Basics for Computer Programming

- Data Types and Variables
- Operators
- Data input and output
- Statements and flow control

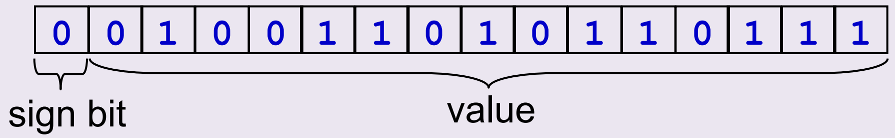
Data Types and Variables

Primitive Built-in Types

Type	Keyword
• Integer	int
• Floating point	float
• Double floating point	double
• Boolean	bool
• Character	char

Integers

- For example, assume in your system an integer has 16 bits.



- Leftmost bit is used for the sign, so 15 bits are left for the value. So, you have 2^{15}=32,768 positive values, ranging from 0 to 32,767. Similarly, you have 32,768 negative values, this time ranging from -1 to -32,768.
- If you have 32 bits (4 bytes) for an integer, than the maximum value is 2^{31}=2,147,483,647.

Type Modifier

Some basic types can be modified using one or more of these type modifiers:

- signed
- unsigned
- short
- long

Type	Size	Range
int	16	-32768 ~ 32767
short [int]	16	-32768 ~ 32767
long [int]	32	-2147483648 ~ 2147483647
unsigned [int]	16	0 ~ 65535
unsigned short	16	0 ~ 65535
unsigned long	32	0 ~ 4294967295

Floating-point numbers

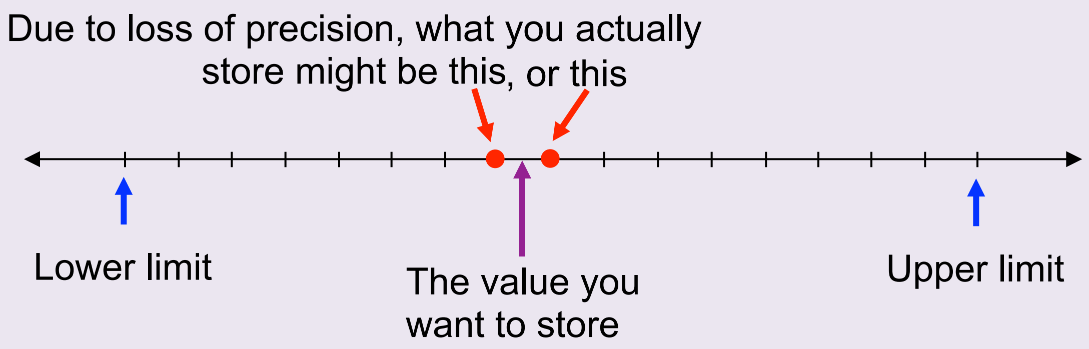
- Syntax:

```
float variable_list;
```

- Float type is used for real numbers.
- Note that all integers may be represented as floating-point numbers, but not vice versa.

Floating-point numbers

- Similar to integers, floats also have their limits: maximum and minimum values are limited as well as the precision.


Due to loss of precision, what you actually store might be this, or this
Lower limit
The value you want to store
Upper limit

Floating-point numbers

- There are two variations of `float`: `double` and `long double`.
  - They have wider range and higher precision.
- The sizes of these types are ordered as follows:

  float ≤ double ≤ long double

Characters

- Syntax:

```
    char variable_list;
```

- Character is the only type that has a fixed size in all implementations: 1 byte.
- All letters (uppercase and lowercase, separately), digits, and signs (such as +,-,!,?,$, £,^,#, comma itself, and many others) are of type character.

Characters

- Since every value is represented with bits (0s and 1s), we need a mapping for all these letters, digits, and signs.
- This mapping is provided by a table of characters and their corresponding integer values.
  - The most widely used table for this purpose is the ASCII table.

Characters

- The ASCII table contains the values for 256 values (of which only the first 128 are relevant for you). Each row of the table contains one character. The row number is called the ASCII code of the corresponding character.

  (The topic of character encoding is beyond the scope of this course. So, we will work with the simplified definition here.)

Characters

- Never memorize the ASCII codes. They are available in all programming books and the Internet. (Eg: http://www.ascii-code.com)
- What is important for us is the following three rules:
  - All lowercase letters (a,b,c,...) are consecutive.
  - All uppercase letters (A,B,C,...) are consecutive.
  - All digits are consecutive.

ASCII table (partial)

ASCII code	Symbol	ASCII code	Symbol	ASCII code	Symbol	ASCII code	Symbol
...	...	66	B	84	T	107	k
32	blank	67	C	85	U	108	l
37	%	68	D	86	V	109	m
42	*	69	E	87	W	110	n
43	+	70	F	88	X	111	o
...	...	71	G	89	Y	112	p
48	0	72	H	90	Z	113	q
49	1	73	I	...	...	114	r
50	2	74	J	97	a	115	s
51	3	75	K	98	b	116	t
52	4	76	L	99	c	117	u
53	5	77	M	100	d	118	v
54	6	78	N	101	e	119	w
55	7	79	O	102	f	120	x
56	8	80	P	103	g	121	y
57	9	81	Q	104	h	122	z
...	...	82	R	105	i	...	...
65	A	83	S	106	j		

Characters

- A character variable actually stores the ASCII value of the corresponding letter, digit, or sign.
- I/O functions (printf(), scanf(), etc.) do the translation between the image of a character displayed on the screen and the ASCII code that is actually stored in the memory of the computer.

Characters

- Note that **a** and **A** have different ASCII codes (**97** and **65**).
- You could also have a variable with name **a**. To differentiate between the variable and the character, we specify all characters in single quotes, such as **'a'**. Variable names are never given in quotes.
  - Example:

```
char ch;
ch='a';
```

- Note that using double quotes makes it a string (to be discussed later in the course) rather than a character. Thus, **'a'** and **"a"** are different.
- Similarly, **1** and **'1'** are different. Former has the value **1** whereas the latter has the ASCII value of **49**.

Characters

- Example: Consider the code segment below.

```
char ch;
ch='A';
printf("Output is %c", ch);
```

- The string in printf() is stored as

```
79,117,116,112,117,116,32,105,115,32,37,99
```

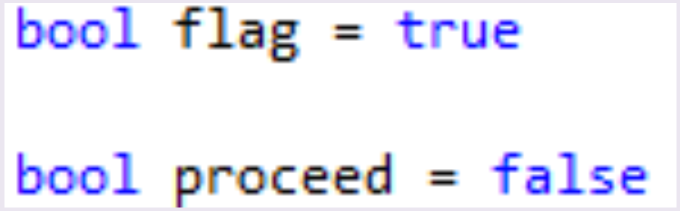
which are the ASCII codes of the characters in the string.

- When printf() is executed, it first replaces **`37,99`** (**`%c`**) with **`65`** (**`A`**), and then displays the corresponding characters on the screen.

Boolean

- Boolean variables only have two possible values: true (1) and false (0).

Example:



```
bool flag = true

bool proceed = false
```

Variables

Properties of variables:

1. Named storage that our programs can manipulate.
2. Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory;
3. The set of operations that can be applied to the variable

Naming Rule

1. Rule: The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C ++ is case-sensitive. The name of a variable can not be the same as the C++ reserved names for example (int)
2. Convention: The name of a variable is normally lowercase.
3. Meaningful: For instance, “sum” is a good name for summarization, instead of using “aab”, “A1” or B23.

Variable Types

Type	Description
Bool	Stores either value true or false.
char is	Typically a single octet(one byte). This an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
Double	A double-precision floating point value.

Variable Definition

Variable definition:

Tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type, and contains a list of one or more variables of that type.

For example:

```
char x;            // definition of x.

int f = 5;         // definition and initializing  f.
```

Operators

- We will cover the most basic operators in class. More operators will be covered in the labs.
- Assignment operator (=)
  - Note that this is not the "equals" operator. It should be pronounced as "becomes." (Equals is another operator.)
  - The value of the expression on the RHS is assigned (copied) to the LHS.
  - It has right-to-left associativity.

```
a=b=c=10;
```

makes all three variables `10`.

NEW YORK UNIVERSITY ABU DHABI

Assignment and type conversion

- When a variable of a narrower type is assigned to a variable of wider type, no problem.
  - Eg:
    ```
    int a=10;    float f;
        f=a;
    ```
- However, there is loss of information in reverse direction.
  - Eg:
    ```
    float f=10.9; int a;
        a=f;
    ```

Operators

- Arithmetic operators (+,-,*,/,%)
  - General meanings are obvious.
  - What is important is the following: If one of the operands is of a wider type, the result is also of that type. (Its importance will be more obvious soon.)
    - Eg: Result of int+float is float. Result of float+double is double.
  - In C++ language, there are two types of division: integer division and float division.
    - If both operands are of integer class, we perform integer division and the result is obtained by truncating the decimal part.
      - Eg: `8/3` is `2`, not `2.666667`.
    - If one of the operands is of float class, the result is float.
      - Eg: `8.0/3` or `8/3.0` or `8.0/3.0` is `2.666667`, not `2`.

Operators

- Remainder operator is `%`. Both operands must be of integer class.
  - Eg: `10%6` is `4` (equivalent to `10 mod 6`)
- +,-,*,/,% have left-to-right associativity. That means `a/b/c` is equivalent to `(a/b)/c`, but not `a/(b/c)`.

