Programs with Loops: The for Loop)

ESC101: Fundamentals of Computing

Nisheeth

Announcements

- Major Quiz 1 this Wednesday, Jan 29, 12pm-1pm, L-20
- Don’t be late. Don’t be absent
- Must carry your Student ID
- No material allowed except one haA4 sheet of paper
- Answers to be written on question paper itself (just like minor quizzes)
  - Have to write name and roll number on both sides of each sheet
  - Any sheet missing both details will not be graded
- Carry pencil, eraser, sharpener, pen
  - Must write final answers using pen

Bitwise Operators (not in Major Quiz 1)

Operation	C Code	a	b	c	d	e	f
BITWISE AND	c = a & b	0000	1111	0000	1111	1111	1111
BITWISE OR	d = a | b	0101	1100	0100	1101	1001	1010
BITWISE XOR	e = a ^ b	1010	1110	1010	1110	0100	0101
BITWISE COMPLEMENT	f = ~a	1001	0111	0001	1111	1110	0110

Bitwise AND Operator &

- The output of bitwise AND is 1 if the corresponding bits of two operands are both 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0
- In C Programming, bitwise AND operator is denoted by &

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bitwise AND of 12 and 25
   0000 1100
& 0001 1001
  __________
   0000 1000  = 8 (In decimal)
```

```c
#include <stdio.h>
int main(){
    int a = 12, b = 25;
    printf("Output = %d", a & b);
    return 0;
}
```

Bitwise OR Operator |

- The output of bitwise OR is 1 if at least one of the corresponding bit of two operands is 1
- In C Programming, bitwise OR operator is denoted by |

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bitwise OR of 12 and 25
   0000 1100
|  0001 1001
  __________
   00011101  = 29 (In decimal)
```

```
#include <stdio.h>
int main(){
    int a = 12, b = 25;
    printf("Output = %d", a | b);
    return 0;
}
```

Bitwise XOR Operator ^

- The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite i.e. one is 1 and the other is 0
- In C Programming, bitwise XOR operator is denoted by ^

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bitwise XOR of 12 and 25
   00001100
^ 00011001
  ________
  00010101  = 21 (In decimal)
```

```
#include <stdio.h>
int main(){
    int a = 12, b = 25;
    printf("Output = %d", a^b);
    return 0;
}
```

Bitwise Complement Operator ~

- A unary operator that simply flips each bit of the input
- In C Programming, bitwise complement operator is denoted by ~

```
12 = 0000 0000  0000 0000  0000 0000  0000 1100
Bitwise complement of 12
~ 0000 0000  0000 0000  0000 0000  0000 1100
  ________________________________________
  1111 1111   1111  1111    1111 1111   1111
0011
= -13 (decimal)
```

```
#include <stdio.h>
int main(){
    int a = 12;
    printf("Output = %d", ~a);
    return 0;
}
```

Right Shift Operator >>

- Right shift operator shifts all bits towards right by a certain number of locations
- Bits that “fall off” from the right most end are lost
- Blank spaces in the leftmost positions are filled with sign bits
- 212 = 0000 0000   0000 0000   0000 0000   1101 0100
- 212 >> 0 = 0000 0000   0000 0000   0000 0000   1101 0100
- 212 >> 4 = 0000 0000   0000 0000   0000 0000   0000 1101
- 212 >> 6 = 0000 0000   0000 0000   0000 0000   0000 0011
- 212 >> 3 = 0000 0000   0000 0000   0000 0000   0001 1010
- Right shift by k is equivalent to integer division with 2^k

Left Shift Operator <<

- Left shift operator shifts all bits towards left by a certain number of locations
- Bits that “fall off” from the left most end are lost
- Blank spaces in the right positions are filled with 0s
- 212 = 0000 0000   0000 0000   0000 0000   1101 0100
- 212 << 0 = 0000 0000   0000 0000   0000 0000   1101 0100
- 212 << 4 = 0000 0000   0000 0000   0000 1101   0100 0000
- 212 << 6 = 0000 0000   0000 0000   0011 0101   0000 0000
- 212 << 28 = 0100 0000   0000 0000   0000 0000   0000 0000
- Left shift by k is equivalent to integer multiplication with 2^k

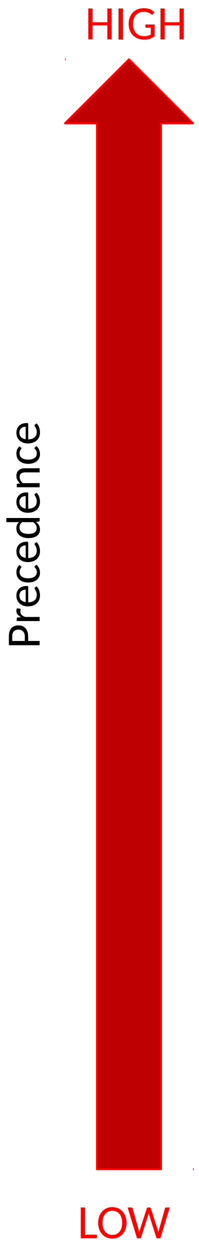
Example use of bitwise operators

- Can use “masks” to extract certain bits of a number
- Suppose I want to look at the last 6 bits of a number a
- Create a mask with only last bits set to 1 and take & with a

