Day: Date: / /

Loops

While loop:

Syntax:

```
while (condition) {
    body;
}
```

Working:

- A while first checks the condition, if the condition is true then the body is run.
- After the whole body runs then the condition is checked again, and if true body runs again.
- This will keep on happening until condition becomes false. If the condition always remains true then the code will never terminate.

Examples

1- Many many examples but one is that you can use it for input validation.

Day: Date: / /

Counter loops:

- These loops work with range. This means they will run only for a certain count for a certain amount of times e.g:

```
1  int count = 0;
2
3  while ( count <5) {
4          cout << " Hello \n";
5          count++;
6  }
```

Output

Hello
Hello
Hello
Hello
Hello

How?

Dry run:

iteration	count
1st	1
2nd	2
3rd	3
4th	4
5th	5

so now as 5<5 This is not True so only 5 times

Using continue and break.

Break will end the loop

Continue will restart the loop from the condition

e.g

Break:

we want to end the loop when the value of a certain number reaches 20.

Continue:

we want the loop to continue but just skip the value 20.

if (value == 20) { continue; }

if we apply a break inside a switch which is inside of loop then it will end the loop if no other statement is with it

```
while ( )
{ switch () {
  case 1:
  case 2: break;
}
```

case 2 has only break statement

Day: Date: / /

Nesting loops for pattern formation:

- You should first learn the 4 basic triangles, then you can form any figure by them.

1)
```
x
x x
x x x
```

- The outer loop normally is used for number of rows. We iterate the loop only the times we need rows. But e.g user enters 3 rows we have 2 options either we can go from $1 \to 3$ or $3 \to 1$, it is the same number of iteration.

So which is best?

Now it depends on inner loop.. e.g if want
```
x
x x
x x x
```
we will use 1 as start as inner loop depends upon outer but if write
```
x x x
x x
x
```
, then we start from 3.

Day: Date: / /

```
int main(){
    int rows;
    cout << "Rows: ";
    cin>>rows;
    int i=1;
    while ( i <= rows){

        int j = 1;
        while ( j <=i ){
            cout << "x";
            j++;
        }
        cout<<endl;
        i++;
    }
}
```

Output:

rows = 3

x

x x

x x x

Dry run:

	i	j	output
1st iteration	1	1	x
2nd iteration	2	1	
		2	x x
3rd iteration	3	1	
		2	
		3	x x x

Date: / /

2) x x x
   x x
   x

This is same as (1) but it is reflection of (1) on x-axis.

If a two patterns are symmetric about x-axis, the only difference in code is the outer loop condition.
e.g in (2) we will change i = rows and make i > 0 and i--; Nothing else will changed.

when we start from i = 0 then we run it < limit if we start from i = 1 then <= limit

Day:

Date: / /

3)

```
  x
 x x
x x x
```

Before making any pattern, see everything that is happening:

```
  - - x
  - x x
  x x x
```

These are spaces.

Now for this we will have one outer loop and two inner loops, one for spaces and one for △.

now look at spaces:

They are decreasing and starting with one less than rows so spaces = rows - 1.

In this question we will require to different variables as we need one for decreasing spaces and another for increasing "x".

code next page

Prog: Date: / /

```
int main() {

    int rows;
    cout << "Rows: ";
    cin >> rows;

    int spaces = rows - 1;   for spaces as it went
    int i = 1;   for "x" as it went
    while ( i <= rows) {

        int j = spaces;
        while ( j > 0 ) {
            cout << " ";
            j--;
        }

        spaces --;   // decreasing spaces variable.

        j = 1
        while (j <= i) {
            cout << "x";
            j++;
        }

        i++;
        cout << endl;
}
```

Output

```
Rows: 5
    x
   x x
  x x x
 x x x x
x x x x x
```

Day: Date: / /

(4)
```
x x x x x
 x x x x
  x x x
   x x
    x
```

This is the symmetric for the (3) so we only change the outer loop.

we will change outer loop conditions only BUT outer loop has 2 condition in reality.

- one was for spaces
- second was for " x "

So both will changed. ie

```
spaces = 0;
i = rows;
i --;
spaces ++;
```

[illegible]

Page: Date: / /

for loop:

for (initializing or assigning. ; condition ; changing the value)

for loops make while easier as they include (int i; i<0 ; i++) in one line instead of 3 for while.

