# Intermediate Programming

## Day 4

# Outline

- Logical operators
- Control structures
- Assignment and increment/decrement
- Loops
- Review questions

# Logical operators

Takes boolean value(s) (including integers acting as Boolean values) and returns a boolean value

- Unary:

| | | |
|---|---|---|
| ! | logical “not” | !A is true iff. A is false |

- Binary:

| | | |
|---|---|---|
| && | logical “and” | (A && B) is true iff. both A and B are true |
| \|\| | logical “or” | (A \|\| B) is true iff. either or both A and B are true |

# Logical operators

Takes integer/floating-point value and returns a boolean value

- Equality operators:
  - `==` (`A == B`) is true iff `A` equals `B`*
  - `!=` (`A != B`) is true iff `A` does not equal `B`*
- Relational operators
  - `>` (`A > B`) is true iff `A` is greater than `B`
  - `<` (`A < B`) is true iff `A` is less than `B`
  - `>=` (`A >= B`) is true iff `A` is greater than or equal to `B`*
  - `<=` (`A <= B`) is true iff `A` is less than or equal to `B`*

*You should avoid using these to compare floating point values!

# Outline

- Logical operators
- Control structures
- Assignment and increment/decrement
- Loops
- Review questions

# Control structures

- The if statement evaluates a boolean predicate and executes the code in braces if the predicate is true.

```
#include <stdio.h>
int main( void )
{
        int n = 12;
        if( n % 2 == 0 )
        {
                printf( "E\n" );
        }
        return 0;
}
```

```
>> ./a.out
E
>>
```

# Control structures

- The if statement evaluates a boolean predicate and executes the code in braces if the predicate is true.
- It no braces are provided, the if only affects the next command (i.e. up to the next ";").

```
#include <stdio.h>
int main( void )
{
        int n = 12;
        if( n % 2 == 0 )
                printf( "E\n" );
        return 0;
}
```

Note: White-space / indentation has no effect on what the if applies to.

# Control structures

- The if statement evaluates a boolean predicate and executes the code in braces if the predicate is true.
- It no braces are provided, the if only affects the next command (i.e. up to the next ";").
- Can even put on one line (if it's readable).

```
#include <stdio.h>
int main( void )
{
        int n = 12;
        if( n % 2 == 0 ) printf( "E\n" );
        return 0;
}
```

Note: White-space / indentation has no effect on what the if applies to.

# Control structures

- The **if / else** statement evaluates a boolean predicate and follows the **if** branch if the predicate is true and the **else** branch otherwise.

```
#include <stdio.h>
int main( void )
{
        int n = 13;
        if( n % 2 == 0 )
        {
                printf( "E\n" );
        }
        else
        {
                printf( "O\n" );
        }
        return 0;
}
```

```
>> ./a.out
O
>>
```

# Control structures

- The **if / else** statement evaluates a boolean predicate and follows the **if** branch if the predicate is true and the **else** branch otherwise.
- It no braces are provided, the **if / else** only effect the next command (i.e. up to the next ";").

```
#include <stdio.h>
int main( void )
{
        int n = 13;
        if( n % 2 == 0 ) printf( "E\n" );
        else             printf( "O\n" );
        return 0;
}
```

# Control structures

- The **if / else** statement evaluates a boolean predicate and follows the **if** branch if the predicate is true and the **else** branch otherwise.
- It no braces are provided, the **if / else** only effect the next command (i.e. up to the next ";").
- The **else** is always associated to the last (unmatched) **if**.

```
#include <stdio.h>
int main( void )
{
        int n = 13;
        if( n % 2 == 0 )
                if( n==11 ) printf( "11\n" );
                else        printf( "E\n" );
        return 0;
}
```

```
>> ./a.out
>>
```

# Control structures

- The **if / else** statement evaluates a boolean predicate and follows the **if** branch if the predicate is true and the **else** branch otherwise.
- It no braces are provided, the **if / else** only effect the next command (i.e. up to the next ";").
- The **else** is always associated to the last (unmatched) **if**.

```
#include <stdio.h>
int main( void )
{
        int n = 13;
        if( n % 2 == 0 )
        {
                if( n==11 ) printf( "11\n" );
        }
        else printf( "O\n" );
        return 0;
}
```

```
>> ./a.out
0
>>
```

# Control structures

- The **if / else if / else** statement evaluates a sequence of boolean predicates, and executes the code when the predicate is true.

```
#include <stdio.h>
int main( void )
{
        int x = 79;
        if      ( x >= 90 ) printf( "A\n" );
        else if( x >= 80 ) printf( "B\n" );
        else if(x >= 70)   printf( "C\n" );
        else if(x >= 60)   printf( "D\n" );
        else               printf( "F\n" );
        return 0;
}
```

```
>> ./a.out
C
>>
```

