# Introduction to Programming

CSC1102 &1103

Lecture-4

American International University Bangladesh

Dept. of Computer Science

Faculty of Science and Information Technology

## Lecture 3: Outline

- Program Looping

The `for` Statement

  - Relational Operators
  - Nested `for` Loops
  - Increment Operator
  - Program Input
  - `for` Loop Variants
  - The `while` Statement
  - The `do` Statement
  - The `break` Statement
  - The `continue` Statement

# Executing a program

- Program = list of statements
  - ***Entrypoint***: the point where the execution starts
  - ***Control flow***: the order in which the individual statements are executed

```
Statement1
Statement2
Statement3
Statement4
Statement5
Statement6
Statement7
Statement8
```

# Structure of a C program

Entry point of a C program

```
int main (void)
{
  int value1, value2, sum;
  value1 = 50;
  value2 = 25;
  sum = value1 + value2;
  cout<<"The sum = " <<sum<<endl;
  return 0;
}
```

Sequential flow of control

## Controlling the program flow

- Forms of controlling the program flow:
  - Executing a sequence of statements
  - Repeating a sequence of statements (until some condition is met) (looping)
  - Using a test to decide between alternative sequences (branching)

```
Statement1
Statement2
Statement3
Statement4
Statement5
Statement6
Statement7
Statement8
```

