Also applies only to inner most loop

Examples of While Loops

```
while (x < y) {
    ....
}

while (x < y && !error) {
    ...
}

while (!done) {
    while (x < y) {
        ...
    }
}
```

More Examples of While Loop

```
while (!done) {
    ...
    if (a == b) {
        done = 1;
    }
    …
}
```

```
while (1) {
    ...
    if (a == b) {
        break;
    }
    …
}
```

For Statement

- for (expression1; expression2; expression3) …
  - Expression1
    - Run once before first iteration
    - Used to initialize values
  - Expression2
    - Evaluated before each loop iteration
    - Controls loop execution, when it's 0 loop stops
  - Expression3
    - Run after each iteration
  - Can omit any expression
  - Loop body may contain break or continue like while loops

Examples of For Loop

```
for (i=0; i < 10; i++) {
    a[i] = 0;
}

for (;;) {
}

for (i=0, b=21; i < 10; i++) {
    a[i] = b + i;
}
```

Do Statement

```
do {
    statement
} while (expression);
```

- Like a while, but the body always runs
- Rarely used

Switch Statement

```
switch (expression) {
   case constant-expression:
      statement;
      break;
   ....
   default:
       statement;
       break;
}
```

- Execution starts at the constant-expression equal to the expression
  - Without the break, will continue to next statement!!!
- Default constant expression matches anything else

Example of Switch Statement

```
switch (command) {
    case 'A':
            add_entry();
            break;
    case 'D':
            delete_entry();
            break;
    case 'P':
            print_entry();
            break;
    case 'E':
            edit_entry();
            break;
    default:
            printf("Error");
            break;
}
```

Functions

- type name (parameters) block
  - type: return type for the function
  - name: name of the function
  - parameters: a comma separated list of types and names
  - block: the code to run
- return statement: return expression;
  - terminates a function at that point
  - expression is the value returned by the function

Function Example

```
int find(int data[10], int value)
{
        for (i=0; i < 10; i++)  {
             if (data[i] == value)  {
                return i;
             }
        }
}
```

Function Arguments

- Comma separated list of values
- All parameters are passed by value
  - called function can modify values
  - arrays are passed by the address of their 0th element
    - modifying elements of array seen by calling function
- Can pass address of variable to change values

```
int foo(int *a) {
        *a = 3;
}

int x;
foo(&x);
```

Recursive Functions

- Functions that call themselves
  - can be indirect: a calls b calls a
- Example:

```
void binary_to_ascii(unsigned int value)
{
        unsigned int quotient;
        quotient = value / 10;
        if (quotient != 0) {
                binary_to_ascii(quotient);
        }
        printf("%d", value % 10 + '0');
}
```

Abstract Data Types

- Key Idea: Hide implementation from users
  - lets implementation evolve without changing use
  - makes it easier to understand code
    - simply know what a function should do not how

- Static keyword helps with this
  - hides global variables from other modules
  - hides functions that should not be called from other modules