Operators

- Logic operators (&&, ||, !)
  - Logic operators take integer class operands.
    - Zero means false.
    - Anything non-zero means true.
  - "`&&`" does a logical-AND operation. (True only if both operands are true.)
  - "`||`" does a logical-OR operation. (False only if both operands are false.)
  - "`!`" does a negation operation. (Converts true to false, and false to true.)

Operators

- Logic operators follow the logic rules

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

- The order of evaluation is from left to right
- As usual parenthesis overrides default order

Operators

- If the first operand of the "`&&`" operator is false, the second operand is not evaluated at all (since it is obvious that the whole expression is false).
  - Eg: In the expression below, if the values of `b` and `c` are initially `0` and `1`, respectively,

    ```
    a = b && (c=2)
    ```

    then the second operand is not evaluated at all, so `c` keeps its value as `1`.
- Similarly, if the first operand of the "`||`" operator is true, the second operand is not evaluated at all.

Operators

- Bitwise operators (&, |, ^, <<, >>, ~)
  - Bitwise operators take integer class operands.
    - For the logic operators, the variable represents a single logical value, true or false.
    - For the bitwise operators, each bit of the variable represents true or false.
  - **&**, **|**, and **^** perform bitwise-AND, -OR, -XOR, respectively.
  - **<<** and **>>** perform left- and right-shifts.
  - "**~**" takes bitwise one’s complement.

Operators

Operation	Result

`5 & 10`
`(0000 0101 &`
` 0000 1010)`
	`0`
	`(0000 0000)`

`5 && 10`
`(0000 0101 &&`
` 0000 1010)`
	`1`
	`(0000 0001)`

`5 | 10`
`(0000 0101 |`
` 0000 1010)`
	`15`
	`(0000 1111)`

`8 ^ 10`
`(0000 0111 ^`
` 0000 1010)`
	`13`
	`(0000 1101)`

`7 << 2`
`(0000 0111 << 0000 0010)`
	`28`
	`(0001 1100)`

`7 >> 2`
`(0000 0111 >> 0000 0010)`
	`1 (0000 0001)`

`~5`
`(~0000 0101)`
	`-6 (in two's complement)`
	`(1111 1010)`

Operators

- Other assignment operators (+=, -=, *=, /=, %=)
  - Instead of writing `a=a+b`, you can write `a+=b` in short. Similar with `-=`, `*=`, `/=`, and others.

Operators

NEW YORK UNIVERSITY ABU DHABI

- Pre/Post increment/decrement operators (++, --)
  - The operator ++ increments the value of the operand by 1.
    - If the operator comes BEFORE the variable name, the value of the variable is incremented before being used, i.e., the value of the expression is the incremented value. This is pre-increment.
    - In post-increment, the operator is used after the variable name, and incrementation is performed after the value is used, i.e., the value of the expression is the value of the variable before incrementation.

Operators

– Eg:

```
a=10;          c=10,
b=++a;         d=c++;
```

Both `a` and `c` will be come `11`, but `b` will be `11` while `d` is `10`.

Operators

- Comparison operators (==,!=,<,<=,...)
  - "==" is the "is equal to" operator. Like all other comparison operators, it evaluates to a Boolean value of true or false, no matter what the operand types are.
  - IMPORTANT: When you compare two float values that are supposed to be equal mathematically, the comparison may fail due to the loss of precision discussed before.

Operators

NEW YORK UNIVERSITY ABU DHABI

Symbol	Usage	Meaning
==	x == y	is x equal to y?
!=	x != y	is x not equal to y?
>	x > y	is x greater than y?
<	x < y	is x less than y?
>=	x >= y	is x greater than or equal to y?
<=	x <=y	is x less than or equal to y?

Operators

- We can create complex expressions by joining several expressions with logic operators.

Symbol	Usage	Meaning
&&	exp1 && exp2	AND
||	exp1 || exp2	OR
!	! exp	NOT

Operators

- While using multiple operators in the same expression, you should be careful with the precedence and associativity of the operands.
  - Eg: The following does NOT check if `a` is between `5` and `10`.

    ```
    bool = 5<a<10;
    ```

    - `bool` will be true if `a` is `20`. (Why?)
  - Don't hesitate to use parentheses when you are not sure about the precedence (or to make things explicit).

Operator precedence table

Operator	Associativity
() [] . ->	left-to-right
++ -- + - ! ~ (type) * & sizeof	right-to-left
* / %	left-to-right
+ -	left-to-right
<< >>	left-to-right
< <= > >=	left-to-right
== !=	left-to-right
&	left-to-right
^	left-to-right
|	left-to-right
&&	left-to-right
||	left-to-right
?:	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	right-to-left
,	left-to-right

NEW YORK UNIVERSITY ABU DHABI

Operators

- Precedence, associativity, and order of evaluation:
  - In the table is given in the previous slide, precedence decreases as you go down.
  - If two operands in an expression have the same precedence, you decide according to the associativity column.
  - There is a common misunderstanding about associativity.
    - **Note that associativity has nothing to do with the order of evaluation of the operands.**
    - **Order of evaluation of operands is not specified in C++ language.**

Data Input and Output

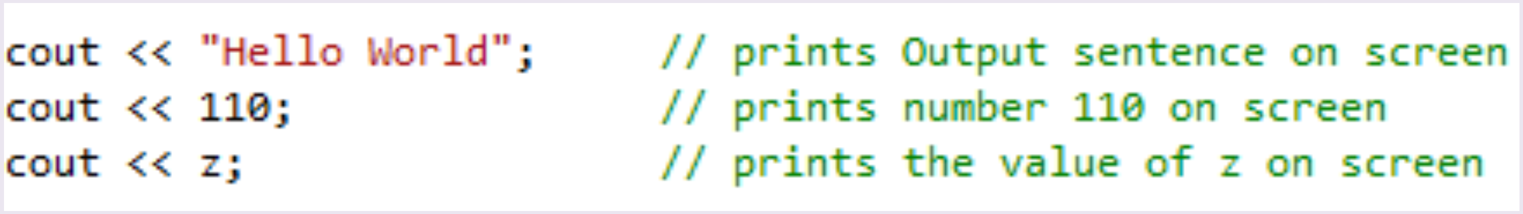
- C++ standard library provides definitions of a group of defines a handful of stream objects that can be used to access what are considered the standard sources and destinations of characters by the environment where the program runs:

Stream	Description
– cin	standard input stream
– cout	standard output stream
– cerr	standard error (output) stream
– clog	standard logging (output) stream

Data Output: Cout

- On most program environments, the standard output by default is the screen, and the C++ stream object defined to access it is cout.

For examples:



```
cout << "Hello World";      // prints Output sentence on screen
cout << 110;                // prints number 110 on screen
cout << z;                  // prints the value of z on screen
```

Data Output: cout

1. The double quoting is what makes the difference:

```
cout << "OutputValue";  // prints OutputValue
cout << OutputValue;    // prints the content of variable OutputValue
```

2. Multiple insertion operations (<<) may be chained in a single statement:

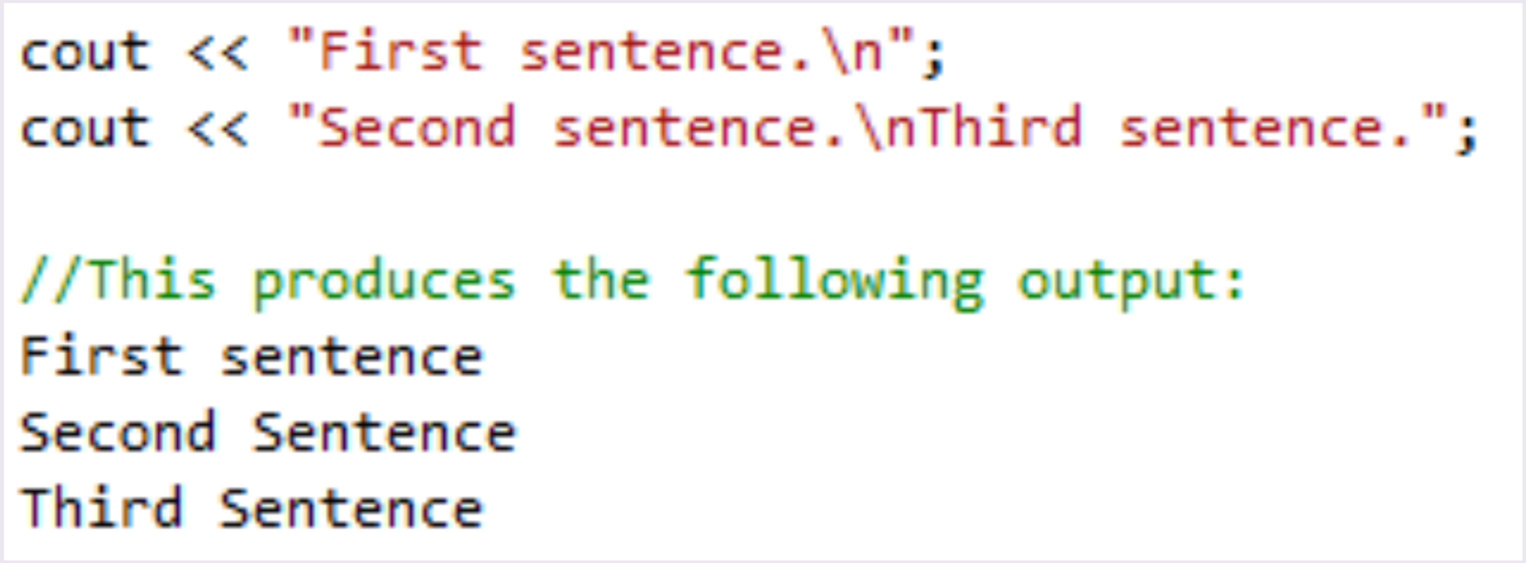
```
// Prints Welcome to Computer System Programming: ENGR-AD-202 on screen
cout << "Welcome to" << " Computer System Programming:  " << "ENGR-AD-202";
```