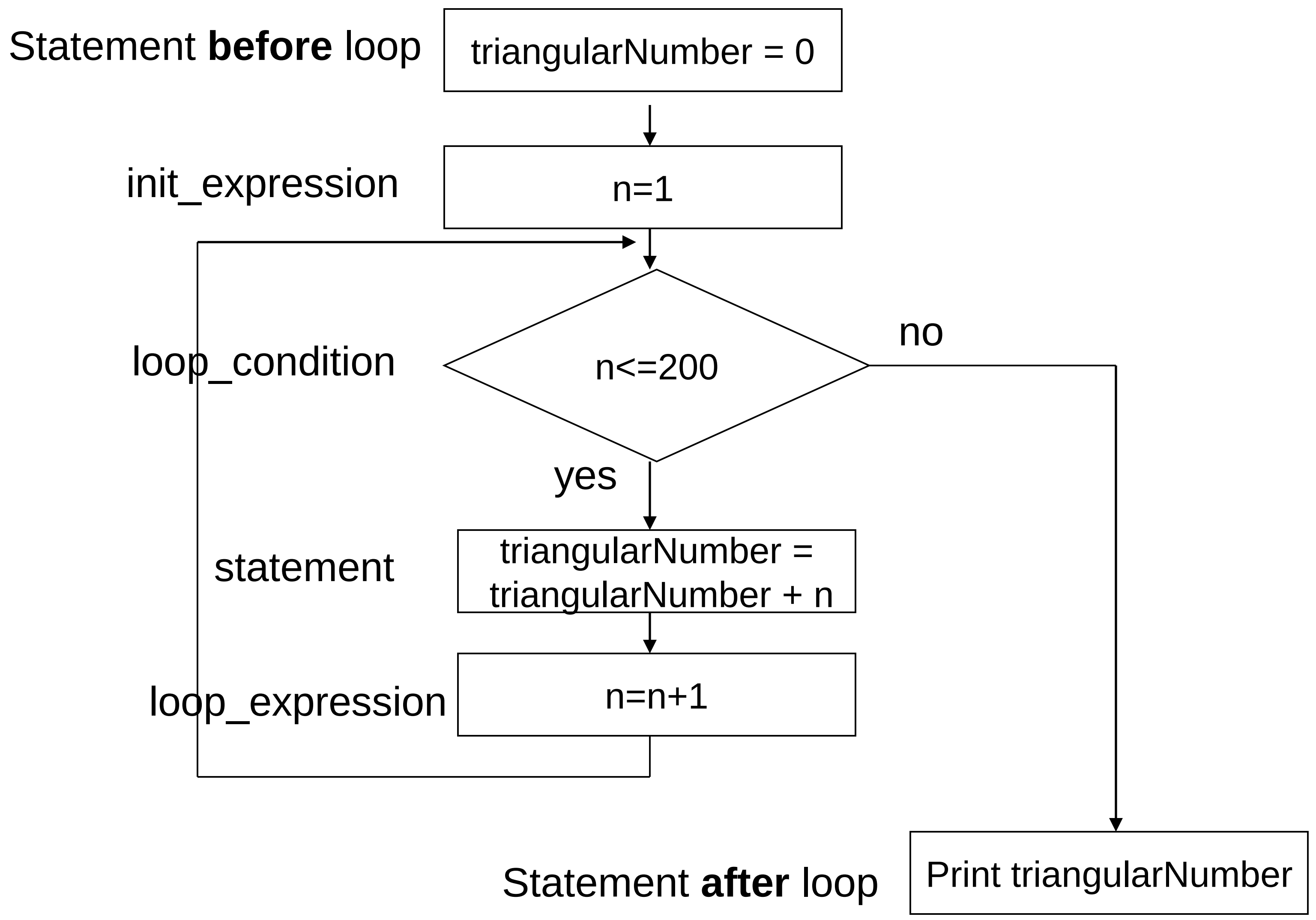
# Program Looping

- Looping: doing one thing over and over
- Program loop: a set of statements that is executed repetitively for a number of times

- Simple example: displaying a message 100 times:

*Program looping:* enables you to develop concise programs containing repetitive processes that could otherwise require many lines of code !