```
a = 0000 0000  0000 0000  0000 0001  1010 1011
p = 0000 0000  0000 0000  0000 0000  0000 0001
q = 0000 0000  0000 0000  0000 0000  0100 0000
m = 0000 0000  0000 0000  0000 0000  0011 1111
r =  0000 0000  0000 0000  0000 0000  0010 1011
```

```c
int a = 427;
int p = 1;
int q = p << 6;
int m = q – 1;
int r = a & m;
printf("%d", r); // 43
```

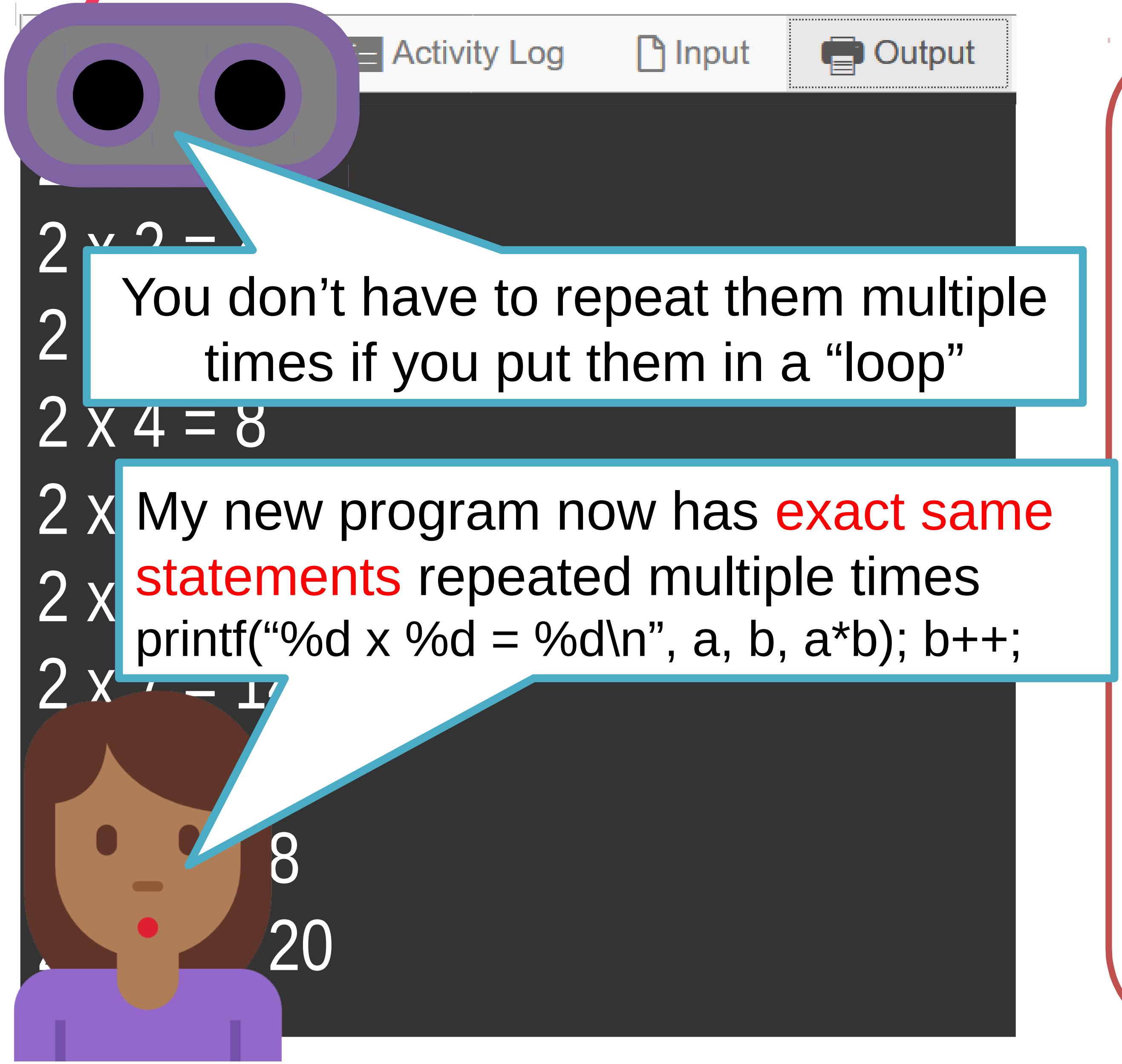
Precedence Table with Bitwise Operators



Operators	Description	Associativity
unary + -, ++, --, type, sizeof, ~	Unary plus/minus, increment/decrement, typecast, sizeof, bitwise complement	Right to left
* / %	Arithmetic: Multiply, divide, remainder	Left to right
+ -	Arithmetic: Add, subtract	Left to right