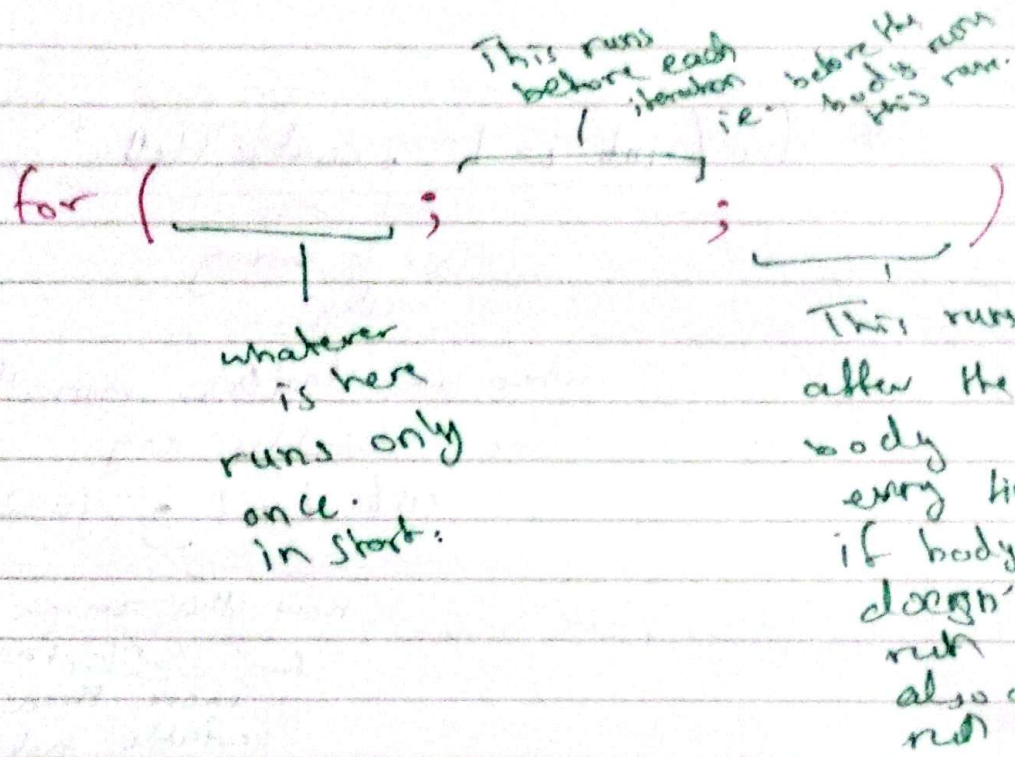
e.g:

```
int i=1
while (i<rows){
    i
    i++;
}
```

```
for (int i=1; i<rows; i++){
    body.
}
```

How does for work?:

Day: Date: / /



Fun facts:

1) Initialization place for (This place ; ;)

- anything can be written here even cout << "Hello world". It will be run once.
- you can have multiple statements seperated by a , (comma). e.g.:

for(cout<<"Hi", i=1; ;)

- But if we are declaring something e.g int i, we cannot have multiple statements, we can declare multiple variables but not write many statements. Why?

Page

Date / /

e.g for (int i=1 , cout << "Hello" ; ;)

This is wrong
why?
when we declare more than one variables e.g
int i=1, j=2;

now this comma is for declaring more than one variable not to seperate statements so as cout cannot be a variable it can't be declared. so thats why.

for (cout << "Hi" , int i=1 ; ;) ✗
Not allowed.

int i=0;
for (cout << "Hi" , i=3 ; ;) ✓ allowed.

Day: Date: / /

2) condition place: for (; <u>condition</u> ,)
- ~~X~~ declaring or initializing is allowed (same rules)
- You can write more than one statement.
- Can be anything but always remember to write the true condition at last.

e.g ~~for(cout<<"Hi";i<6;)~~

1) for (; cout<<"Hi", <u>i<6</u> ;)

This will be checked by for a a condition

2) for (; i<6 , <u>cout<<"Hi"</u> ;)

This is a condition now but as this is not a condition loop will never end. it will print Hi also every time.

3) for(; <u>i<6</u> , <u>j<5</u> , <u>k>0</u> ;)

These will be run as normal code NOT a condition

This is the only condition for checks.

Yousaf

3) increment/dec place: for(; ;)

- Again anything can be written
- Declaring or initializing NOT Allowed.
  assigning allowed.

anything can be written

eg

for(; ; cout << "Hi", i=7, i++, j=7+i)

Do - while loops.

do {

body.

} while ( );