# Example – 200th triangular number

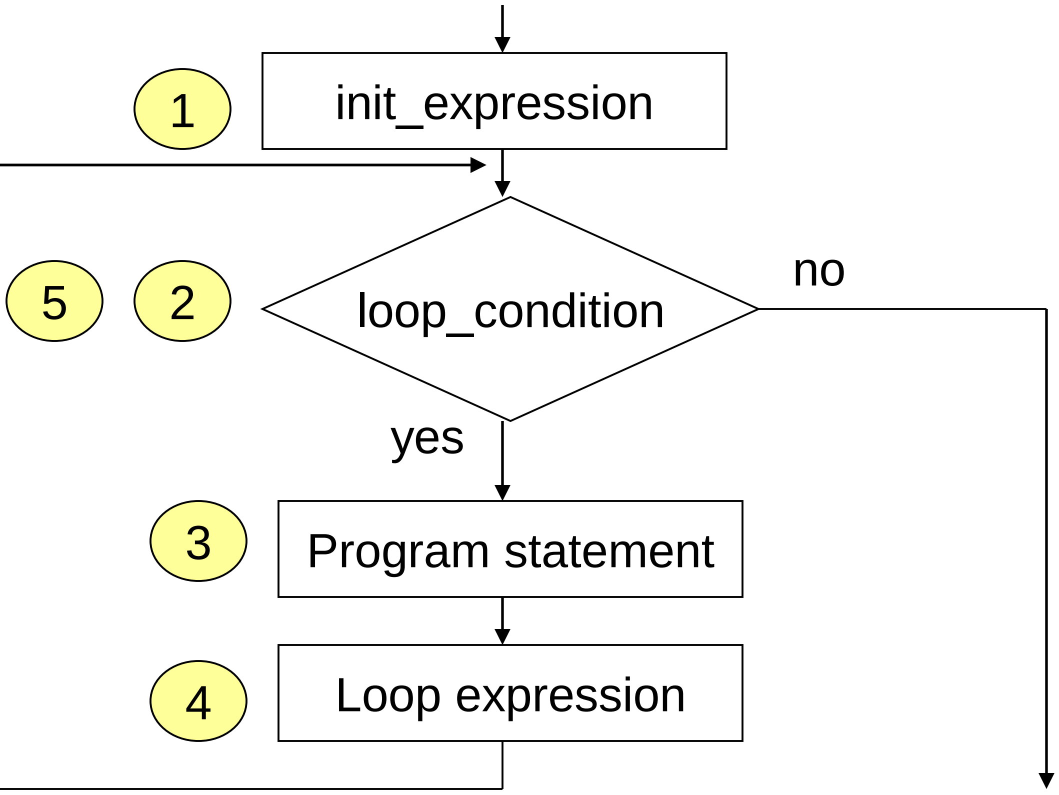


## Example - `for`

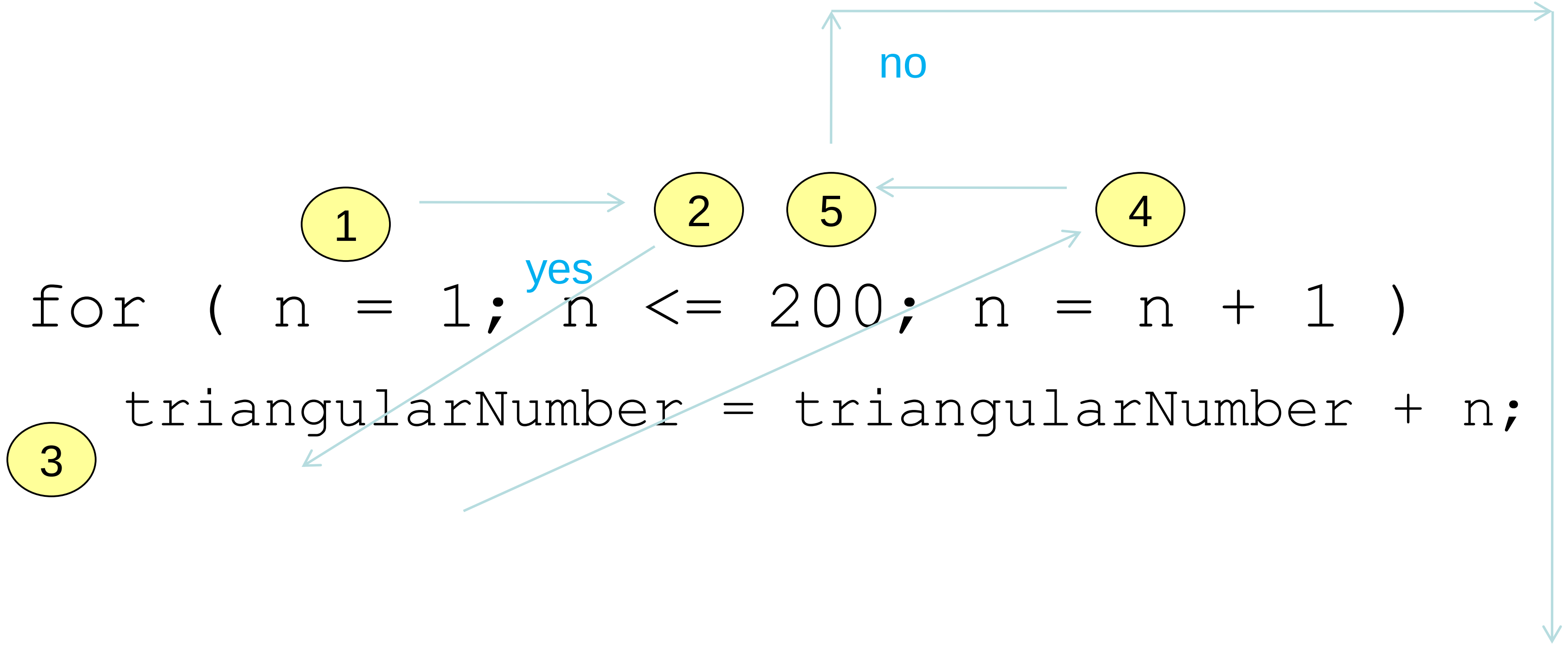
```
/* Program to calculate the 200th triangular number
Introduction of the for statement */

#include<iostream>
using namespace std;
int main (void)
{
   int n, triangularNumber;
   triangularNumber = 0;
   for ( n = 1; n <= 200; n = n + 1 )
      triangularNumber = triangularNumber + n;
   cout<<"The 200th triangular number = ">>triangularNumber;
   return 0;
}
```

# The `for` statement

for ( *init_expression; loop_condition; loop_expression* )
*program statement*



# The `for` statement



```
for ( n = 1; n <= 200; n = n + 1 )
    triangularNumber = triangularNumber + n;
```

## How `for` works

- The execution of a for statement proceeds as follows:
    1. The initial expression is evaluated first. This expression usually sets a variable that will be used inside the loop, generally referred to as an *index* variable, to some initial value.
    2. The looping condition is evaluated. If the condition is not satisfied (the expression is false – has value 0), the loop is immediately terminated. Execution continues with the program statement that immediately follows the loop.
    3. The program statement that constitutes the body of the loop is executed.
    4. The looping expression is evaluated. This expression is generally used to change  the value of the index variable
    5. Return to step 2.

