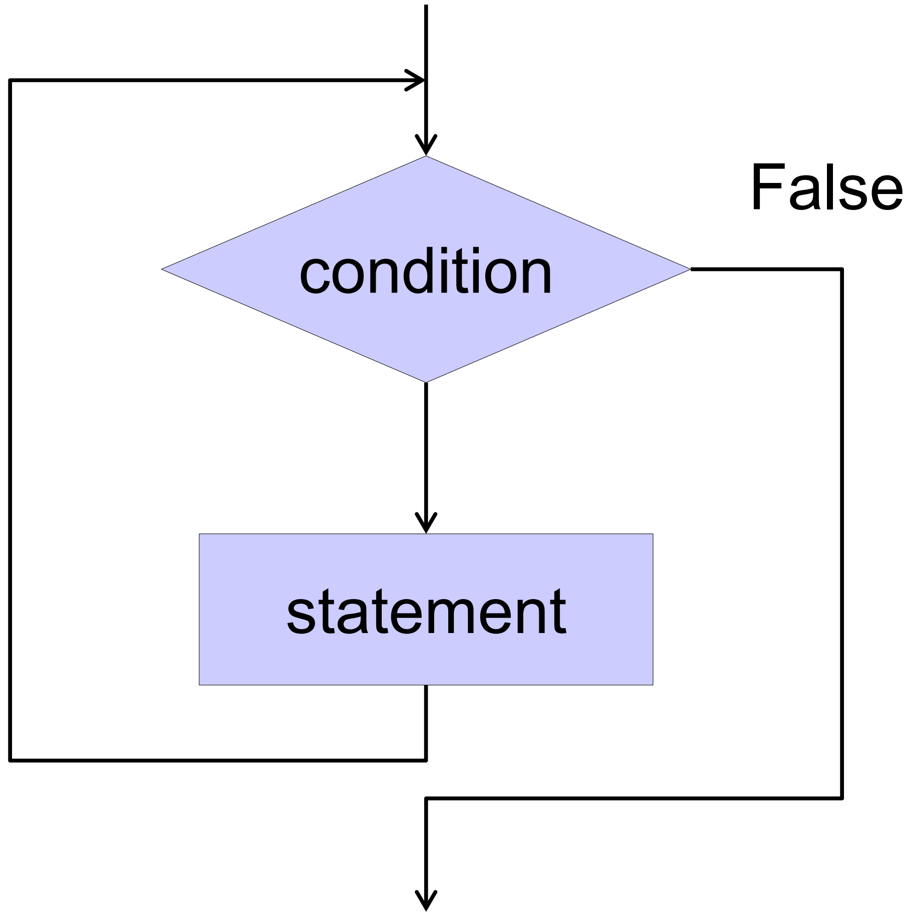
Iterations

Iteration

- The repetition of a statement or block of statements in a program.
- Also called ***loops*** because of their cycle nature.
- Used in many programs, especially, when we need to process arrays.
- Three iteration statements in C/C++
  - The while statement,
  - The do ... while statement
  - The for statement.

The while statement



Syntax

while (condition) statement;

(cont'd.)

- If the condition is non-zero (“true”) the statement is executed **repeatedly** until the condition evaluates to zero.
- If the condition is zero (“false”) then the statement is ignored and program execution jumps immediately to the next statement.

Displaying “Hello World!”

Writing a C++ program to

Display 10 “Hello World!”

Display 100 “Hello World!”

Display *n* “Hello World!”

Displaying 1 to ...

Writing a C++ program to

Display all integer numbers from 1 to 10

Display all integer numbers from 1 to 100

Display all integer numbers from 1 to n

A Sum of Consecutive Integers

```
int main() {
        int n, i = 1;
        cout << "Enter a positive integer: ";
        cin >> n;
        long sum = 0;
        while (i <= n)
                sum = sum + i++;      //one statement
        cout << "The sum of the first " << n
            << " integers is " << sum << endl;
}
```

A Sum of Consecutive Integers

```
int main() {
        int n, i = 1;
        cout << "Enter a positive integer: ";
        cin >> n;
        long sum = 0;
        while (i <= n) {   //statement block
                i++;
                sum = sum + i;
                cout << “sum to “ << i << “is“ << sum << endl;
        }
        cout << "The sum of the first " << n
            << " integers is " << sum << endl;
}
```

A Sum of Reciprocals

```
int main() {
        int bound;
        cout << "Enter a positive integer: ";
        cin >> bound;
        double sum = 0.0;
        int i = 0;
        while (sum < bound) {
                sum = sum +  1.0/++i;
        }
        cout << "The sum of the first " << i
            << " reciprocals is " << sum << endl;
}
```