# Control structures

- The **switch** statement tests if a value matches one of a set of prescribed cases and executes *all* the code after if it does.
  - **switch**:
    Specifies the value to be tested
  - **case**:
    specifies the case to execute
  - **break**:
    do not continue to the next case
  - **default**:
    if nothing else matched...

```
#include <stdio.h>
int main( void )
{
        char grade = 'C';
        int points = 0;
        switch( grade )
        {
                case 'A':
                        points = 4;
                        break;
                case 'B':
                        points = 3;
                        break;
                case 'C':
                        points = 2;
                        break;
                case 'D':
                        points = 1;
                        break;
                default:
                        points = 0;
                        break;
        }
        printf( "Grade %c -> %d GPA points\n", grade, points);
}
```

```
>> ./a.out
Grade C -> 2 points
>>
```

# Control structures

Short-circuiting:

- When C evaluates the composition of logical expression. . .*

```
if( (statement_1) || (statement_2) )
if( (statement_1) && (statement_2) )
```

. . . it *short circuits* as soon as answer is definitely true or definitely false.

- `if( a == 7 || b == 7 )`:

  When `(a==7)` is true, the entire expression is true so we don't need to test if `(b == 7)` is true.

- `if( a == 7 && b == 7 )`:

  When `(a==7)` is false, the entire expression is false so we don't need to test if `(b == 7)` is true.

*This statement remains true even when the composition is not the predicate of an `if` statement.

# Outline

- Logical operators
- Control structures
- Assignment and increment/decrement
- Loops
- Review questions

# Compound assignment

Combine binary operators with assignment operators:

```
A += B;   ⇒   A = A+B;
A -= B;   ⇒   A = A-B;
A *= B;   ⇒   A = A*B;
A /= B;   ⇒   A = A/B;
A %= B;   ⇒   A = A%B;
```

The right hand side can be either a variable or a constant

# Increment and decrement

Increase / decrease the value by one:

| | | |
|---|---|---|
| `A++;` | ⇒ | `A = A+1;` |
| `A--;` | ⇒ | `A = A-1;` |
| `++A;` | ⇒ | `A = A+1;` |
| `--A;` | ⇒ | `A = A-1;` |

The difference between ***A++*** and ***++A*** (or ***A--*** and ***--A***) is precedence.

# Increment and decrement

Increase / decrease the value by one:

```
B = A++;  ⇒   { B = A;      A = A+1;          }
B = A--;  ⇒   { B = A;      A = A-1;          }
B = ++A;  ⇒   {             A = A+1;    B = A; }
B = --A;  ⇒   {             A = A-1;    B = A; }
```

# Increment and decrement

Increase / decrease the value by one:

```
#include <stdio.h>
int main( void )
{
        int i = 0;
        if( ++i ) printf( "++i was non-zero\n" );
        printf( "i=%d\n" ,  i );

        i = 0;
        if( i++ ) printf( "i++ was non-zero\n" );
        printf( "i=%d\n" , i );
}
```

```
>> ./a.out
++i was non-zero
i=1
i=1
>>
```

# Outline

- Logical operators
- Control structures
- Assignment and increment/decrement
- Loops
- Review questions

# Loops

The **for** loop

```
#include <stdio.h>
int main( void )
{
        for( int i=0 ; i<10 ; i++ )
        {
                printf( "%d\n" , i );
        }
```

```
>> ./a.out
0
1
2
3
4
5
6
7
8
9
>>
```

# Loops

The `for` loop:

- Initializes a loop variable

```
#include <stdio.h>
int main( void )
{
        for( int i=0 ; i<10 ; i++ )
        {
                printf( "%d\n" , i );
        }
}
```

# Loops

The `for` loop:

- Initializes a loop variable
- Iterates while the looping condition is met

```
#include <stdio.h>
int main( void )
{
        for( int i=0 ; i<10 ; i++ )
        {
                printf( "%d\n" , i );
        }
}
```

# Loops

The `for` loop:

- Initializes a loop variable
- Iterates while the looping condition is met
- Adjusts the loop value after each iteration

```
#include <stdio.h>
int main( void )
{
        for( int i=0 ; i<10 ; i++ )
        {
                printf( "%d\n" , i );
        }
}
```

# Loops

The `for` loop:

- Initializes a loop variable
- Iterates while the looping condition is met
- Adjusts the loop value after each iteration
- Performs the calculation in braces at each iteration

```
#include <stdio.h>
int main( void )
{
        for( int i=0 ; i<10 ; i++ )
        {
                printf( "%d\n" , i );
        }
}
```

# Loops

The `for` loop:

- Initializes a loop variable
- Iterates while the looping condition is met
- Adjusts the loop value after each iteration
- Performs the calculation in braces at each iteration
  - If no braces are provided, it performs the next command

```
#include <stdio.h>
int main( void )
{
        for( int i=0 ; i<10 ; i++ )
                printf( "%d\n" , i );
}
```

