APS 105 Lecture Notes 8

Last time: if-else nested statements, De Morgan's Law, discussed a C code printing maximum of 3 #

Today: we discuss a more readible code to get max of 3 #s, while, for and do-while loops.

* Find maximum of 3 ints.

Last lecture

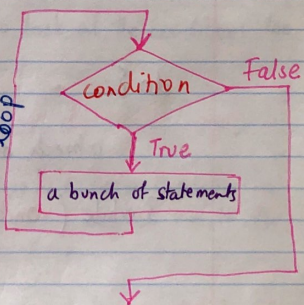
```
if (x>y)
    if (x>z)
        printf("%d", x);
    else
        printf("%d", z);
else
    if (y>z)
        printf("%d", y);
    else
        printf("%d", z);
```

More readible way:

```
if (x>y && x>z)
    printf("%d", x);
else if (y>x && y>z)
    printf("%d", y);
else
    printf("%d", z);
```

Loops

```
while ( condition ) {
    a bunch of statements;
}
```



Take numbers from the user and calculate the sum of these numbers until the user enters -1, you print the sum of the numbers to the user.

```
int input;
int sum = 0;

scanf("%d", input);
while (input != 0) {

    sum += input;
    scanf("%d", &input);
}
printf("The sum is %d", input);
```

→ can't initialize input to 0, because the loop will not execute.

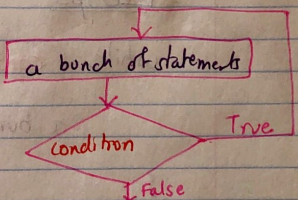
→ initialize input

Q- What happens if condition never goes false?

It is an infinite loop – it will never stop

* Not useful in this course, but elsewhere it is useful

```
do {
    a bunch of statements
} while (condition);
```



↖ you'll forget this and suffer :C

The body of the loop will be evaluated at least once regardless of the condition.

```
do {


} while (true);
```

This is an infinite loop, but is there a way out of this loop?

break is a keyword that can help us break a loop regardless of the condition — bad coding style.

E.g.
```
int x = -1;
do {
    if (x < 0)
        break;          ← This will break the loop and exit without evaluating printf
    printf("%d", x);
} while (true);
```

break is bad coding style as it is difficult to understand when will a loop exit (i) you have to look at breaks in the loop
(ii) you have to look at the conditions too.

Q- When do we use a while loop or do-while loop?

The difference is that do-while loop executes the body at least once, and while loop may not execute the body at all if the condition is false.

Hence, do-while loops are used to check for validity of input from user, since you need to take input from the user at least once.

Write C code that takes input from the user and checks if it is valid (between 0 and 1), if not the user is prompted to enter the number again. (4)

```
int inputNum;

do {
    printf("Please enter a number between 0 and 1");
    scanf("%d", &inputNum);
} while (inputNum < 0 || inputNum > 1)
```

need to check for validity of input

Exercise: Using a while loop print 15 "*" on separate lines

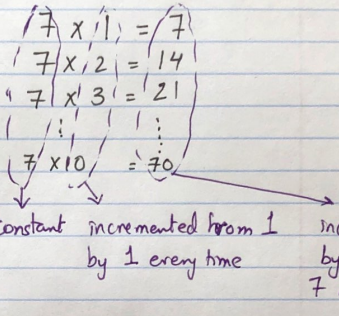
```
int count = 0;
while (count < 15) {
    printf("* \n");
    count += 1;
}
```

* count = 0
* count = 1
* count = 2
!
* count = 14

count is incremented after printing to 15 so count < 15 is false and loop exists.

Exercise: Using a while loop print time table of 7.

Output should be




```
int multOf = 7;
int count = 0;
while (count < 10) {
    printf("7 x %d = %d", (count+1), (count+1)*7);
}
```

Exercise: Using while loops, count # of digits in a number

We remove least significant digit in a number by dividing by 10

so 534/10 = 53 → 53/10 = 5 → 5/10 = 0

to remove 3 digits in a number I divided by 10 3 times until the number is 0

```
int digits = 5378;
int numOfdigits = 0;
while (digits > 0) {        → another way: digits != 0
    digits /= 10;
    numOfdigits++;
}
printf("Total digits: %d", numOfdigits);
```