APS 105 Lecture Notes 9

Last lecture: Make computers repeat - do-while and while loops

Today: For loops and nested loops

Recall example

Write a C program that prints 15 stars each on a separate line

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

you want to last enter the loop when count is 14, but exit when count is 15, as count starts from 0.

you want to enter the loop when count is 15, as count started from 1.

In the example above, we have a fixed # of times that we need to iterate/loop (e.g. 15), so the general form is

```
<initialization>;                               e.g. set count to 0
while (<condition>) {                            condition for entering loop.
    <statement>;                                 e.g. printf
    <change the variable in condition>;
}
```

There is a specific structure for this kind of loops (with fixed # of iterations: (i) initialization, (ii) condition, (iii) change variables in condition

It is the for loop

For the above example (printing 15 stars), this is how we do it in For loops.

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

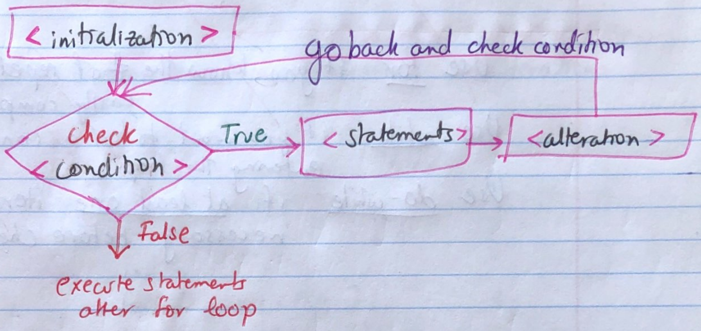
- no need for curly brackets if you have 1 statement, just like if statements

General form of for loops:

```
   for (<initialization>; <condition> ; <alteration >){
        <statements >;
   }
```

①initialization ②⑤condition ③statements ④alteration

Flow:



In C99 standard,
Initialization and declaration can be inside for loop.

```
for(int count=0; count<15; count++)
        printf("* \n");
```

But if count is declared in for loop, it can only be used inside for loop

count is "undefined" outside for loop

We say "scope" of count is limited to inside for loop

Variations on for loops.

1st expression Third expression

① You can omit <initialization> <alteration>

E.g.
```
int n=1;
for( ; n<=15 ; n++)
    printf("*");
```

E.g.
```
int n=1;
for ( ; n<=15; ){
    printf("*");
    n++;
}
```

② If 2nd expression <condition> is empty, then it is true

③ You can have complex expressions in for loop.

```
for (int n=1 ; n <=15; printf("*\n"), n++)
```

important → ;

Can only do this in 1st and 3rd part of the for statement header.

In the 2nd part <condition>, you can form complex expressions using ! && || > < <= >=

Important:

initializes and declares 2 variables

```
for (int m=1, n=10; ----)
```

different types

```
for (int m=1, double n=10.0; ---)
```

You can also have something like

complex condition

```
bool done = false;
for (int i = 1 ; i <=max && ! done ; i++) {
         ---
         if ( --- )
               done = true;      } if statement
                                   inside for loop.
         ~.
}
```

When to use for, do-while, while loop?

Use **for** if you know # of repetitions, or can easily compute it

Use **while** if you want to test a condition before entering the loop.

Use **do-while** if at least 1 iteration is necessary before checking the condition.

Nested loops

Just like if-statements can be nested, loops can be nested.

Write a C program that prints.

```
*
* *
* * *
* * * *
```

→ Break down the problem, how to print a particular number of stars in 1 line?

```
for (int count=0; count < 3; count++)
    printf("*");
```

```
* * *
```

→ To change this # of stars, you need to change 3 to a variable say row

```
for (int count=0; count<row; count++)
    printf("*");
```

if row is 1, it prints 1 star, if row is 2 you print 2 stars.

→ Change the variable row in an outer loop.

```
int lines = 4;
for (int row=0; row < lines ; row++) {
        for (int count=0; count <row ; count++)
                                                {
                print("*");
        }
        printf("\n"); → every line you need
                        to end it with endline.

}
```

Next lecture: More complex nested loops.