## Infinite loops

- It’s the task of the programmer to design correctly the algorithms so that loops end at some moment !

```
// Program to count 1+2+3+4+5
int main (void)
{
    int  i, n = 5, sum =0;
    for ( i = 1; i <= n; n = n + 1 ){
      sum = sum + i;
      cout<<sum<<endl;
    }
    return 0;
}
```

**What is wrong here ?**

**Does the loop end?**

# Relational operators

| Operator | Meaning |
|---|---|
| == | Is equal to |
| != | Is not equal to |
| < | Is less than |
| <= | Is less or equal |
| > | Is greater than |
| >= | Is greater or equal |

The relational operators have lower precedence than all arithmetic operators:
`a < b + c` is evaluated as `a < (b + c)`

ATTENTION ! Do not confuse:
the "is equal to" operator == and the "assignment" operator =

ATTENTION when comparing floating-point values !
Only < and > comparisons make sense !

## Increment operator

- Because addition by 1 is a very common operation in programs, a special operator was created in C for this.
- *Increment operator*: the expression ++n is equivalent to the expression n = n + 1.
- *Decrement operator*: the expression --n is equivalent to the expression n = n – 1
- Increment and decrement operators can be placed in front (*prefix*) or after (postfix) their operand.
- The difference between prefix and postfix:
- Example: if n=4:
    - a=n++  leads to a=4, n=5
    - a=++n leads to a=5, n=5