Repeating a computation

```
int main() {
        double x;
        cout << "Enter a positive number: ";
        cin >> x;
        while (x > 0) {
                cout << "Square root (" << x << ") = "
                    << sqrt(x) << endl;
                cout << "Enter another positive number "
                    << "(or 0 to quit): ";
                cin >> x;
        }
}
```

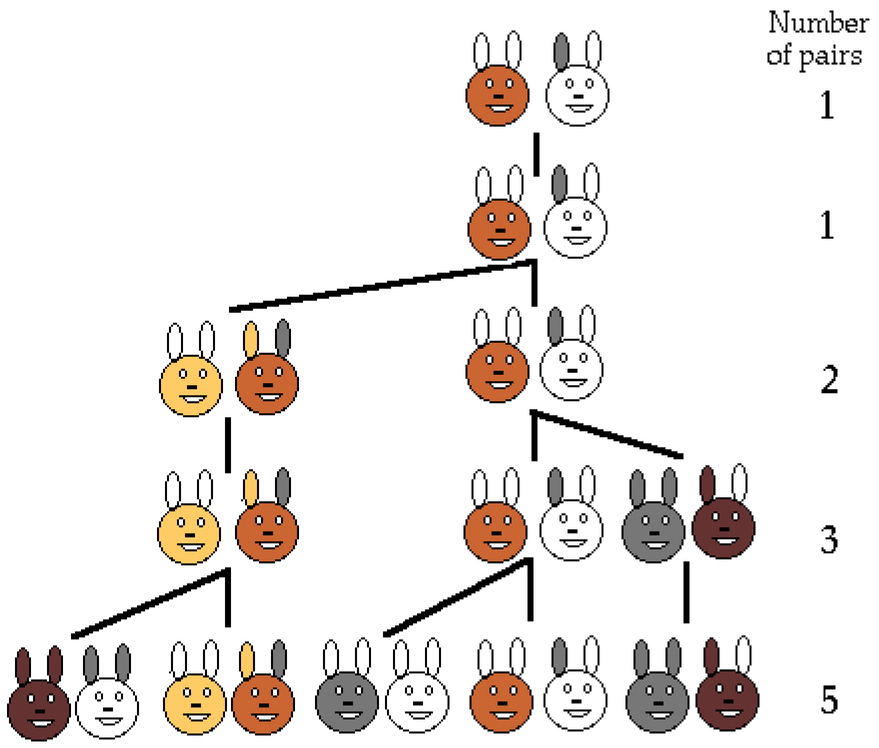
Fibonacci's Rabbits

The original problem that Fibonacci investigated (in the year 1202) was about how fast rabbits could breed in ideal circumstances.

Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits. Suppose that our rabbits never die and that the female always produces one new pair (one male, one female) every month from the second month on. The puzzle that Fibonacci posed was...

How many pairs will there be in one year?

Fibonacci's Rabbits



1. At the end of the first month, they mate, but there is still one only 1 pair.
2. At the end of the second month the female produces a new pair, so now there are 2 pairs of rabbits in the field.
3. At the end of the third month, the original female produces a second pair, making 3 pairs in all in the field.
4.At the end of the fourth month, the original female has produced yet another new pair, the female born two months ago produces her first pair also, making 5 pairs

The number of pairs of rabbits in the field at the start of each month is 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibnat.html#Rabbits

Exercise #1

Print the first *n* Fibonacci numbers

Fibonacci number is defined as the following

$$f_1 = 1$$
$$f_2 = 1$$
$$f_3 = f_1 + f_2 = 1 + 1 = 2$$
$$f_4 = f_2 + f_3 = 1 + 2 = 3$$
$$f_5 = f_3 + f_4 = 2 + 3 = 5$$
$$\ldots$$
$$f_n = f_{n\text{-}2} + f_{n\text{-}1}$$

loop compute new value
from old and older value

The Fibonacci Numbers

```
int main() {
        int n;
        cout << "Enter a positive integer: ";
        cin >> n;
        cout    << "First " << n << " Fib. numbers: ";
        if (n >= 1) cout << "1";
        if (n >= 2) cout << ", 1";
        long f_n, f_n_2 = 1, f_n_1 = 1;
        int i = 3;
        while (i <= n) {
                f_n = f_n_1 + f_n_2;            // new value of fib
                cout << ", " << f_n;            // print out
                f_n_2 = f_n_1;                  // old value become older
                f_n_1 = f_n;                    // new value become old one
                i++;                            // counting number
        }
        // go bake on looping
}
```