3. Mixture of literals and variables in a single statement:

```
// Prints My name is Yi Fang and my age is 32
cout << "My name is " << name << " and my age is " << age;
```

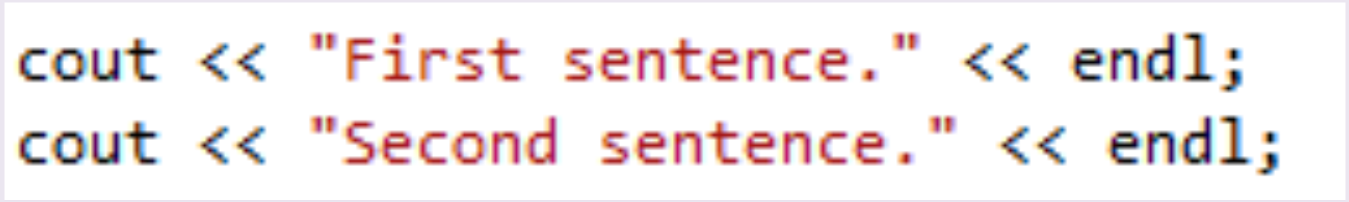
Note: assuming the name variable contains Yi Fang and age variable contains 32

4. Break lines in C++ statement:



```
cout << "First sentence.\n";
cout << "Second sentence.\nThird sentence.";

//This produces the following output:
First sentence
Second Sentence
Third Sentence
```

Alternatively, the endl manipulator can also be used to break lines.

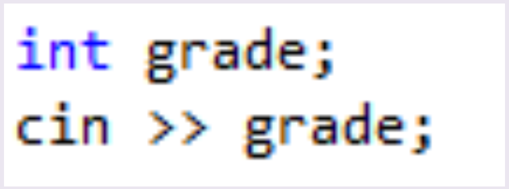


```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

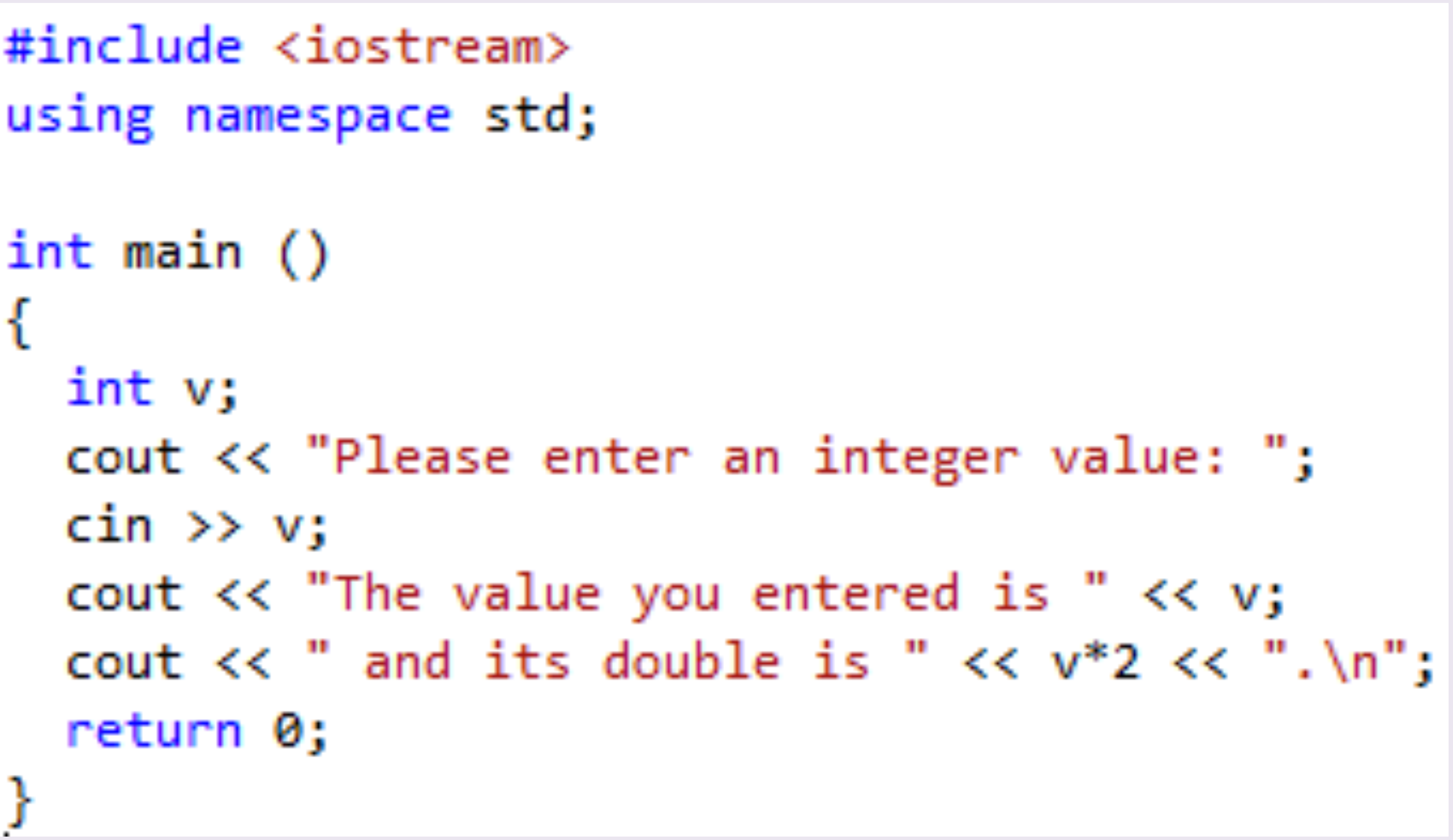
Data Output: cin

- In most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is cin.

For examples:

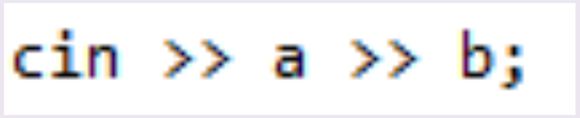


```
int grade;
cin >> grade;
```

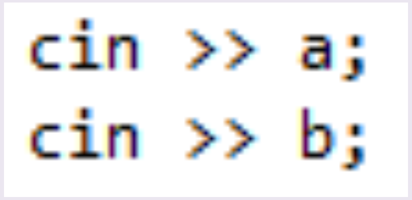


```cpp
#include <iostream>
using namespace std;

int main ()
{
  int v;
  cout << "Please enter an integer value: ";
  cin >> v;
  cout << "The value you entered is " << v;
  cout << " and its double is " << v*2 << ".\n";
  return 0;
}
```

Extractions on cin can also be chained to request more than one datum in a single statement:



```
cin >> a >> b;
```



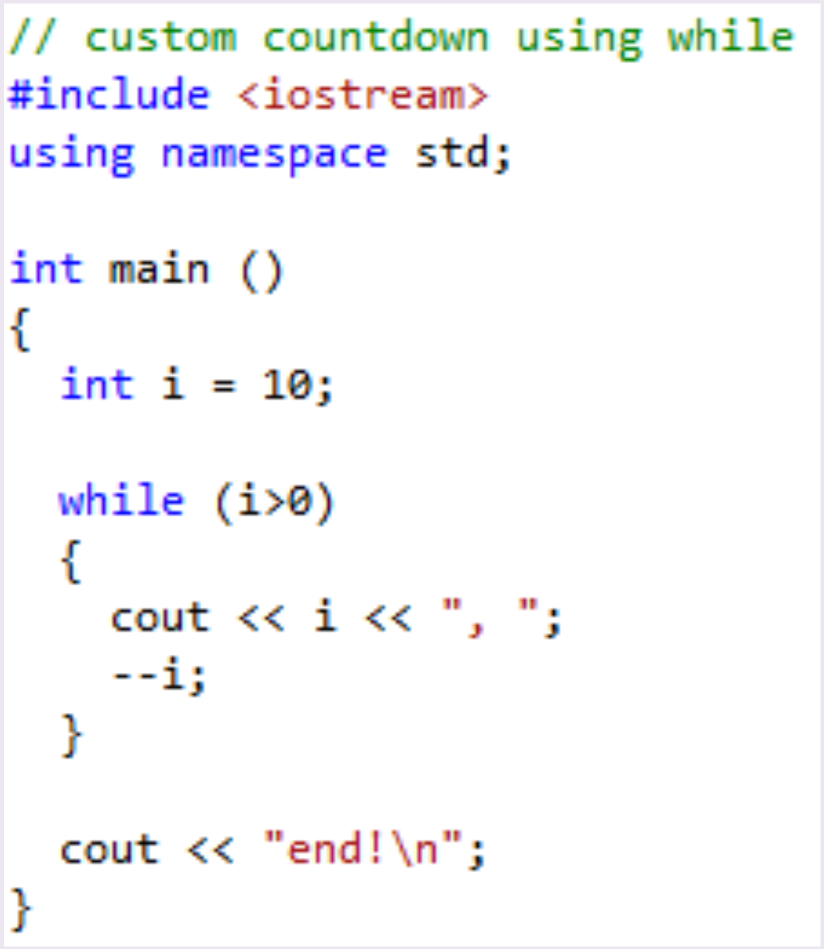
```
cin >> a;
cin >> b;
```

Statements and flow control

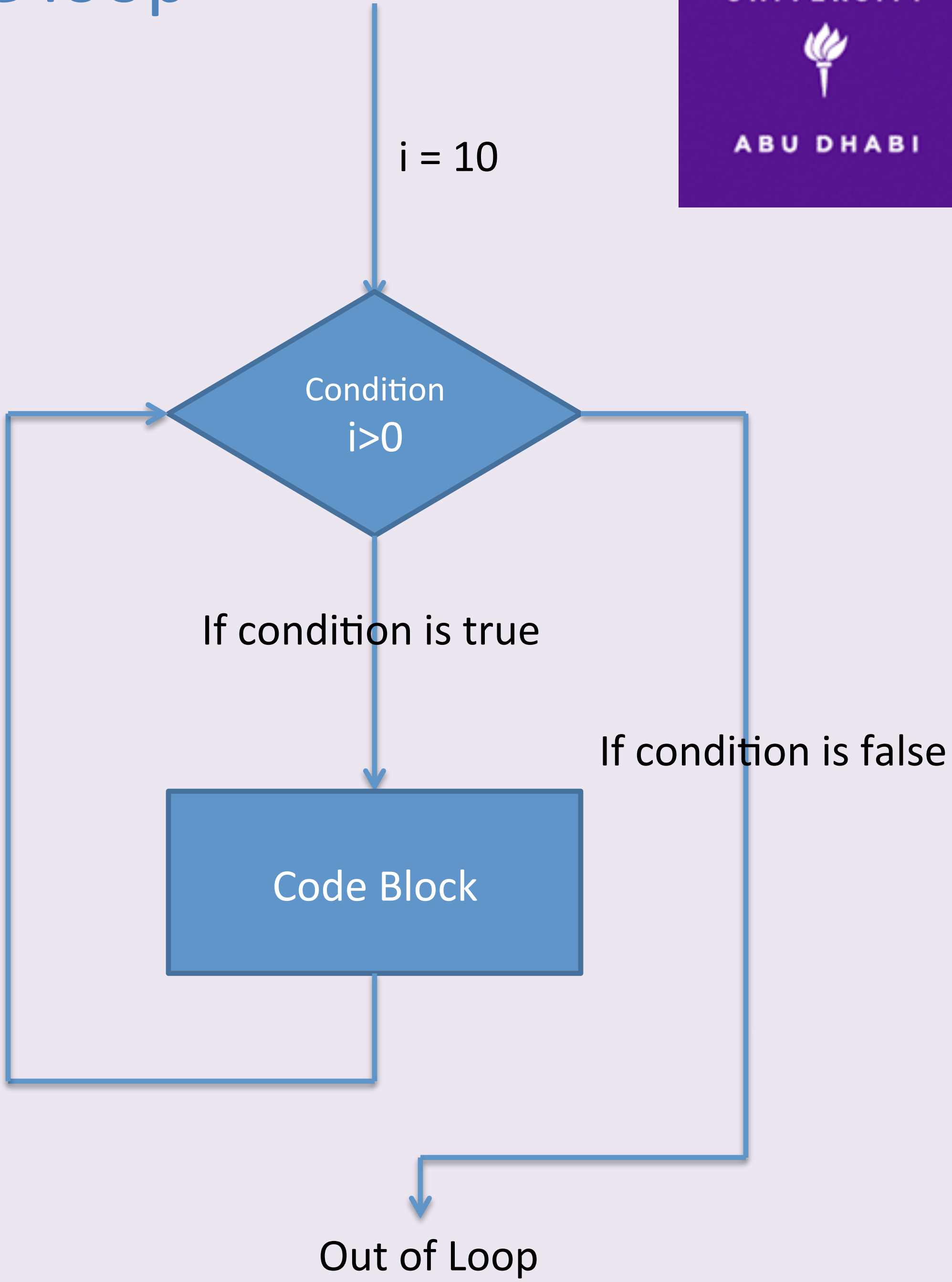
- Iteration statements (loops)
  1. while loop
  2. for loop
  3. do...while loop
  4. nested loops

- Selection statements
  1. if statement
  2. if...else statement
  3. switch statement
  4. nested if statements
  5. nested switch statements

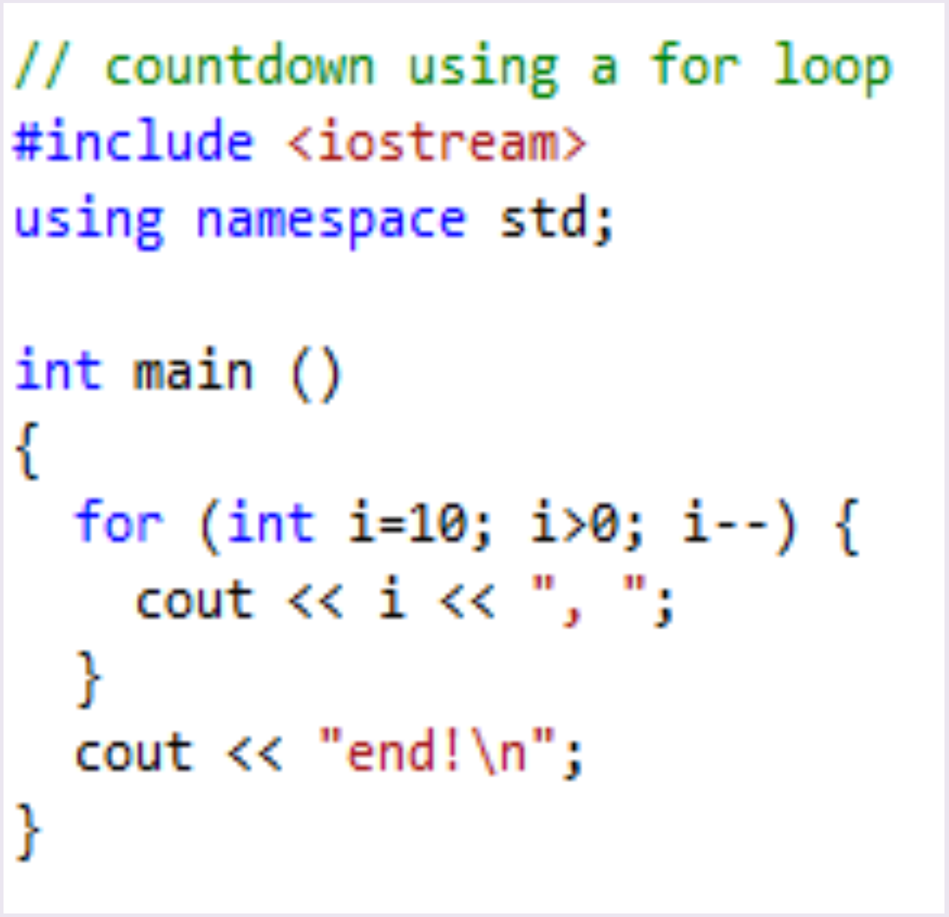
Iteration statements: while loop



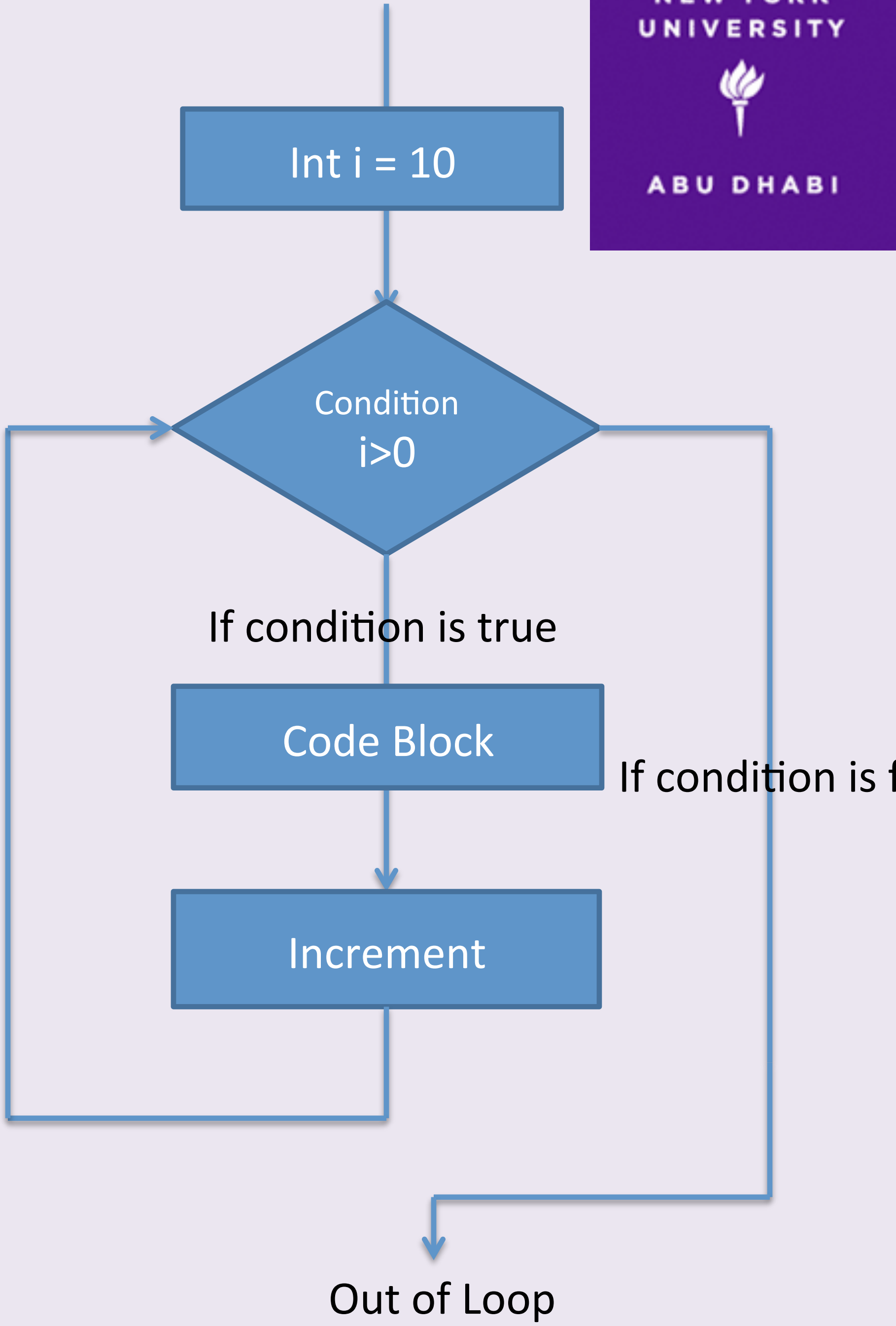
```
// custom countdown using while
#include <iostream>
using namespace std;

