Midterm Review

CS 2060 Week 8

Prof. Jonathan Ventura

Modulo operator

- % is the *modulo* or remainder operator.
- `a % b` gives the remainder of `a / b` – the integer portion left over after division.
- `a` is equal to `b * (a/b) + (a%b)`:

```
10 / 3 == 3
10 % 3 == 1
10 = 3 * 3 + 1
```

Modulo operator

With modulo we can get a repeating sequence of numbers:

```
for ( int i = 0; i < 10; i++ ) printf("%d ",i%4);
```

Output:

```
0 1 2 3 0 1 2 3 0 1
```

Order of Arithmetic Operations

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first, inner to outer.
*	Multiplication	Evaluated second, left to right.
/	Division	
%	Modulo	
+	Addition	Evaluated third, left to right.
-	Subtraction	
=	Assignment	Evaluated last.

Table: Precedence of arithmetic operators in C.

Arithmetic puzzles

```
1 + 2 + 3 + 4 + 5 / 3
```

Arithmetic puzzles

```
  1 + 2 + 3 + 4 + 5 / 3
= 11
```

Arithmetic puzzles

```
 1 + 2 + 3 + 4 + 5 / 3
= 11
 (1 + 2 + 3 + 4 + 5) / 3
```

Arithmetic puzzles

```
 1 + 2 + 3 + 4 + 5 / 3
= 11
 (1 + 2 + 3 + 4 + 5) / 3
= 5
```

Arithmetic puzzles

```
 1 + 2 + 3 + 4 + 5 / 3
= 11
 (1 + 2 + 3 + 4 + 5) / 3
= 5
 10*10/5
```

Arithmetic puzzles

```
 1 + 2 + 3 + 4 + 5 / 3
= 11
 (1 + 2 + 3 + 4 + 5) / 3
= 5
 10*10/5
= 20
```

Arithmetic puzzles

```
 1 + 2 + 3 + 4 + 5 / 3
= 11
 (1 + 2 + 3 + 4 + 5) / 3
= 5
 10*10/5
= 20
 10*10/3
```

Arithmetic puzzles

```
 1 + 2 + 3 + 4 + 5 / 3
= 11
 (1 + 2 + 3 + 4 + 5) / 3
= 5
 10*10/5
= 20
 10*10/3
= 33
```

Arithmetic puzzles

```
 1 + 2 + 3 + 4 + 5 / 3
= 11
 (1 + 2 + 3 + 4 + 5) / 3
= 5
 10*10/5
= 20
 10*10/3
= 33
 10*(10/3)
```

Arithmetic puzzles

```
 1 + 2 + 3 + 4 + 5 / 3
= 11
 (1 + 2 + 3 + 4 + 5) / 3
= 5
 10*10/5
= 20
 10*10/3
= 33
 10*(10/3)
= 30
```

Arithmetic puzzles

```
10*10/3/2
```

Arithmetic puzzles

```
  10*10/3/2
= 16
```

Arithmetic puzzles

```
  10*10/3/2
= 16
  10*10+3/2
```

Arithmetic puzzles

```
  10*10/3/2
= 16
  10*10+3/2
= 101
```

Arithmetic puzzles

```
  10*10/3/2
= 16
  10*10+3/2
= 101
  10*(10+3)/2
```

Arithmetic puzzles

```
  10*10/3/2
= 16
  10*10+3/2
= 101
  10*(10+3)/2
= 65
```

Increment and decrement operators

- Increment and decrement operators:

  a++ or ++a
  will add one to a.

- The difference is in what the expression evaluates to.

  Postfix: a++ will increment a **after** returning the (old) value of a.

  Prefix: ++a will increment a **before** returning the (new) value of a.