<< >>	Bitwise left-shift, bitwise right shift	Left to right
< > >= <=	Relational operators	Left to right
== !=	Relational operators	Left to right
&	Bitwise AND	Left to right
^	Bitwise XOR	Left to right
|	**Bitwise OR**	**Left to right**
&&	Logical AND	Left to right
||	Logical OR	Left to right
? :	Conditional	Right to left
=	Assignment	Right to left

Programs with Loops

Printing the multiplication table of 2



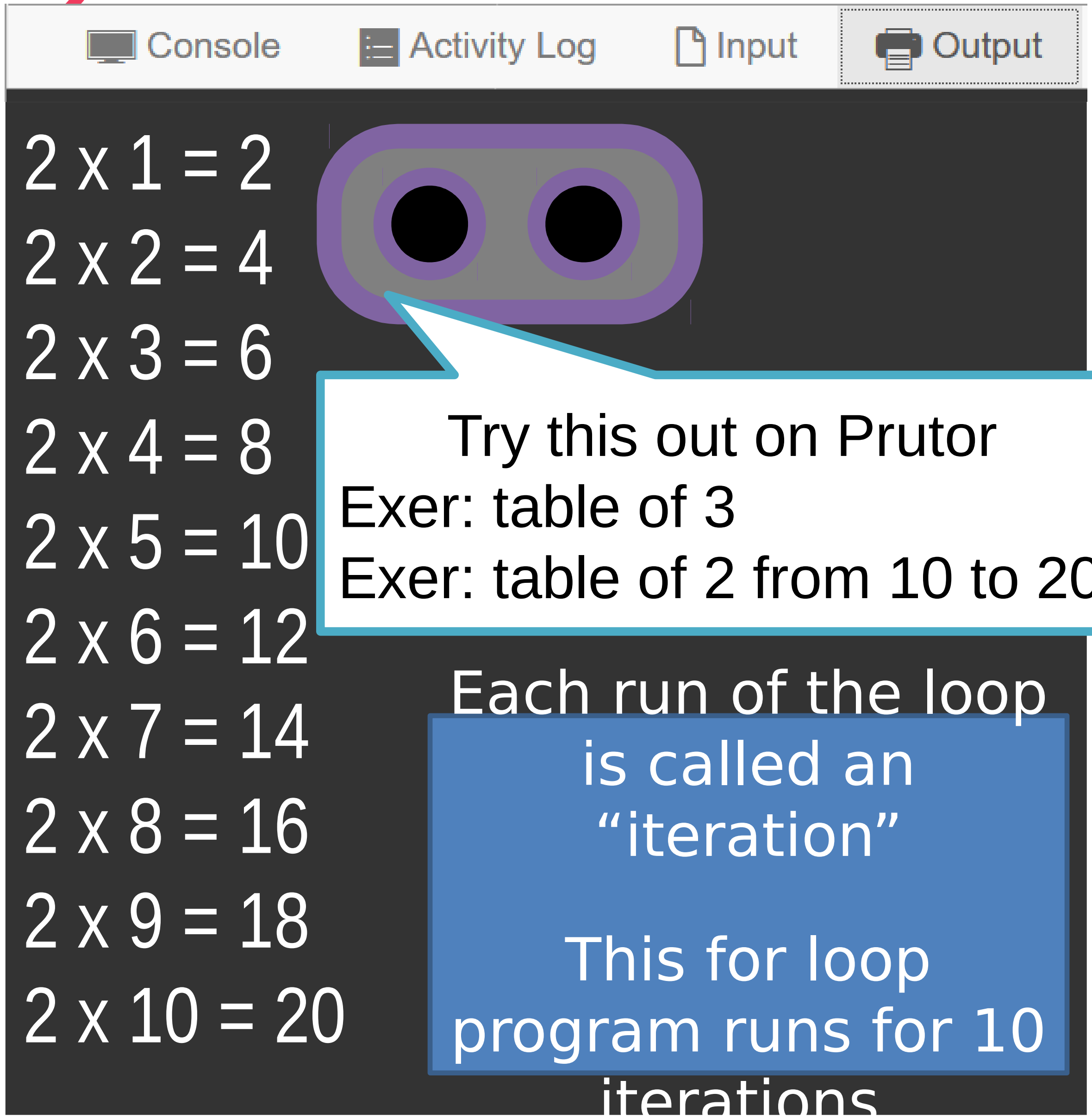
```
2 x 2 =
2
2 x 4 = 8
2 x
2 x
2 x 7 = 1
8
20
```

You don’t have to repeat them multiple times if you put them in a “loop”

My new program now has exact same statements repeated multiple times printf(“%d x %d = %d\n”, a, b, a*b); b++;