int main ()
{
  int i = 10;

  while (i>0)
  {
    cout << i << ", ";
    --i;
  }

  cout << "end!\n";
}
```



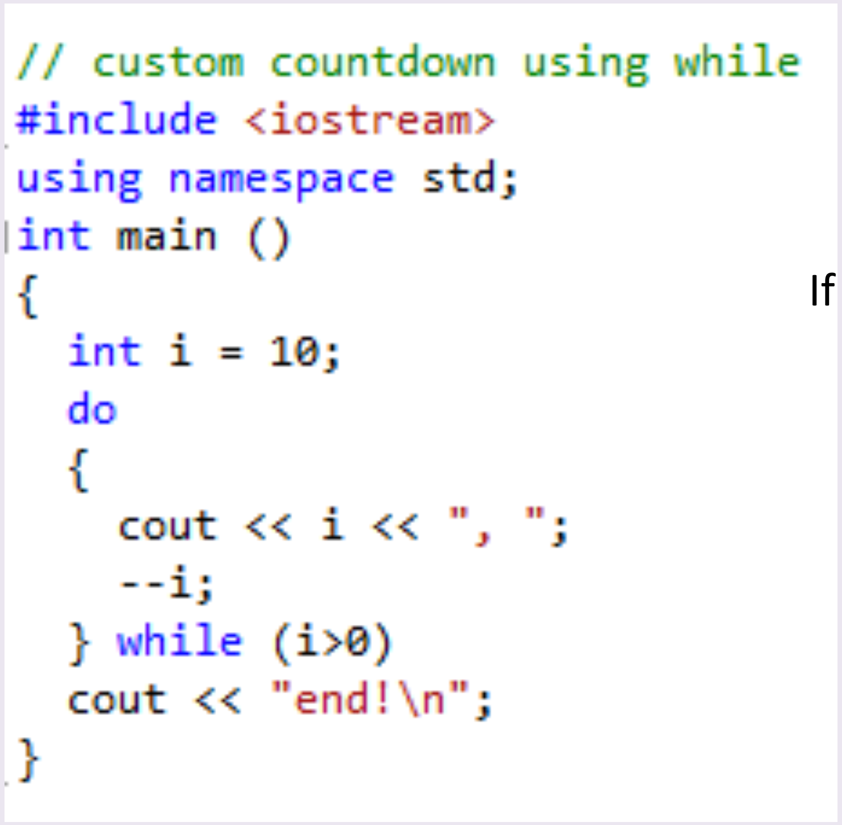
Iteration statements: for loop



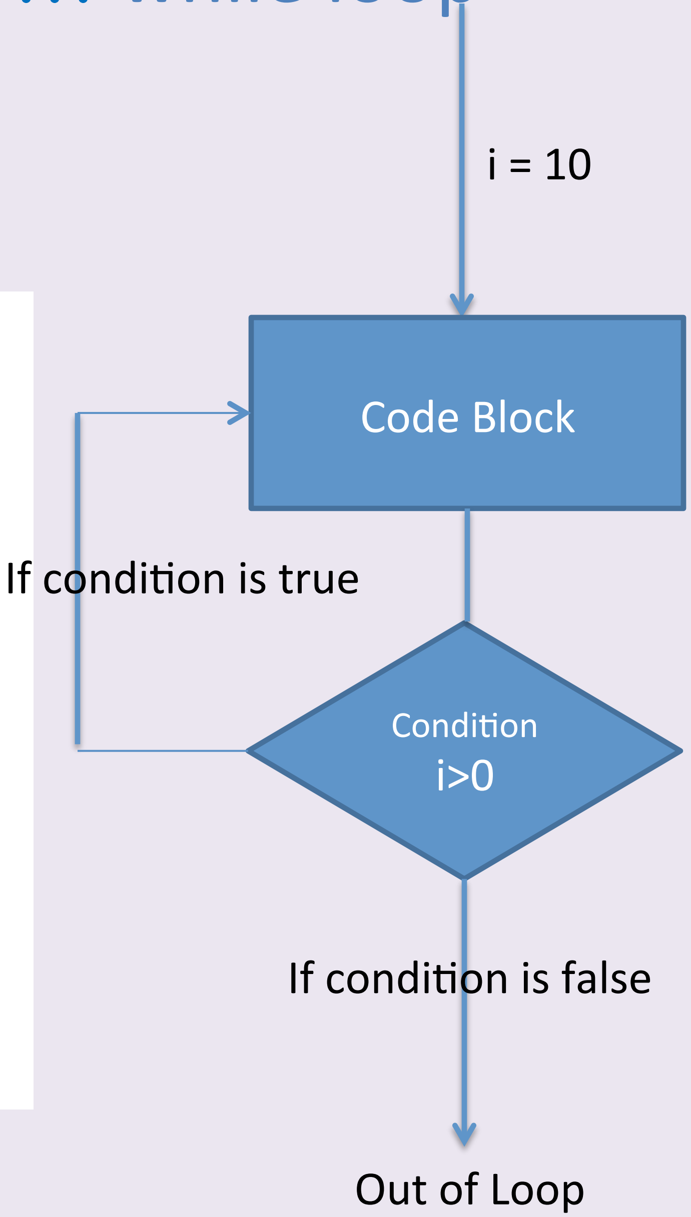
```cpp
// countdown using a for loop
#include <iostream>
using namespace std;

