Control Structure (Repetition)

DM2111
C++ Programming

Introduction

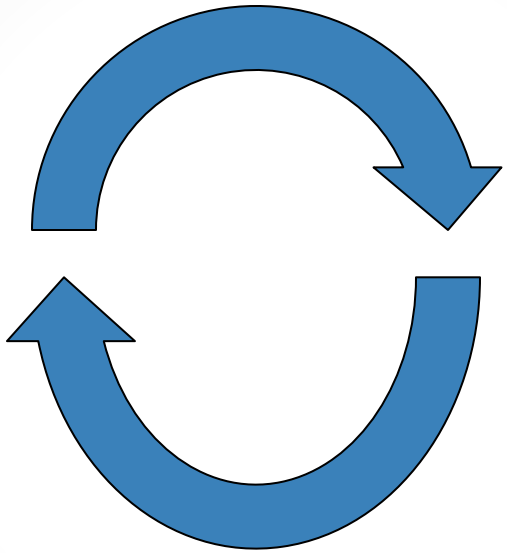
Introduction	Array and Strings
Problem solving	Array and Strings
Basic elements of C++	Pointers
Basic elements of C++	Pointers
Statements	I/O operations
Repetition	Structs
Functions	Others
Functions	

Agenda

- while
- do… while
- for
- break
- continue

Repetition

- Meaning: The act of repeating something over and over again.



- Computer is good at repeating tasks!!!


IN·SAN·I·TY
/IN'SANITē/:
DOING THE EXACT
SAME THING OVER AND
OVER AGAIN, EXPECTING
THINGS TO CHANGE.
THAT. IS. CRAZY.

while loop

```
while (expression)
   statement
```

Statement will run repeatedly while expression is true

while compared to if

- Similar to `if`, except <statement> will execute repeatedly until <condition> evaluates to false

```
if (<condition>)
{
    <statement>
}
```

vs

```
while (<condition>)
{
    <statement>
}
```

```
int k = 0;

if (k < 5)
{
    cout << "*";
}
```

```
int k = 0;

while (k < 5)
{
    cout << "*";
}
```

while

```
int k = 0;

while (k < 5)
{
    cout << "*";
}
```

```
k = 0   ->   *
k = 0   ->   *
k = 0   ->   *
...
```

Will the expression ever evaluate to false?

How can we solve this problem?

```
int k = 0;

while (k < 5) {
    cout << "*";
    ++k;
}
```

```
k = 0  ->  *
k = 1  ->  *
k = 2  ->  *
k = 3  ->  *
k = 4  ->  *
k = 5  ->
```

What if we initialise k = 6?

Loop controlled variable in while

- Must be initialised before loop entry
- Variable is tested at every iteration
- Variable must be updated such that eventually <condition> would evaluate to false

```
int k = 0;              // initialise

while (k < 5) {         // test
    cout << "*";
    ++k;                // update
}
```

Flag controlled while loop

```
bool run = true;

while (run) {
    ...

    if (<exit loop condition>)
        run = false;
}
```

while example

```
int noOfGuesses = 0, guess;
bool done = false;

while (noOfGuesses < 5 && !done)
{
    cin >> guess;

    if (guess == 1234)
        done = true;
    else
        ++noOfGuesses;
}
```

A programmer heads out to the store. His wife says "while you're out, get some milk."

He never came home.

```
while (out)
{
   getMilk();
}
```

do… while loop

```
do
    statement
while (condition);
```

Do first, test later

statement will run at least once in a `do...while` loop

a `do while` loop ends with a semicolon

do… while compared with while

```
do {
    <statement>
} while (<condition>);
```

vs

```
while (<condition>) {
    <statement>
}
```

Variables used in *condition* must be declared outside of `do while`.

do… while in false condition

True condition

```
int k = 0;

while (k < 5) {
    cout << "*";
    ++k;
}
```

vs

```
int k = 0;

do {
    cout << "*";
    ++k;
} while (k < 5);
```

False condition

```
int k = 6;

while (k < 5) {
    cout << "*";
    ++k;
}
```

vs

```
int k = 6;

do {
    cout << "*";
    ++k;
} while (k < 5);
```

for loop

```
for (initializer; condition; update)
    statement
```

initializer - Initialize or assign a starting value

condition - loop control, evaluated at end of every loop

update - usually modifies the variables in initializer and tested in condition

statement - body of your loop

for and while loop comparison

```
for (<init>; <test>; <update>)
{
    <stmt>
}
```

```
int k = 0;              // initialise

while (k < 5)           // test
{
    cout << "*";
    ++k;               // update
}
```

converting while loop to for loop

```
int k = 0;              // initialise

while (k < 5)           // test
{
    cout << "*";
    ++k;               // update
}
```

```
int k;

for (k = 0; k < 5; ++k)
{
    cout << "*";
}
```

```
for (int k = 0; k < 5; ++k)
{
    cout << "*";
}
```

Omitting parts in a for loop

```
int a, b;

for (a = 2, b = 1; a > b; ++a, b *= 2)
{
    cout << "*";
}
```

```
for (int k = 0; k < 5; )
{
    cout << "*";
}
```

```
for (;;)
{
    cout << "*";
}
```

- If *condition* is omitted, it is assumed true.

Nested Loops

```
int i = 0;

while (i < 5)
{
    int j = 0;

    while (j < 5)
    {
        cout << "*";
        ++j;
    }

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

Nested while loops replaced by for loops

```
int i = 0;

while (i < 5)
{
    int j = 0;

    while (j < 5)
    {
        cout << "*";
        ++j;
    }

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

```
for (int i = 0; i < 5; ++i)
{
    for (int j = 0; j < 5; ++j)
    {
        cout << "*";
    }

    cout << endl;
}
```

break

How do you get out of a loop?

```
int noOfGuesses = 0, guess;
bool done = false;

while (noOfGuesses < 5 && !done)
{
    cin >> guess;

    if (guess == 1234)
        done = true;
    else
        ++noOfGuesses;
}
```

break terminates the nearest enclosing loop

```
int noOfGuesses = 0, guess;
bool done = false;

while (noOfGuesses < 5 && !done)
{
    cin >> guess;

    if (guess == 1234)
        done = true;
        break;
    else
        ++noOfGuesses;
}
// break transfers control to here
```

continue

How about if you want to “skip” this loop?

Consider a code processes only odd numbers < 10.

```
//coding level < 7!!!
for (int i = 0; i < 10; ++i)
{
    if (i % 2 != 0)
    {
        cout << i;
        // a bunch of other code
    }
}
```

continue

Can we skip over the even numbers?

```
//coding level < 8!!!
for (int i = 0; i < 10; ++i)
{
    if (i % 2 == 0) // note condition
    {
        continue;
    }
    cout << i;
    // a bunch of other code
}
```

break vs continue

`break`

Terminates the enclosing loop

`continue`

Terminates the current iteration

Use on `while`, `do while`, `for` or `switch` statements.


GAME OVER
INSERT COINS
TO CONTINUE