

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.
- If 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.
- If 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
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.
- If 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.
- If 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.
- If 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:

$A += B; \Rightarrow A = A+B;$

$A -= B; \Rightarrow A = A-B;$

$A *= B; \Rightarrow A = A*B;$

$A /= B; \Rightarrow A = A/B;$

$A \% = B; \Rightarrow A = A\%B;$

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

Increment and decrement

Increase / decrease the value by one:

$A++;$	\Rightarrow	$A = A+1;$
$A--;$	\Rightarrow	$A = A-1;$
$++A;$	\Rightarrow	$A = A+1;$
$--A;$	\Rightarrow	$A = A-1;$

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

Increment and decrement

Increase / decrease the value by one:

$B = A++;$	\Rightarrow	$\{ B = A;$	$A = A+1;$	$\}$
$B = A--;$	\Rightarrow	$\{ B = A;$	$A = A-1;$	$\}$
$B = ++A;$	\Rightarrow	$\{$	$A = A+1;$	$B = A; \}$
$B = --A;$	\Rightarrow	$\{$	$A = A-1;$	$B = A; \}$

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 (and vice versa).

```
#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) { 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 ; increment) { statements }**
 - init happens first; usually declares & assigns a variable
 - Iterates 0 times, as long as boolean expression is true
 - 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) { 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

```
#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

(i++)

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 is the result of evaluating:
 $(34+2)/40 \ || \ 80 > 'A' \ \&\& \ 15\%4$

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	
29	1d	035	GS	61	3d	075	=	93	5d	135]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

Review questions

3. What is the result of evaluating:
 $(34+2)/40 \parallel 80>'A' \&\& 15\%4$

$\Rightarrow 36/40 \parallel 80>65 \&\& 15\%4$

$\Rightarrow (36/40) \parallel (80>65) \&\& (15\%4)$

$\Rightarrow \text{false} \parallel \text{true} \&\& \text{true}$

$\Rightarrow \text{false} \parallel (\text{true} \&\& \text{true})$

$\Rightarrow \text{false} \parallel \text{true}$

$\Rightarrow \text{true}$

Precedence	Operator	Associativity
1	++ -- () [] . -> (type){list}	Left-to-right
2	++ -- + - ! ~ (type) * & sizeof _Alignof	Right-to-left
3	* / %	Left-to-right
4	+ -	
5	<< >>	
6	< <= > >=	
7	== !=	
8	&	
9	^	
10		
11	&&	
12		

Review questions

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

Terminates the loop

Review questions

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

Code skips the remainder of the loop block

Review questions

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

1

Exercise 4

- Website -> Course Materials -> Exercise 4