int main ()
{
  for (int i=10; i>0; i--) {
    cout << i << ", ";
  }
  cout << "end!\n";
}
```



Iteration statements: do … while loop



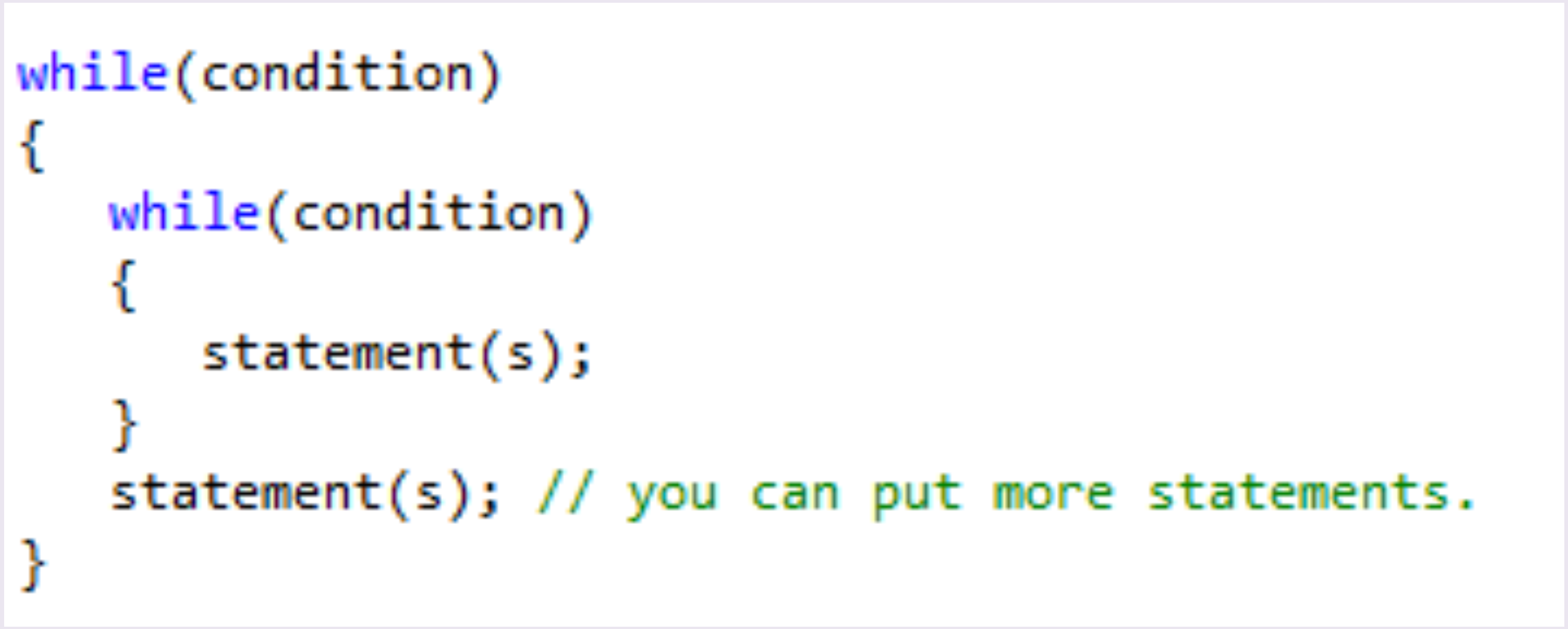
```
// custom countdown using while
#include <iostream>
using namespace std;
int main ()
{
  int i = 10;
  do
  {
    cout << i << ", ";
    --i;
  } while (i>0)
  cout << "end!\n";
}
```



Iteration statements: nested loop

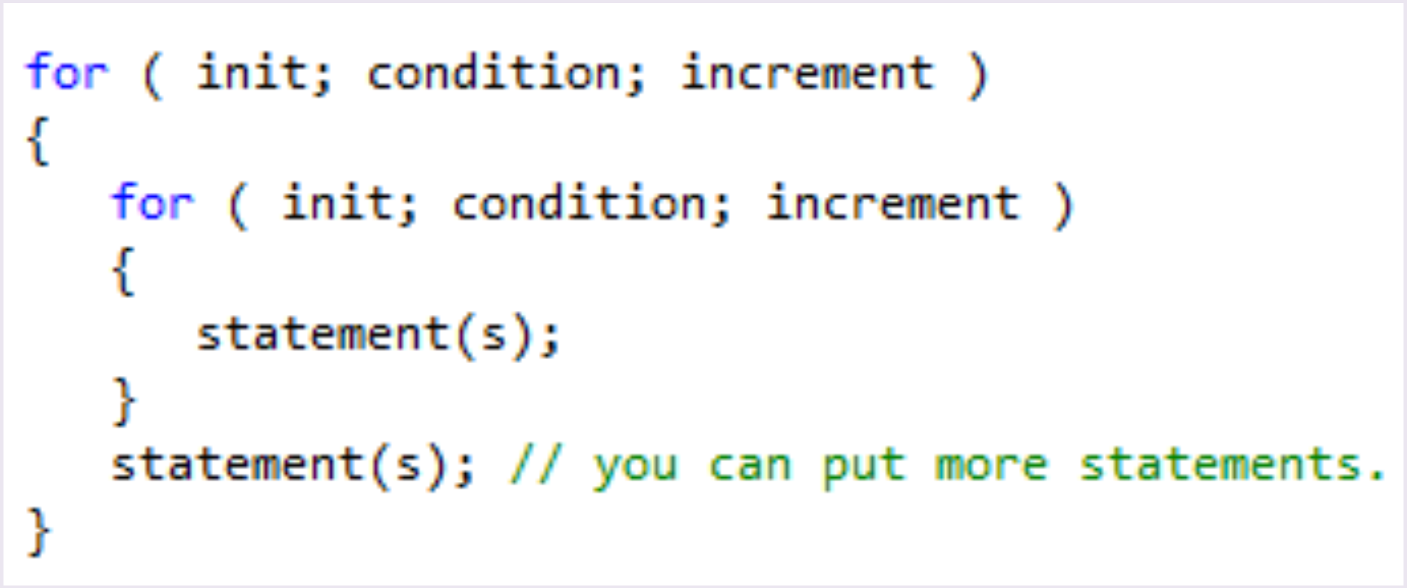
A loop can be nested inside of another loop.

For example:



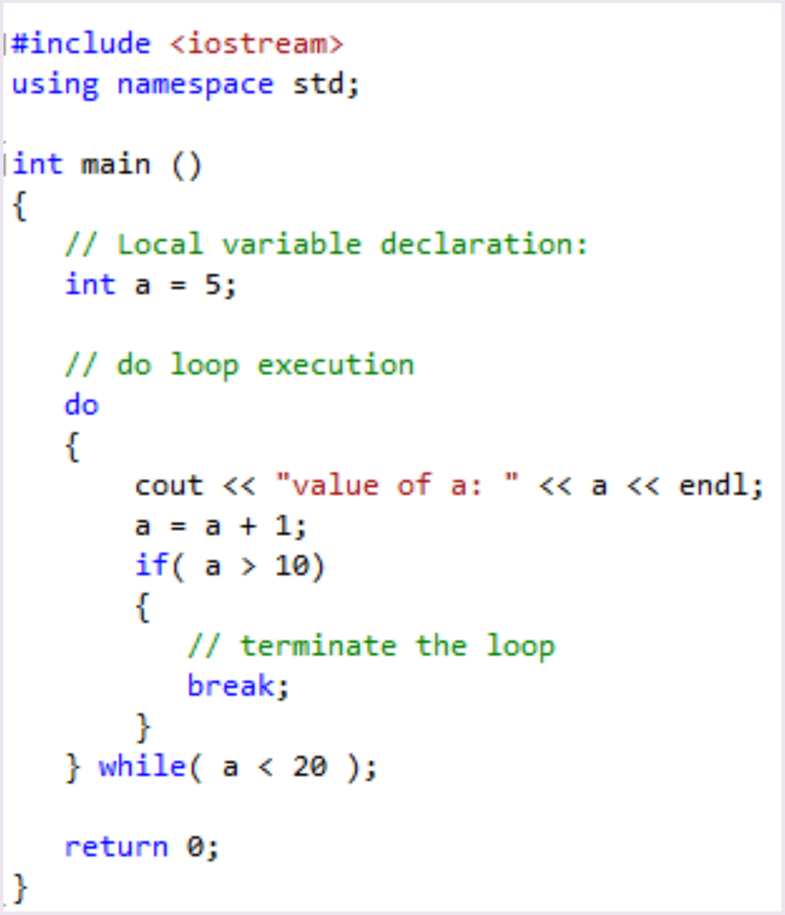
```
while(condition)
{
    while(condition)
    {
        statement(s);
    }
    statement(s); // you can put more statements.
}
```

Iteration statements: nested loop

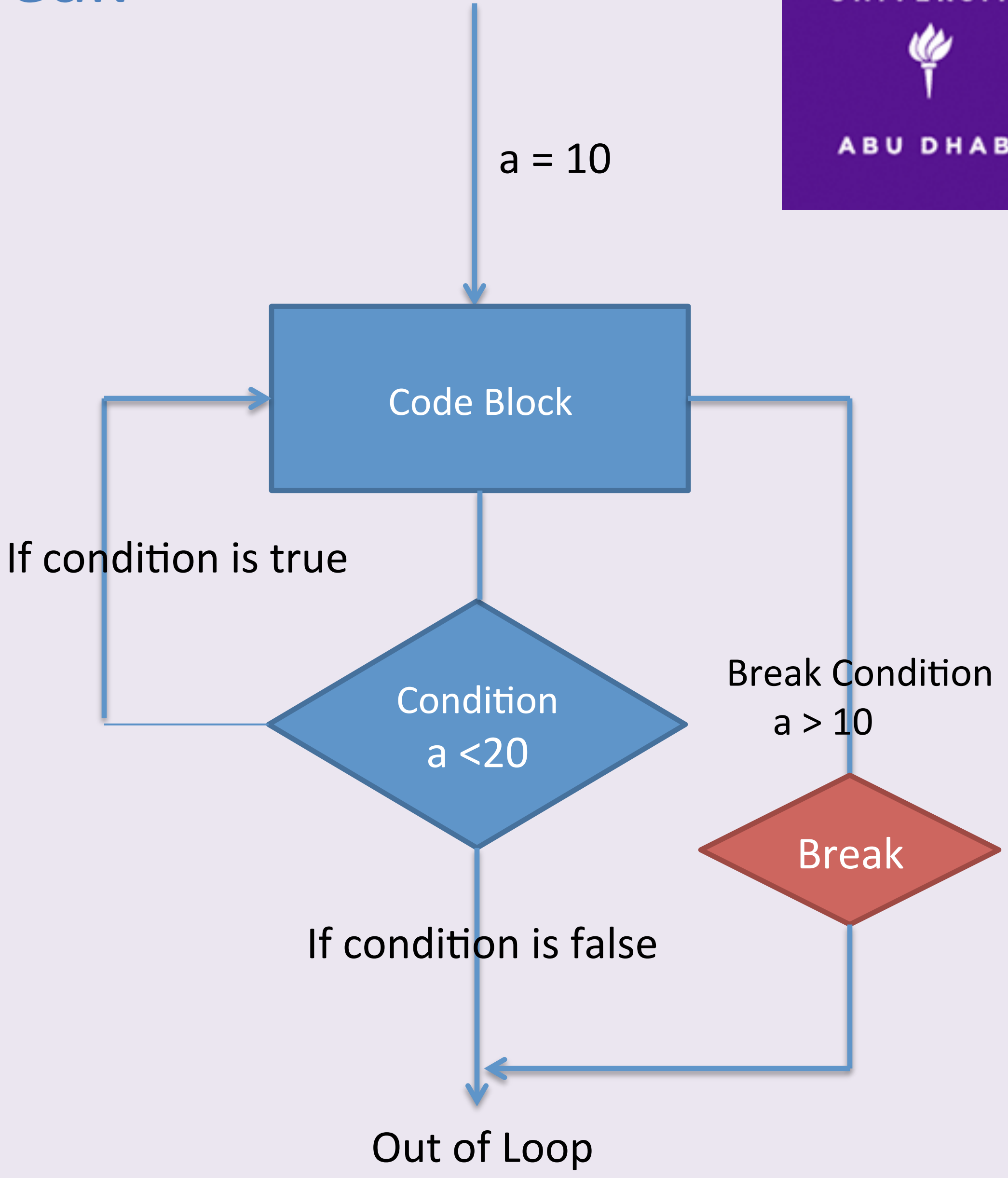