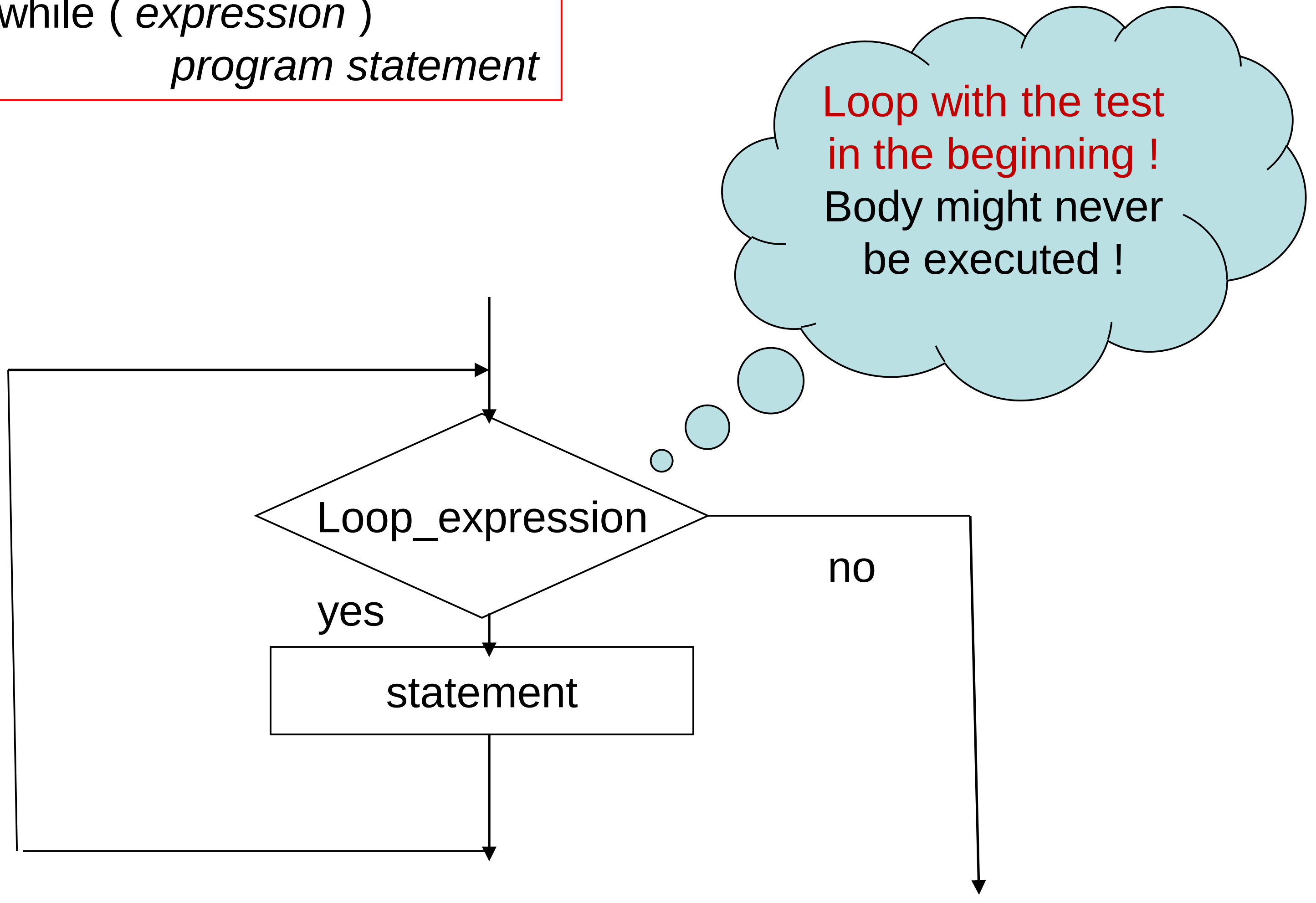
# The `while` statement

while ( *expression* )
    *program statement*

```
while ( number <= 0 )
{
cout<<"The number must be >0"<<endl;
cout<<"Give a new number:  "<<endl;
cin>>number;
}
```

# The `while` statement

```
while ( expression )
        program statement
```



## Example:

- A program to find the greatest common divisor of two nonnegative integer values …

# Example - `while`

```
/* Program to find the greatest common divisor
of two nonnegative integer values */
#include <iostream>
using namespace std;
int main (void)
{
int u, v, temp;
cout<<"Please type in two nonnegative integers."<<endl;
cin>>u>>v;
while ( v != 0 ) {
temp = u % v;
u = v;
v = temp;
}
cout<<"Their greatest common divisor is "<< u;
return 0;
}
```