# Loops

The **while** loop:

- Iterates until the **while** condition fails.
- Performs the calculation in braces at each iteration

```
#include <stdio.h>
int main( void )
{
        int i = 1;
        while( (i%7) != 0 )
        {
                printf( "%d\n" , i );
                i++;
        }
}
```

```
>> ./a.out
1
2
3
4
5
6
>>
```

# Loops

The **while** loop:

- Iterates until the **while** condition fails.
- Performs the calculation in braces at each iteration

How about this?

```
#include <stdio.h>
int main( void )
{
        int i = 0;
        while( (i%7) != 0 )
        {
                printf( "%d\n" , i );
                i++;
        }
}
```

```
>> ./a.out
>>
```

# Loops

The **while** loop:

- Iterates until the **while** condition fails.
- Performs the calculation in braces at each iteration
  - If no braces are provided, it performs the next command

```
#include <stdio.h>
int main( void )
{
        int i = 1;
        while( (i%7) != 0 )
                printf( "%d\n" , i++ );
}
```

# Loops

The **while** loop:

- Iterates until the **while** condition fails.

Note that a **for** loop can always be implemented as a **while** loop.

```
#include <stdio.h>
int main( void )
{
        int i = 1;
        while( (i%7) != 0 )
                printf( "%d\n" , i++ );
}
```

```
#include <stdio.h>
int main( void )
{
        for( int i=1 ; (i%7) != 0 ; i++ )
                printf( "%d\n" , i );
}
```

# Loops

The do / while loop:

- Like a while loop, but is always guaranteed to perform at least one iteration (i.e. tests the condition after the loop, not before)
- Performs the calculation in braces at each iteration

```
#include <stdio.h>
int main( void )
{
        int i = 0;
        do
        {
                printf( "%d\n" , i );
                i++;
        }
        while( (i%7) != 0 );
}
```

# Loops

The do / while loop:

- Like a while loop, but is always guaranteed to perform at least one iteration (i.e. tests the condition after the loop, not before)
- Performs the calculation in braces at each iteration
  - If no braces are provided, it performs the next command

```
#include <stdio.h>
int main( void )
{
        int i = 0;
        do printf( "%d\n" , i++ );
        while( (i%7) != 0 );
}
```

# Loops (summary)

- `while( boolean expression ){ statements }`
  - Iterates 0 times, as long as `boolean expression` is true
  - Execute `statements` at each iteration
- `do { statements } while ( boolean expression )`
  - Iterates 1 times, as long as `boolean expression` is true
  - Execute `statements` at each iteration
- `for( init ; boolean expression ; update ){ statements }`
  - `init` happens first; usually declares & assigns “index variable”
  - Iterates 0 times, as long as `boolean expression` is true
  - Execute `statements` at each iteration
  - `update` is run after `statements`; often it increments the loop variable (`i++`)

# Loops (summary)

- while( boolean expression ) { stat
  - Iterates 0 times, as long as boolean exp
  - Execute statements at each iteration
- do { statements } while ( boolean e
  - Iterates 1 times, as long as boolean exp
  - Execute statements at each iteration
- for( init ; boolean expression
  - init happens first; usually declares & ass
  - Iterates 0 times, as long as boolean exp
  - Execute statements at each iteration

```
#include <stdio.h>
int main( void )
{
        int i = 0;
        do
        {
                printf( "%d\n" , i++ );
                if( (i%7) != 0 )
                        break;
        }
        while( true );
}
```

If statements has the command break, the code terminates the loop regardless of whether or not boolean expression is true.

# Loops (summary)

- while( boolean expression ) { stat
  - Iterates 0 times, as long as boolean exp
  - Execute statements at each iteration
- do { statements } while ( boolean e
  - Iterates 1 times, as long as boolean exp
  - Execute statements at each iteration
- for( init ; boolean expression ; update ) { sta
  - init happens first; usually declares & assigns “index variable”
  - Iterates 0 times, as long as boolean expression is true
  - Execute statements at each iteration

```
#include <stdio.h>
int main( void )
{
        for( int i=0 ; i<6 ; i++ )
        {
                if( i==3 ) continue;
                printf( "%d\n" , i );
        }
}
```

```
>> ./a.out
0
1
2
4
5
>>
```

If statements has the command continue, the code will skip the remainder of the statements block

# Outline

- Logical operators
- Control structures
- Assignment and increment/decrement
- Loops
- Review questions

# Review questions

1. Which one is the logical "and" operator in C, "&&" or "&" or both?

# Review questions

2. Which one is the logical "negation" operator in C, "~" or "!" or both?

# Review questions

3. What does the keyword `break` do in loops?

# Review questions

4. What does the keyword `continue` do in loops?

# Review questions

5. How many times is the *initialize* statement in a `for` loop executed?

# Exercise 2-1

- Website -> Course Materials -> Ex2-1