The Fibonacci Numbers

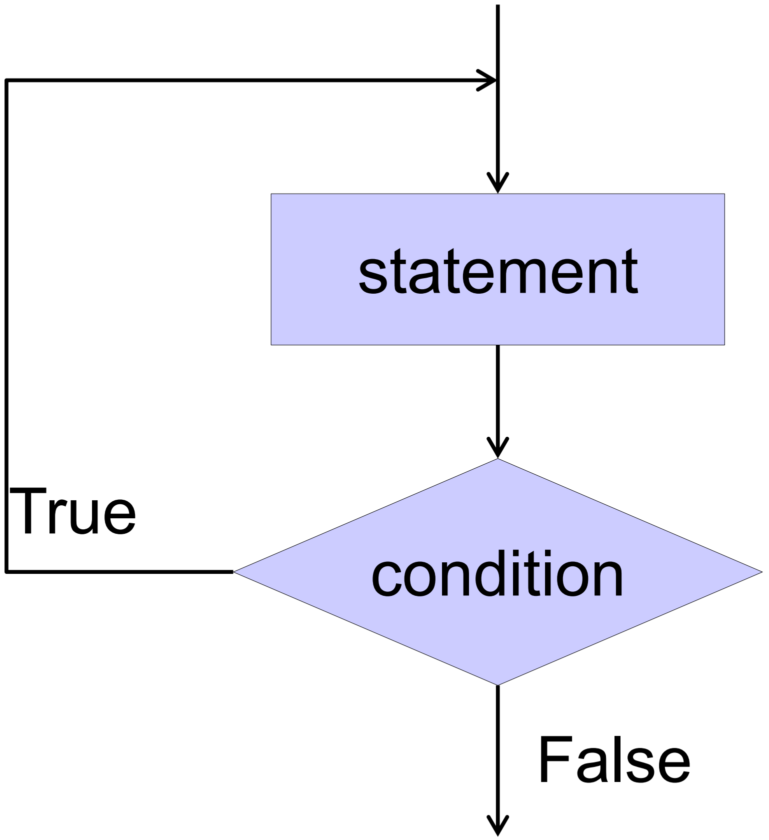
```
while (i <= n) {
        f_n = f_n_1 + f_n_2;
        cout << ", " << f_n;
        f_n_2 = f_n_1;

        f_n_1 = f_n;

        i++;
}
```

```
f_n = f_n_1 + f_n_2;        2 = 1 + 1
cout << ", " << f_n;
f_n_2 = f_n_1;              = 1
f_n_1 = f_n;                = 2
i++;

f_n = f_n_1 + f_n_2;        3 = 2 + 1
cout << ", " << f_n;
f_n_2 = f_n_1;              = 2
f_n_1 = f_n;                = 3
i++;

f_n = f_n_1 + f_n_2;        5 = 3 + 2
cout << ", " << f_n;
f_n_2 = f_n_1;              = 3
f_n_1 = f_n;                = 5
i++;
```

Terminating a Loop

```
int main() {
        int n, i = 1;
        cout << "Enter a positive integer: ";
        cin >> n;
        long sum = 0;
        while (true) {
                if (i > n) break;
                sum = sum + i;
                cout << i << “ : “ <<sum << “ “ << endl;
                i++;
        }
        cout << "The sum of the first "
        << n << " integers is " << sum << endl;
}
```

The do...while statement



Syntax

do statement while (condition);

(cont'd.)

- The statement is executed repeatedly until the condition evaluates to zero (“false”).
- This is the same as while except that the condition is tested at the end of the loop instead of the beginning.

Comparison

```
 cout << “begin” << endl;

while(false) {
  cout << “in loop” << endl;
}



cout << “end” << endl;


---------------
output

begin
end
```

```
cout << “begin” << endl;


do {
  cout << “in loop” << endl;
} while(false);


cout << “end” << endl;


---------------
output


begin
inloop
end
```

Comparison

```
int i = 0;
cout << "begin" << endl;


while(i < 10) {
  cout << i << " " << endl;
  i++;
}

cout << "end " << i << endl;
```

----

output

```
 begin
 0 1 2 3 4 5 6 7 8 9 end 10
```

```
int i = 0;
cout << "begin" << endl;

do {
  cout << i << " " << endl;
  i++;
} while(i < 10);

cout << "end " << i << endl;
```

----

output

```
begin
0 1 2 3 4 5 6 7 8 9 end 10
```

A Sum of Consecutive Integers .. again

```
int main() {
        int n, i = 0;
        cout << "Enter a positive integer: ";
        cin >> n;
        long sum = 0;
        do {
                sum = sum + i++;
        } while (i <= n);
        cout << "The sum of the first " << n
            << " integers is " << sum << endl;
}
```

Exercise #3

Write a program to print the first *n* factorial numbers using do ... while.