```
int a = 2, b = 1;
printf(“%d x %d = %d\n”, a, b, a*b);
b++;
printf(“%d x %d = %d\n”, a, b, a*b);
b++;
printf(“%d x %d = %d\n”, a, b, a*b);
b++;
printf(“%d x %d = %d\n”, a, b, a*b);
b++;
…
```

Printing the multiplication table of 2



```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

Try this out on Prutor
Exer: table of 3
Exer: table of 2 from 10 to 20

Each run of the loop is called an "iteration"

This for loop program runs for 10 iterations

```
int a = 2, b;
for(b = 1; b <= 10; b++){
    printf("%d x %d = %d\n", a, b, a*b);
}
```

++b or b = b + 1
is also fine here

What does this code mean?

1. Let a = 2, b be integer variables
2. First set b = 1
3. Then check if b <= 10 or not
   1. If true, execute printf, execute b++ (or ++b or b=b+1), go to step 3
   2. If false (i.e. b > 10), stop looping

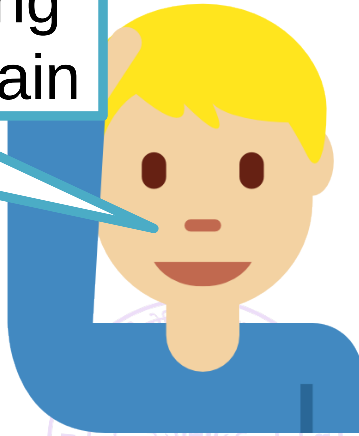
Does My Problem Need Loops?

Read the problem carefully and identify some tasks that have to be repeated again and again

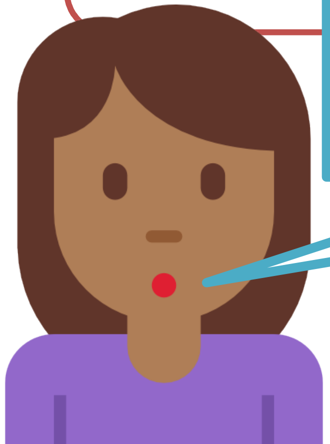
Use this variable that is changing as the loop counter

```
int a = 2, b;
for(b = 1; b <= 10; b++){
    printf(“%d x %d = %d\n”,
a, b, a*b);
}
```