What do these loops output:

```
int a = 0;
while ( a++ < 10 ) printf("%d ",a);
int b = 0;
while ( ++b < 10 ) printf("%d ",b);
```

What do these loops output:

```
int a = 0;
while ( a++ < 10 ) printf("%d ",a);
printf("%d\n",a);
```

Output: 1 2 3 4 5 6 7 8 9 10 11

```
int b = 0;
while ( ++b < 10 ) printf("%d ",b);
printf("%d\n",b);
```

Output: 1 2 3 4 5 6 7 8 9 10

Comparing integers

Comparison operators:

- == for equality
- != for inequality
- >= for greater-than-or-equal-to
- <= for less-than-or-equal-to

Comparing integers

```c
#include <stdio.h>

int main()
{
  int a, b;
  scanf( "%d %d", &a, &b );

  if ( a = b ) printf( "equal\n" );
  else printf( "not equal\n" );
}
```

What is the mistake here?

Comparing integers

```c
#include <stdio.h>

int main()
{
  int a, b;
  scanf( "%d %d", &a, &b );

  if ( a == b ) printf( "equal\n" );
  else printf( "not equal\n" );
}
```

Remember: = for assignment, == for equality

Logical operators

- Logical operators are used to form Boolean expressions.
  - `&&` **(logical AND)**
  - `||` **(logical OR)**
  - `!` **(logical NOT)**
- Their functions are as follows:
  - `(a && b)` is true only if *both* `a` and `b` are true.
  - `(a || b)` is true if *either* `a` or `b` is true.
  - `(!a)` is true if `a` is false.

Logical operators

```
bool a = true;
bool b = true;
bool c = false;
bool d = true;

if ( a && b )
{
  if ( c || d )
  {
    printf("check\n");
  }
}
```

Logical operators

```
bool a = true;
bool b = true;
bool c = false;
bool d = true;

// this is equivalent to the previous slide
if ( ( a && b ) && ( c || d ) )
{
  printf("check\n");
}
```

for loops

- A `for` loop can be used to compactly produce counter-controlled iteration.
- The syntax of a for loop is as follows:

```
for ( initialization ; condition ; increment )
{
statements
}
```

for loops sequence of execution

```
for ( /*initialization*/ ; /*condition*/ ; /*increment*/ )
{
    /* statements */
}
```

1. Execute *initialization*.

   Loop:

   1. Test *condition* – stop if false
   2. Execute *statements*
   3. Execute *increment*

What does this loop output:

```
int a = 0;
for ( a = 0; a++ < 10; ) printf("%d ",a);
printf("%d ",a);
```

What does this loop output:

```
int a = 0;
for ( a = 0; a++ < 10; ) printf("%d ",a);
printf("%d ",a);
```

Output: `1 2 3 4 5 6 7 8 9 10 11`

What does this loop output:

```
int a = 0;
for ( a = 0; a++ < 10; ++a ) printf("%d ",a);
printf("%d ",a);
```

What does this loop output:

```
int a = 0;
for ( a = 0; a++ < 10; ++a ) printf("%d ",a);
printf("%d ",a);
```

Output: `1 3 5 7 9 11`

Multiple selection using switch

```
switch ( char c ) {
  case 'a':
  case 'A':
    a_grades++;
    break;

  case 'b':
  case 'B':
    b_grades++;
    break;

  // ...

  default:
    puts("error: bad grade input");
}
```

Text input/output

Text input/output functions:

```
char c = getchar(); // reads a single character
char str[1024]; scanf("%s",str); // reads a line of text
putchar(c); // writes a single character
printf("%s",str); // writes a string
```

Text input/output

Checking for EOF:

```
int a_count = 0;
int b_count = 0;
char c;
while ( ( c = getchar() ) != EOF )
{
  if ( c == 'a' ) a_count++;
  else if ( c == 'b' ) b_count++;
}
```

Defining, initializing, accessing arrays

```
int a[10] = { 1, 2 }; // remaining values will be zero
int b = a[0];
a[2] = a[1] - a[0];
a[10] = 20; // out-of-bounds -- not a compiler error
```

Iterating over arrays

```
for ( int i = 0; i < 10; i++ ) a[i] = 10-i;
```

What will the array contain?