Also try this exercise using while.

The Factorial Numbers

```
int main() {
        int n;
        cout << "Enter a positive integer: ";
        cin >> n;
        cout << "Factorial number : ";
        long f = 1, i = 1;
        do {
                cout << f << " ";
    i++;
                f =     f * i;
        } while (i <= n);
        cout << endl;
}
```

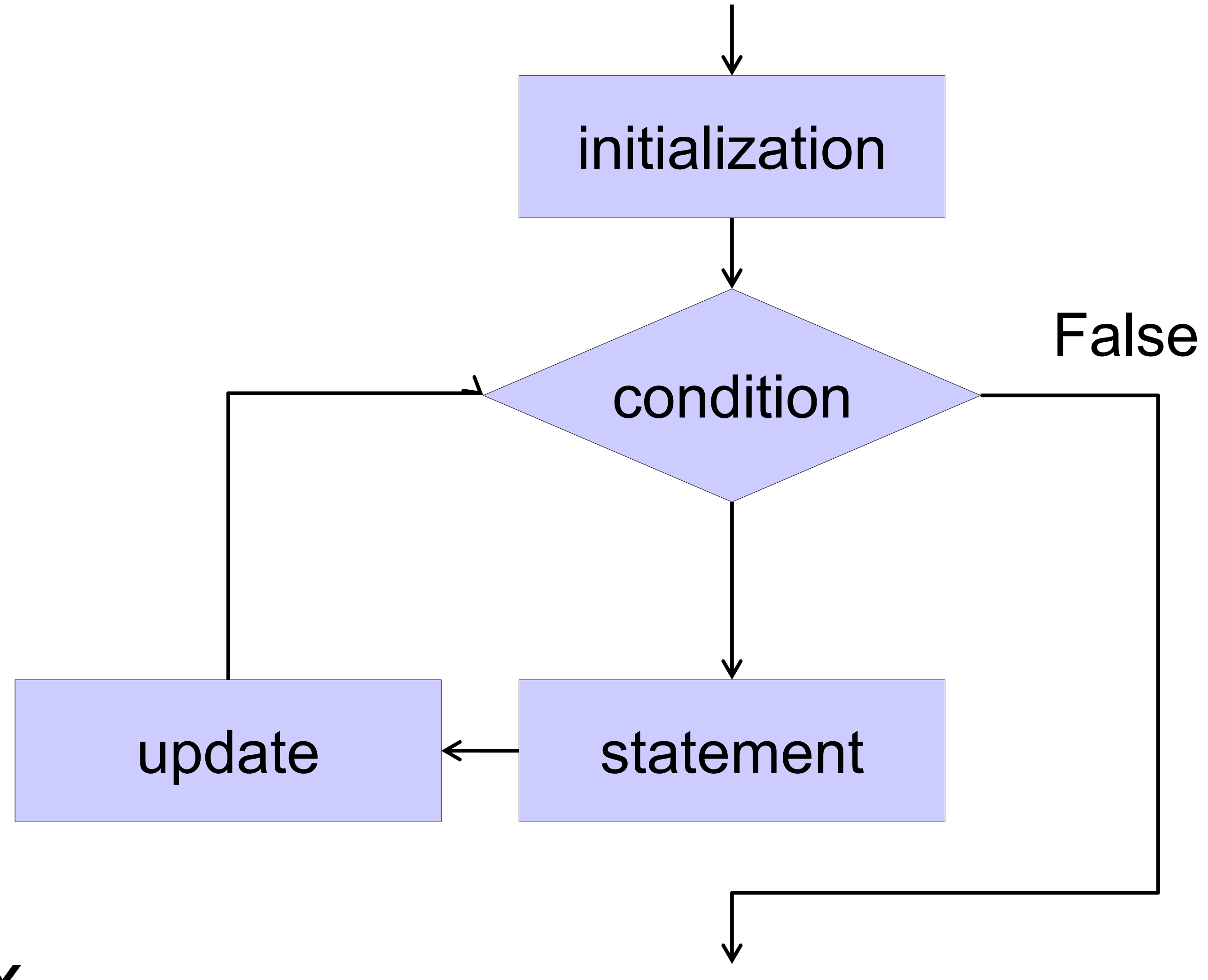
Exercise #4

Sequentially print all the factorial number that are less than or equal to a given bound value using do ... while.