Yes, but we could write the same code printf(“%d x %d = %d\n”, a, b, a*b); to do all the tasks by simply changing the value of variable b again and again

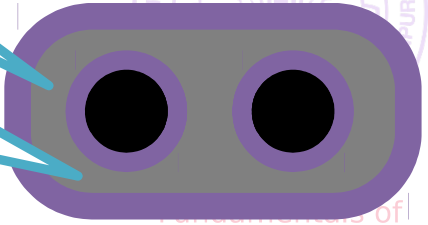


Yes, in the multiplication table example, the tasks were slightly different. First print 2 x 1 = 2, then print 2 x 2 = 4 etc etc.



Very Good!

The tasks may be slightly different from each other



Syntax and Fl

Brackets essential if you want me to do many things while looping

General form of the for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

Initialization expression is executed only once

What does this piece of code mean?

1. First do as specified in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
    Execute all statements inside braces
    Execute update expression
    Go back to step 2
  Else stop looping and execute rest of code

Syntax of the for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
 }
```

The entire for loop is considered one statement

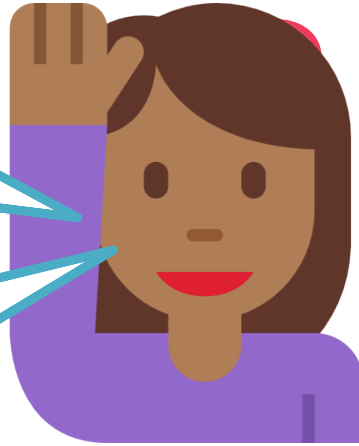
Can also put inside for loop: printf statements, if-else/switch statements, another for loop statement (nested for loop)

Usually init_expr, stopping_expr, update_expr involve the same variable, e.g. b in multiplication table example

Lovingly called variable of the loop/loop counter

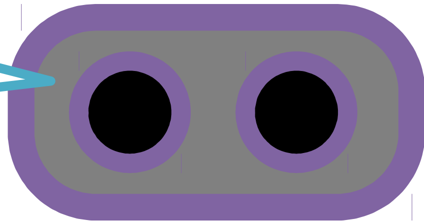
Syntax of the fo

```
for(init_expr; stopping_e
    statement1;
    statement2;
}
```



All expressions generate values, even assignment/relational ones

Mr C considers 0 to be FALSE and 1 (or anything non-zero) to be TRUE



Yes, you can write the init_expr before the loop and the update_expr inside the loop

stopping_expr must give true/false value

- Usually done by making stopping_expr a relational expression
- Warning: you can say b * 2 in stopping_expr but dangerous
- init_expr and update_expr can be anything you want

init_expr and update_expr can even be empty