```
for ( init; condition; increment )
{
   for ( init; condition; increment )
   {
      statement(s);
   }
   statement(s); // you can put more statements.
}
```

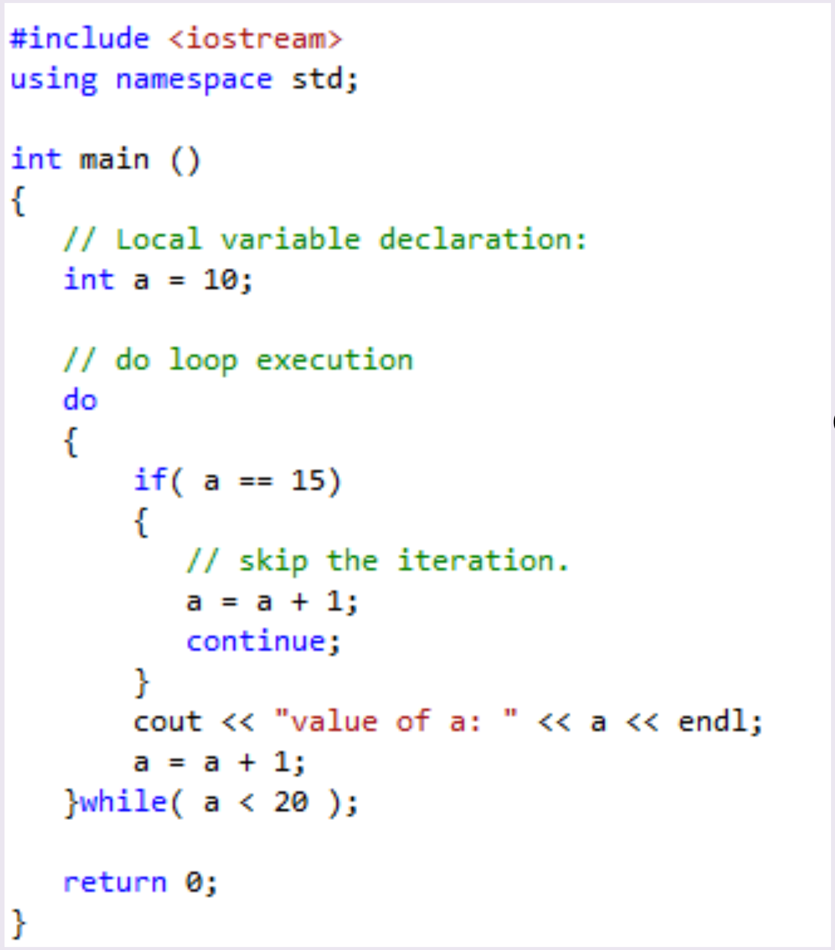
Iteration statements: break



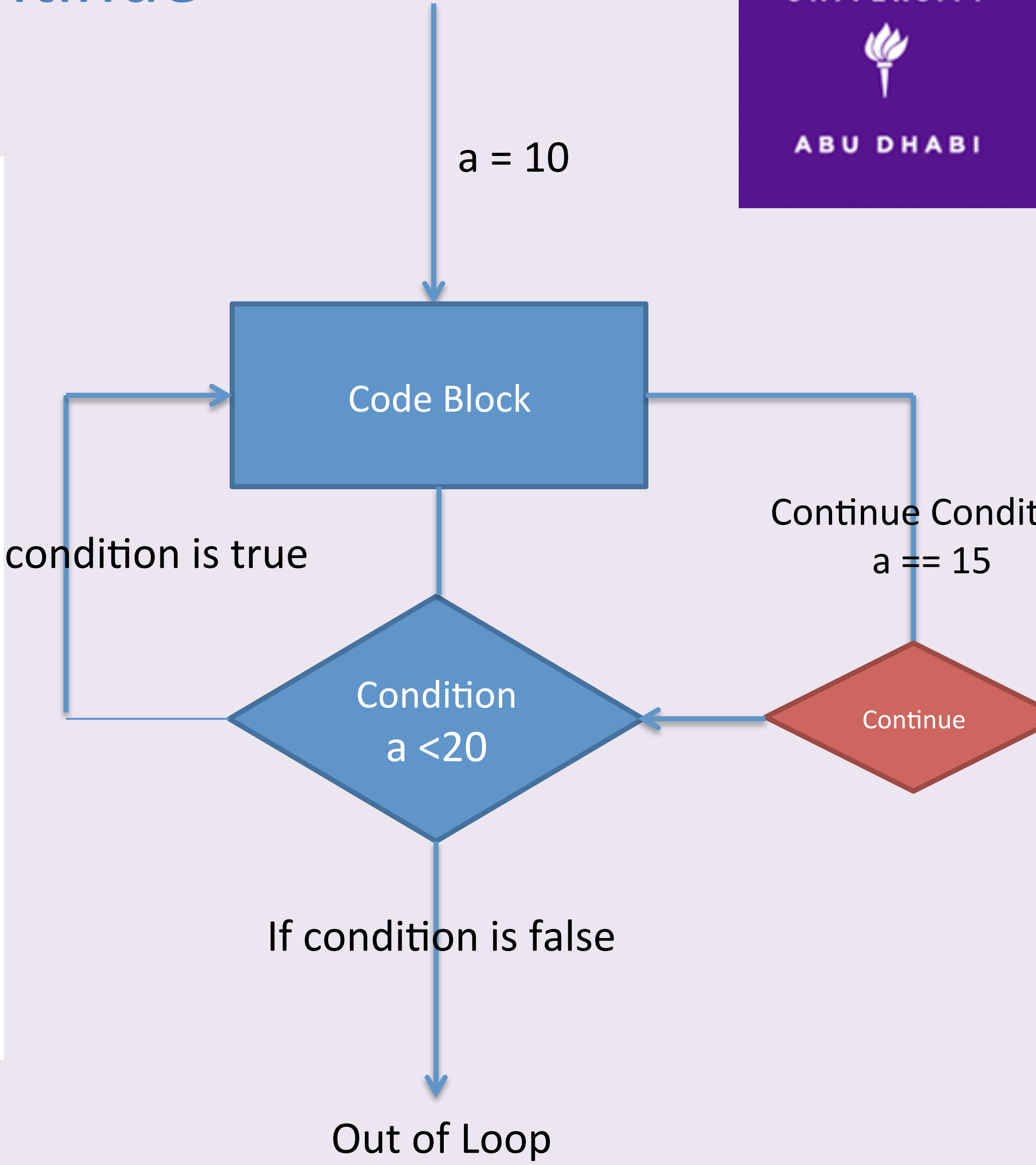
```
#include <iostream>
using namespace std;

int main ()
{
   // Local variable declaration:
   int a = 5;

   // do loop execution
   do
   {
       cout << "value of a: " << a << endl;
       a = a + 1;
       if( a > 10)
       {
          // terminate the loop
          break;
       }
   } while( a < 20 );