The Factorial Numbers

```
int main() {
        int bound;
        cout << "Enter a positive integer: ";
        cin >> bound;
        cout << "Factorial number < " << bound
            << " : 1";
        long f = 1, i = 1;
        do {
                cout << ", " << f;
    i++;
                f = f * i;
        } while (f < bound);
        cout << endl;
}
```

The for statement



Syntax

for (initialization; condition; update) statement;

(cont'd.)

1. Evaluate initialization, which is used to declare and/or initialize control variable(s) for the loop.
2. The condition is evaluated, before iteration occur.
3. Execute the statement.
4. Evaluate the update.
5. Repeat step 2 – 4.

A Sum of Consecutive Integers – for

```
int main() {
        int n;
        cout << "Enter a positive integer: ";
        cin >> n;
        long sum = 0;
        for (int i = 0; i <= n; i++)
                sum = sum + i;
        cout << "The sum of the first " << n
            << " integers is " << sum << endl;
}
```

Comparison

```
for (int i = 0; i<=10; i++)
    statement;



int i = 0;
while( i <= 10 ) {
    statement;
    i++;
}
```

Reusing for Loop Control Variable

```
int main() {
        int n;
        cout << "Enter a positive integer: ";
        cin >> n;
        long sum = 0;
        for (int i = 0; i < n/2; i++)
                sum += i;
        for (int i = n/2; i <= n ; i++)
                sum += i;
        cout << "The sum of the first " << n
            << " integers is " << sum << endl;
}
```

Loop variable

```
int main() {

    int i = 10;
    cout << “begin “ << i << endl;

    for (int i = 0; i <= 20; i++)
                     cout << i;


    cout << endl;
    cout << "end loop " << i;

}
```

```
int main() {

    int i = 10;
    cout << “begin “ << i << endl;

    for (i = 0; i <= 20; i++)
                     cout << i;


    cout << endl;
    cout << "end loop " << i;

}
```

Exercise #5

Try the factorial question using for.

- Print the first *n* factorial.
- Sequentially print the factorial number that is less than a given bound value.

The Factorial Numbers .. again

```
int main() {
        long n;
        cout << "Enter a positive integer: ";
        cin >> n;
        long f = 1;
        cout << "Factorial number : " << f;
        for (int i = 1; i <= n; i++) {
          f = f * i;
          cout << ", " << f;
        }
        cout << endl;
}
```

The Factorial Numbers .. and again

```
int main() {
        long bound;
        cout << "Enter a positive integer: ";
        cin >> bound;
        cout << "Factorial number < " << bound
            << ":\n1";
        long f = 1;
        for (int i = 2; f < bound; i++) {
                cout << ", " << f;
                f = f * i;
        }
        cout << endl;
}
```

Exercise #6

Can we try to do a descent for loop ?

Let's try, just to print the number from *n* to 1.

Descending for Loop

```
int main() {
        int n;
        cout << "Enter a positive integer: ";
        cin >> n;
        for (int i = n; i > 0; i--)
                cout << " " << i;
}
```

Prime Numbers

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself.

Write a program to test if a given number is a prime number or not

Prime Numbers

```
int main() {
        long n;
        cout << "Enter a positive integer: ";
        cin >> n;
        bool is_prime = true;
 if ( n >= 2 ) {                      // 2 is prime need to test here
   for( int i = 2; i < n ; i++) {     // because
      if ( n % i == 0) {              // this statement fail to check 2
        is_prime = false;              // is prime
      }
    }
 }
 if ( is_prime )
   cout << n << " is prime";
 else
   cout << n << " is not prime";
}
```

Prime Numbers (cont'd.)

```
for( int i = 2; i < n ; i++) {
  if ( n % i == 0) {
    is_prime = false;
    break;
  }
}
```

No need to test other numbers because the number can be divided at just one, it is not prime.

Prime Numbers (cont'd.)

If the number is has divisor by any event numbers it is also has divisor by 2.

20 % 4 = 0 => 20 % 2 = 0 is also true.

Just test if the number is a even number then it is not prime, no need to check event divisors.