```
for(;stopping_expr;){ ... }
```

Some common errors in loops

Initialization: forget to do it or did wrong initialization

Update: Forget to do update step or wrong update step

Termination: wrong or missing termination

for(b=1;**b<10**;b++){…} not same as for(b=1;**b<=10**;b++){…}

Infinite loop: The loop goes on forever. Never terminates.

for(b=2;b>=1,b++){…}

Prutor will give “TLE” error (time limit exceeded error)

Example: Find the smallest number

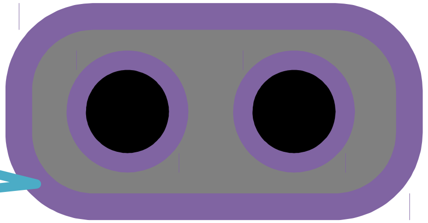
```
int main(){
    int total_num,curr_num,i;
    int min = INT_MAX; // initialize min as a very large integer
    scanf(“%d”,total_num); // read total number of inputs
    for(i = 1; i <= total_num; i++){
        scanf(“%d\n”,&curr_num); // read a number (each on a new line)
        if(curr_num <= min){
            min = curr_num;
        }
    }
printf(“Smallest number = %d”, min);
return 0;
}
```

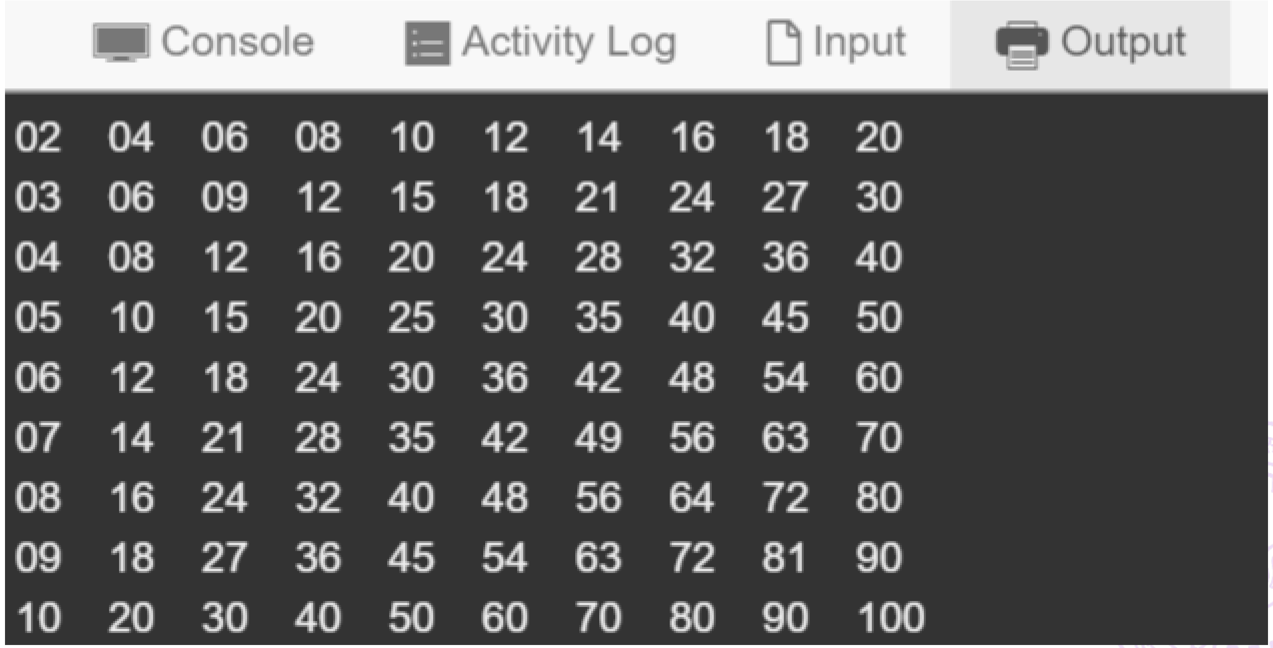
Note: Need limit.h for INT_MAX

Example: Print tables of 2 to 10

```
int main(){
int i,j,val;
for(i = 2; i <= 10; i++){
    for(j=1; j <= 10; j++){
        val = i*j;
        if(val < 10)
            printf("0%d\t",val); // prefix 0 if value < 10
        else
            printf("%d\t",val);
    }
    printf("\n"); // start a new line
}
return 0;
}
```

Example of nested for loop (for loop inside a for loop)





```
02   04   06   08   10   12   14   16   18   20
03   06   09   12   15   18   21   24   27   30
04   08   12   16   20   24   28   32   36   40
05   10   15   20   25   30   35   40   45   50
06   12   18   24   30   36   42   48   54   60
07   14   21   28   35   42   49   56   63   70
08   16   24   32   40   48   56   64   72   80
09   18   27   36   45   54   63   72   81   90
10   20   30   40   50   60   70   80   90   100
```

Use of break/continue in loops

to read 10 numbers and compute sum of those that are > 0.

and continue
only break from
and skip the loop
in

se

```
int main(){
    int i, curr_num, sum = 0; // no numbers seen yet. Sum initialized to 0
    for(i = 1; i <= 10; i++){            // loop will run (a maximum of) 10 times
        scanf("%d\n",&curr_num); // read a number
        if(curr_num == 0) break;    // if input equals 0, quit the loop
        else if (curr_num < 0) continue;  // if input < 0, skip and go to next iteration
loop
        else sum = sum + curr_num;    // if input > 0, add it to the sum
    }
    printf("Sum = %d", sum);   // print the sum of inputs that were > 0
    return 0;
}
```

Use break;
to exit the loop

Use continue; to skip
the current iteration
and go to next one