Iterating over arrays

```
for ( int i = 0; i < 10; i++ ) a[i] = 10-i;
```

What will the array contain?

10 9 8 7 6 5 4 3 2 1

Specifying a size with #define

```
#define ARRAY_SIZE 5
int a[ARRAY_SIZE];
for ( int i = 0; i < ARRAY_SIZE; i++ ) a[i] = 2*i;
```

What will the array contain?

Specifying a size with #define

```
#define ARRAY_SIZE 5
int a[ARRAY_SIZE];
for ( int i = 0; i < ARRAY_SIZE; i++ ) a[i] = 2*i;
```

What will the array contain?

0 20 40 60 80 100

Specifying a size with #define

```
#define ARRAY_SIZE 5
int a[ARRAY_SIZE];
for ( int i = 0; i < ARRAY_SIZE; i++ ) a[i] = 2*i;
```

What will the array contain?

0 20 40 60 80 100

Storing strings in arrays

```
char str[] = "Hello, world!";

// print string
printf("%s\n",str);

// print individual characters
int i = 0;
while ( str[i] != '\0' ) putchar(str[i++]);
puts("");

// print ASCII values
i = 0;
while ( str[i] != '\0' ) printf("%d ",str[i++]);
```

Passing arrays to functions

```
float sumArray( float array[], int size ) {
  float sum = 0;
  for ( int i = 0; i < size; i++ ) sum += array[i];
  return sum;
}

int main() {
  float my_array[3] = { -1.f, 0.f, 1.f };
  float my_sum = sumArray(my_array,3);
  printf("sum: %f\n", my_sum );
}
```

Remember: there is no `array.length()` or `len(array)` in C!!

You could use `sizeof(array)/sizeof(array[0])` but this does not work when the array is passed as an argument to a function.

Searching arrays

```
// finds the first occurence of key in array
// returns -1 if key is not found
int linearSearch( const int array[], int key, int size )
{
    for ( int n = 0; n < size; n++ ) {
        if ( array[n] == key ) return n; // return location of key
    }
    return -1; // key not found
}
```

Searching arrays

```
// counts the number of occurences of key in array
int countFrequency( const int array[], int key, int size )
{
    int count = 0;
    for ( int n = 0; n < size; n++ ) {
        if ( array[n] == key ) count++;
    }
    return count;
}
```

Random numbers

```
#include <stdlib.h>
// ...

// initialize array of 10 integers to 0
int counter[10] = { 0 };

// count the frequency of randomly generated numbers
for ( int i = 0; i < 10000; i++ )
{
    // use modulo to restrict range to [0 9]
    int x = rand() % 10;

    // increment the corresponding counter
    counter[x]++;
}
```

Scope

What does this loop output:

```
int a = 0;
for ( int a = 0; a++ < 10; ) printf("%d ",a);
printf("%d ",a);
```

Scope

What does this loop output:

```
int a = 0;
for ( int a = 0; a++ < 10; ) printf("%d ",a);
printf("%d ",a);
```

Output: `1 2 3 4 5 6 7 8 9 10 0`

The `for` loop creates a new variable `a` in its scope which hides the `a` in the outer scope.

Scope

```
int a = 10;
{ a += 10; }
printf("%d\n",a);
```

Output?

Scope

```
int a = 10;
{ a += 10; }
printf("%d\n",a);
```

Output?

20 – the curly braces do not hide a, that happens only when we define a new variable with th same name as an existing one.

Scope

```
int a = 10;
{ int a = 20; }
printf("%d\n",a);
```

Output: 10 – here the second a hides the first one.

```
enum ArrayOp { SUM, AVERAGE, MIN, MAX };

int analyzeArray( int array[], int size, enum ArrayOp op )
{
  int result;
  switch ( op )
  {
    case SUM:
      result = 0;
      for ( int i = 0; i < size; i++ ) result += array[i];
      break;

    case AVERAGE:
      // ....
  }
  return result;
}
```

Next Time

- Midterm Thursday
- Next week: function pointers, strings