Prime Numbers (cont'd.)

```
int main() {
        long n;
        cout << "Enter a positive integer: ";
        cin >> n;
        bool is_prime = true;
  if ( n >= 3 ) {       // 2 and 3 is prime
   if ( n % 2 == 0 ) {
     is_prime = false;
   } else {
     for( int i = 3; i < n ; i += 2) {
        if ( n % i == 0) {
           is_prime = false;
           break;
        }
      }
    }
  }
 if ( is_prime ) cout << n << " is prime";
 else cout << n << " is not prime";
}
```

Prime Numbers (cont'd.)

It is no need to test from 2 to n-1 just testing from 2 to √n is enough, because √n x √n = n, if number larger than √n the result of multiplication will larger than n too.

```
..
long n;
..
cin >> n;
long limit = sqrt(n);
 ...
  for( int i = 3; i < limit ; i += 2)
 ...
```

Prime Numbers (cont'd.)

Using combine exit condition

```
int main() {
        long n;
        cout << "Enter a positive integer: ";
        cin >> n;
  long limit = sqrt(n);
        bool is_prime = true;

  if ( n >= 3 ) {
                if ( n % 2 == 0 ) is_prime = false;
        for( int i = 3; i < limit && is_prime ; i += 2)
        if ( n % i == 0) is_prime = false;
  }

  if ( is_prim ) {
    cout << n << " is prime";
  } else {
    cout << n << " is not prime";
  }
}
```

Exercise #7

Find the minimum from all input integers

Control Input with a Sentinel

```
int main() {
        int n, count = 0, sum = 0;
        cout << "Enter positive integers "
            << (0 to quit): " << endl;
        for (;;) {
                cout << "\t" << count + 1 << " : ";
                cin >> n;
                if (n <= 0) break;
                ++count;
                sum += n;
        }
        cout << "The average of those " << count
            << "positive numbers is
            << float(sum)/count << endl;
}
```

Nested for Loops

```
#include <iostream>
#include <iomanip>

int main() {
        for (int x = 1; x <= 12; x++) {
                for (int y = 1; y <= 12; y++)
                        cout << setw(4) << x * y;
                cout << endl;
        }
}
```

Nested for Loops

```
#include <iostream>
#include <iomanip>

int main() {
  cin << n;
          for (int x = 1; x <= n; x++) {
                    for (int y = 1; y <= n; y++) {
                              cout << '*';
     }
                    cout << endl;
          }
}
```

Nested for Loops

```
#include <iostream>
#include <iomanip>

int main() {
  cin << n;
          for (int x = 1; x <= n; x++) {
                    for (int y = 1; y <= n; y++) {
      if (x==1 || x==n || y ==1 ||y==n)
                                   cout << '*';
      else
          cout << ' ';
    }
                    cout << endl;
          }
}
```

A break with Nested Loops

```
int main() {
        for (int x = 1; x <= 12; x++) {
                for (int y = 1; y <= 12; y++){
                        if (y > x)
                                break;
                        else
                                cout << setw(4) << x * y;
    }
                cout << endl;
        }
}
```

The continue Statement

The break statement skips the rest of the statements in the loop's block, and jumps immediately to the next statement outside of the loop.

The continue statement is similar to the break statement It skips the rest of the statements in the loop's block, and transfers execution to the next iteration of the loop.

Using continue

```
int main() {
        for (int i = 0; i<10;i++) {
                cout << "Top half :” << i << endl;
                if (i > 5 ) continue;
                cout << "Bottom half:" << i << endl;
        }
        cout << "Outside of loop.";
}
```

try if(i > 5) break;

Using continue

```
int main() {
  int i = 0;
        while( i < 10) {
                cout << "Top half :” << i << endl;
                if (i > 5 ) continue;
                cout << "Bottom half:" << i << endl;
                i++;        // Aware of this
        }
        cout << "Outside of loop.";
}
```

Using a goto Statement

```
int main() {
        const int N = 5;
        for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                        for (int k = 0; k < N; k++)
                                if (i + j + k > N)
                                        goto esc;
                                else
                                        cout << i + j + k << " ";
                        cout << "* ";
                }
                esc: cout << "." << endl;
        }
}
```

Using a Flag to Break Out

```
int main() {
        const int N = 5;
        bool done = false; // 'done' is a flag
        for (int i = 0; i < N; i++) {
                for (int j = 0; j < N && !done; j++) {
                        for (int k = 0; k < N && !done; k++)
                                if (i + j + k > N)
                                        done = true;
                                else
                                        cout << i + j + k << " ";
                        cout << "* ";
                }
                cout << "." << endl;
                done = false;
        }
}
```