Announcements

- Program #1A
  - due Thursday
- Reading
  - Chapter 1-4 & 7 (skip 7.6)
  - Chapter 10 (by Thursday, skip 10.2)

Storage Class and Linkage Review

```
extern int b;
static int c;
Int func1(int e)
{
    static int longLived = 6;
    int funcLifetime;
}
```

Statements

- Assignment is just an expression:
  - Legal statements: assuming int x, y;
    - x = 3;
    - y = 5 + (x = 3);
    - y + 3;
    - x == y;
    - x = (y < 3);
- If Statement

  If (expression)

  statement

  else

  statement
  - () are required
  - statement could also be a block
    - Good style is to **Always** make it a block
  - Expression should be of type int
    - any non-zero is “true”; zero is “false”
  - Can be floating point types (promoted to int)
    - DON’T USE THIS FACT – FLOATS may not be exact

Examples of If Statements (int x, y)

```
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");
}
```

```
if (x=5)
  printf("true\n");
else
  printf("false\n");
```

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

```
if (x-5) {
    printf("X is not five\n");
}
```

While Statement

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

- Similar to while in Java
- Expression is (like if) an integer expression
  - Loop terminates when the expression evaluates to 0 at the top
- Getting out of loop
  - **`break`**
    - continue with line immediately after the loop
    - do not reevaluate the expression
  - **`continue`**
    - return to loop head and evaluate expression
  - Only the **one** innermost loop level of nesting

Examples of While Loops

```
while (x < y) {
     ….
}

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

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

More Examples of while Loop

```
while (!done) {
    …stmts1...
    if (a == b) {
            done = 1;
    }
    …stmts2…
}
```

```
while (1) {
    …stmts1...
    if (a == b) {
            break;
    }
    …stmts2…
}
```

```
while (a != b) {
    …stmts1...
    …stmts2…
}
```

```
while (a != b) {
    …stmts1...
   if (a == b){
        …stmts2…
   }
}
```

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 terminates
  - Expression3
    - Run immediately 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 at least once

String Length Examples

```
#include <stdio.h>
#define MAX 20
```

```
int slen1(char name[MAX]){
   int size=0, index = 0;
   while (name[index] != '\0'){
      size++;
      index++;
   }
   return size;
}
```

```
int slen2(char name[]){
  int size, index;
  for (size=0, index=0; name[index] != 0;
     size++, index++);
  return size;
}
```

```
int slen3(char name[]){
   int size = 0, index=0;
   do {
     size++;
     index++;
   }while (name[index]);
   return size;
}
```

```
int main(void){
    char myname[MAX] = "Jan Plane";
    printf("by slen1: length =%d\n",
                slen1(myname));
    printf("by slen2: length = %d\n",
                slen2(myname));
    printf("by slen3: length = %d\n",
                slen3(myname));
    return 0;
}
```

Switch Statement

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

- Execution starts at the constant-expression equal to the expression
  - When it hits a break, it will continue to next statement after the end of the whole switch
- Default constant expression matches anything else

Example of Switch Statement

```
switch (expression) {
    case 'A':
            add_entry();
            break;
    case 'D':
    case ‘R’:
            delete_entry();
            break;
    case ‘P':
            print_entry();
    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;
             }
        }
}
```

```
int slen4(char name[]){
   int index=0;
   while (1){
      if (name[index++] == '\0')
         return index - 1;
   }
}
```

Function Arguments

- Comma separated list of values
  - must match in number to the parameters
  - must match in type to the parameters
    - promotion and demotion are allowed
  - don't need to match in name
- All parameters are passed by value
  - called function can modify their copy of values
  - will not modify the copy held by the caller function
- But pointers can be passed by value also
  - this allows you to refer to a space in memory
  - the function can change what is in that space in memory
  - this is not the same as changing the value of the argument

Functions for swapping two integers

```
void swap1(int a, int b){
   int c;
   c = a;
   a = b;
   b = c;
}
```

```
void swap2(int *a, int *b){
   int c;
   c = *a;
   *a = *b;
   *b = c;
}
```

```
void swap3(int *a, int *b){
    int *c;
   c = a;
   a = b;
   b = c;
}
```

```
int main(void){
  int x = 5 , y = 23;
   printf("BEFORE 1: a = %d and b = %d\n",x,y);
   swap1(x,y);
   printf("AFTER 1: a = %d and b = %d\n",x,y);
   x = 5; y = 23;
   printf("\n\nBEFORE 2: a = %d and b = %d\n",x,y);
   swap2(&x,&y);
   printf("AFTER 2: a = %d and b = %d\n",x,y);
   x = 5; y = 23;
   printf("\n\nBEFORE 3: a = %d and b = %d\n",x,y);
   swap3(&x,&y);
   printf("AFTER 3: a = %d and b = %d\n",x,y);
   return 0;
}
```

Passing Arrays

- Array Name is actually a pointer type variable
  - indicates the 0^{th} element in the array
  - arrays are passed by the address of their 0th element
  - modifying elements of array seen by calling function

```
#include <stdio.h>
#define MAX 2
void swap1(int a[MAX]){
   int c;
   c = a[0];
   a[0] = a[1];
   a[1] = c;
}
int main(void){
   int x[MAX] = {5,23};
   swap1(x);
   return 0;
}
```

Recursive Functions

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

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

Another Recursion Example (another way to do string length)

```
int slen5(char name[MAX]){
   static int index = 0;
   if (name[index] != '\0'){
     index++;
     return slen5(name) +1;
   }
   else{
       return 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