# Example:

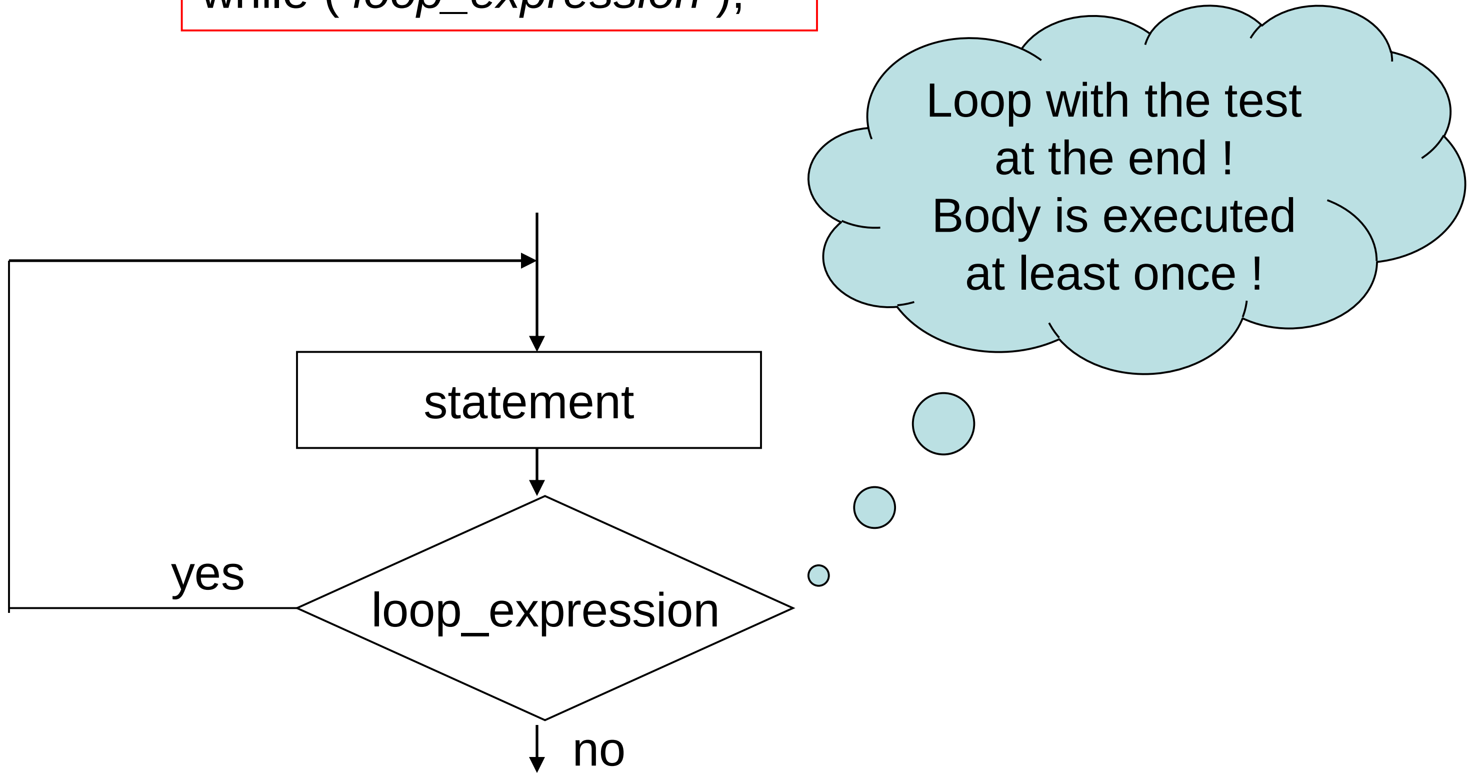
- A program to print out the digits of a number in reverse order …

## Example - `while`

```
// Program to reverse the digits of a number
#include <iostream>
using namespace std;
int main (void)
{
        int number, right_digit;
        cout<<"Enter your number"<<endl;
        cin>>number;
while ( number != 0 )
{
        right_digit = number % 10;
        cout<< right_digit;
        number = number / 10;
}
return 0;
}
```

# The `do` statement

```
do
            program statement
while ( loop_expression );
```


Loop with the test at the end ! Body is executed at least once !
statement
yes
loop_expression
no

## Which loop to choose ?

- Criteria: Who determines looping
  - Entry-condition loop -> for, while
  - Exit-condition loop -> do
- Criteria: Number of repetitions:
  - Indefinite loops ->while
  - Counting loops -> for
- You can actually rewrite any while as a for and viceversa  !

# The `break` Statement

- Can be used in order to immediately exiting from a loop
- After a break, following statements in the loop body are skipped and execution continues with the first statement after the loop
- If a break is executed from within nested loops, only the innermost loop is terminated

# The `break` statement

- Programming style: don’t abuse break !!!

```
...
while ( number != 0 ) {
   // Statements to do something in loop
   cout<<"Stop, answer 1:” <<endl;
   cin>>answer;
   if(answer == 1)
       break; // very bad idea to do this
}
```

## The `continue` statement

- Similar to the break statement, but it does not make the loop terminate, just skips to the next iteration

# The `continue` statement

Continue also not so good style!!!

```
...
while ( number != 0 ) {
    // Statements to do something in loop
    cout<<"Skip next statements answer 1: "<<endl;
    cin>>answer);
    if(answer == 1)
       continue; // not so good idea…
    // Statements to do something in loop
    // If answer was 1 these statements are
    // not executed. They are skipped.
    // Go straight to the beginning of while
}
```