   return 0;
}
```



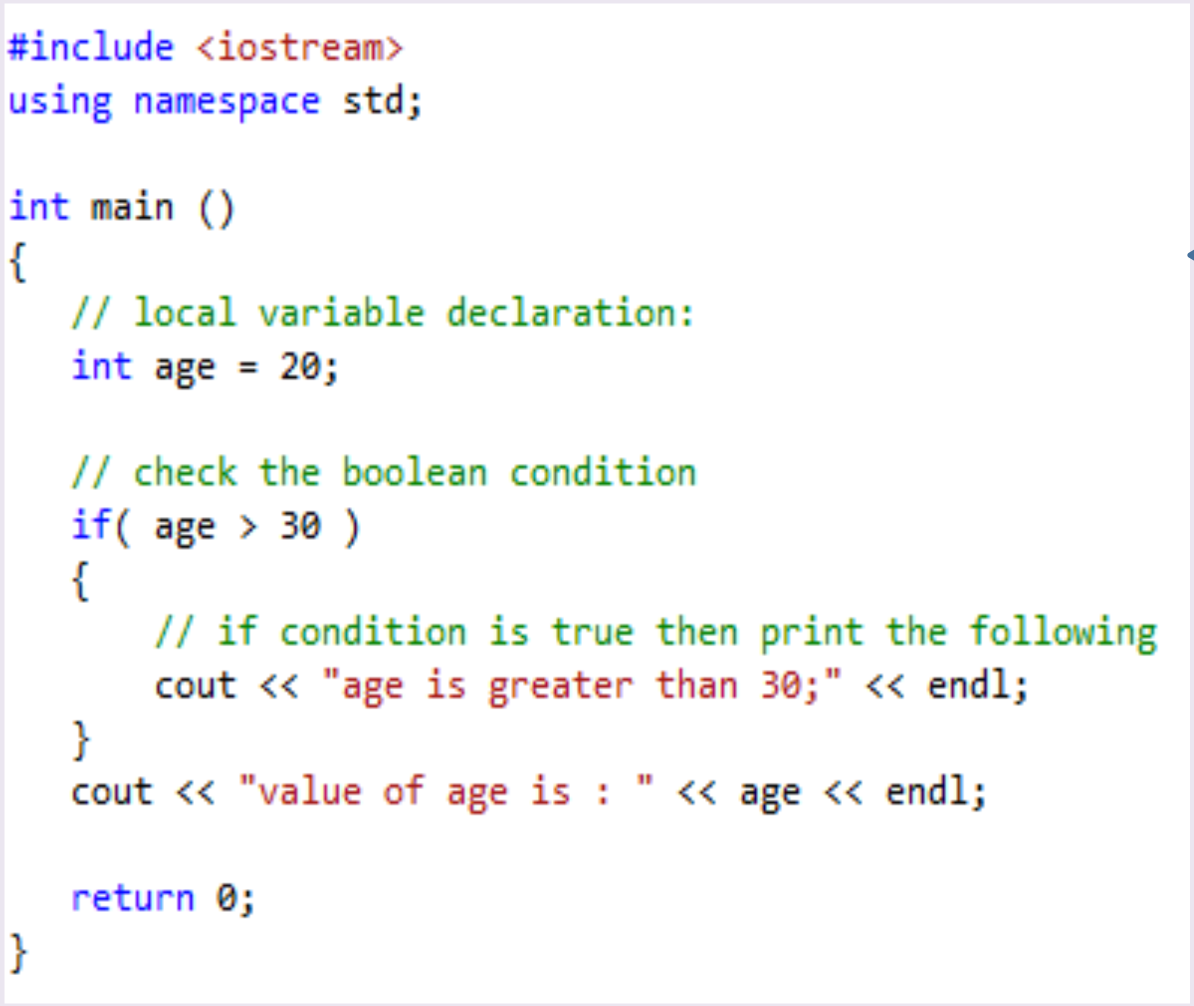
Iteration statements: Continue



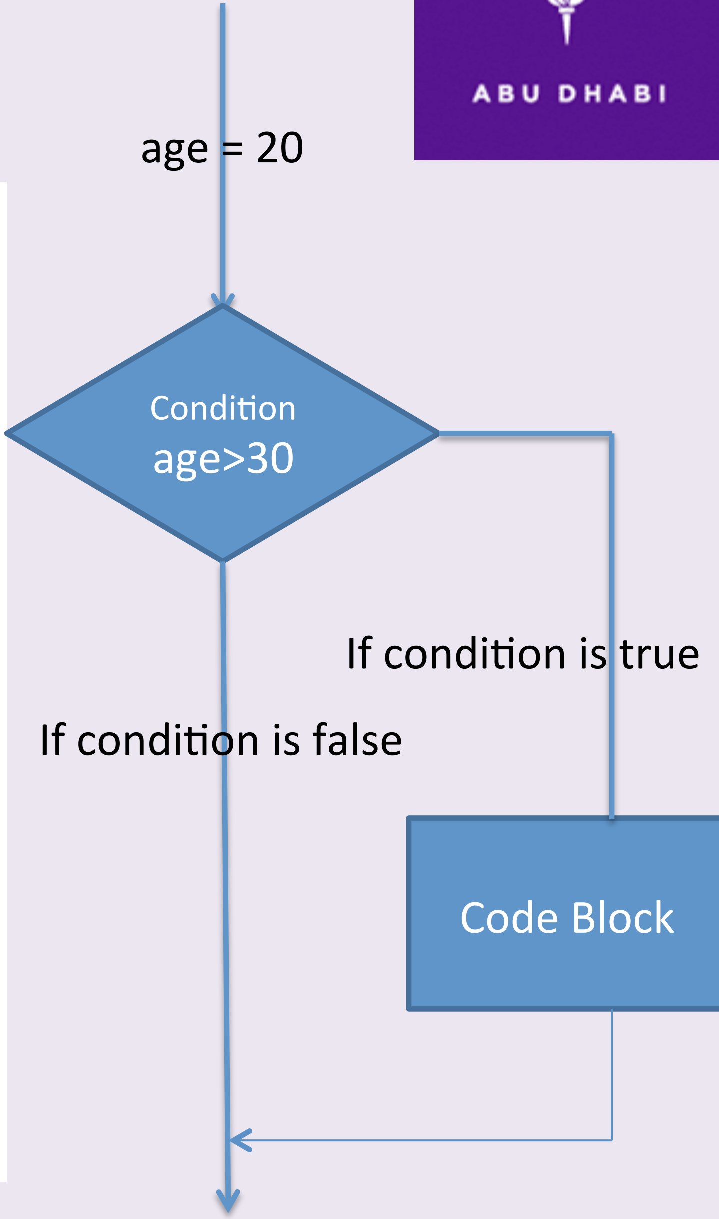
```
#include <iostream>
using namespace std;

int main ()
{
   // Local variable declaration:
   int a = 10;

   // do loop execution
   do
   {
       if( a == 15)
       {
          // skip the iteration.
          a = a + 1;
          continue;
       }
       cout << "value of a: " << a << endl;
       a = a + 1;
   }while( a < 20 );

   return 0;
}
```



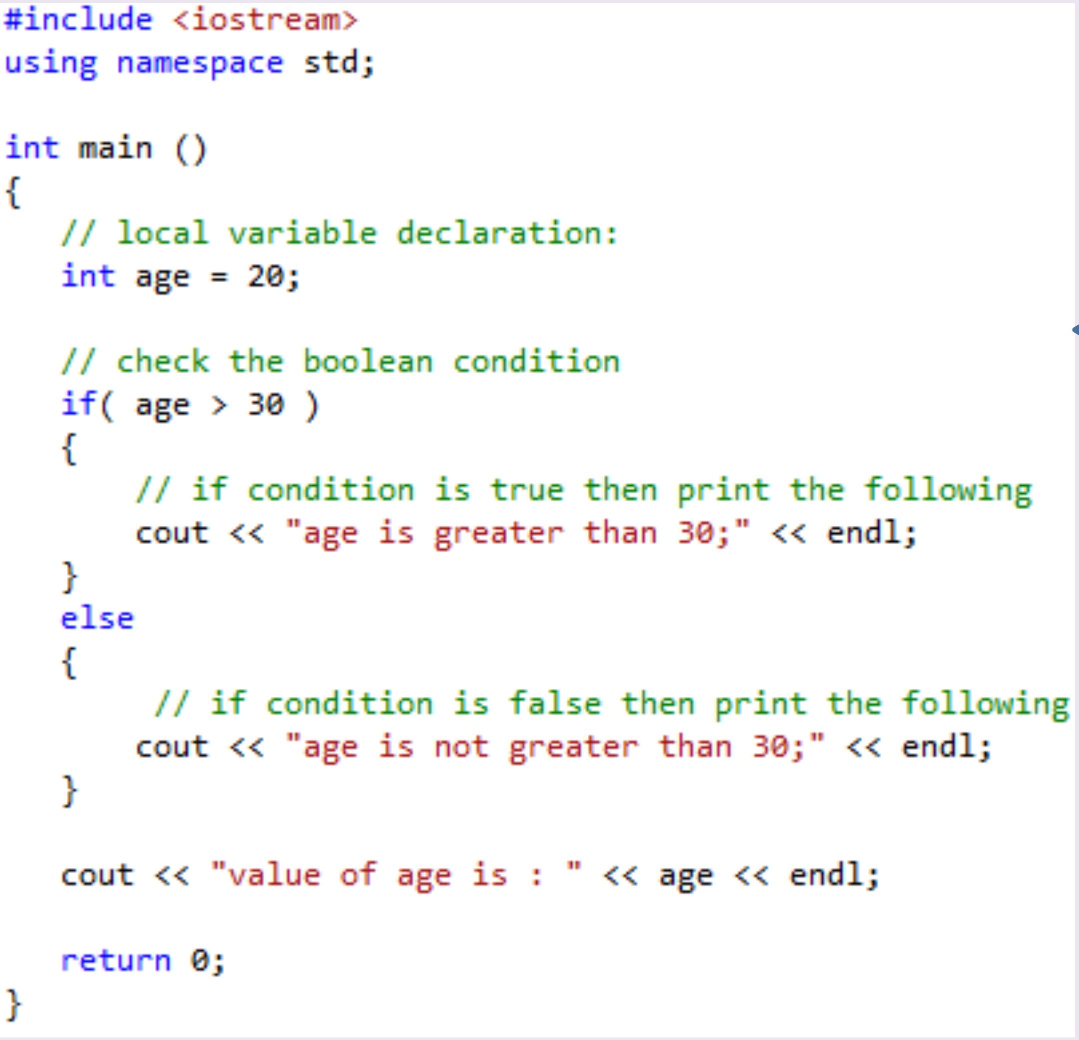
Selection statements: if statement



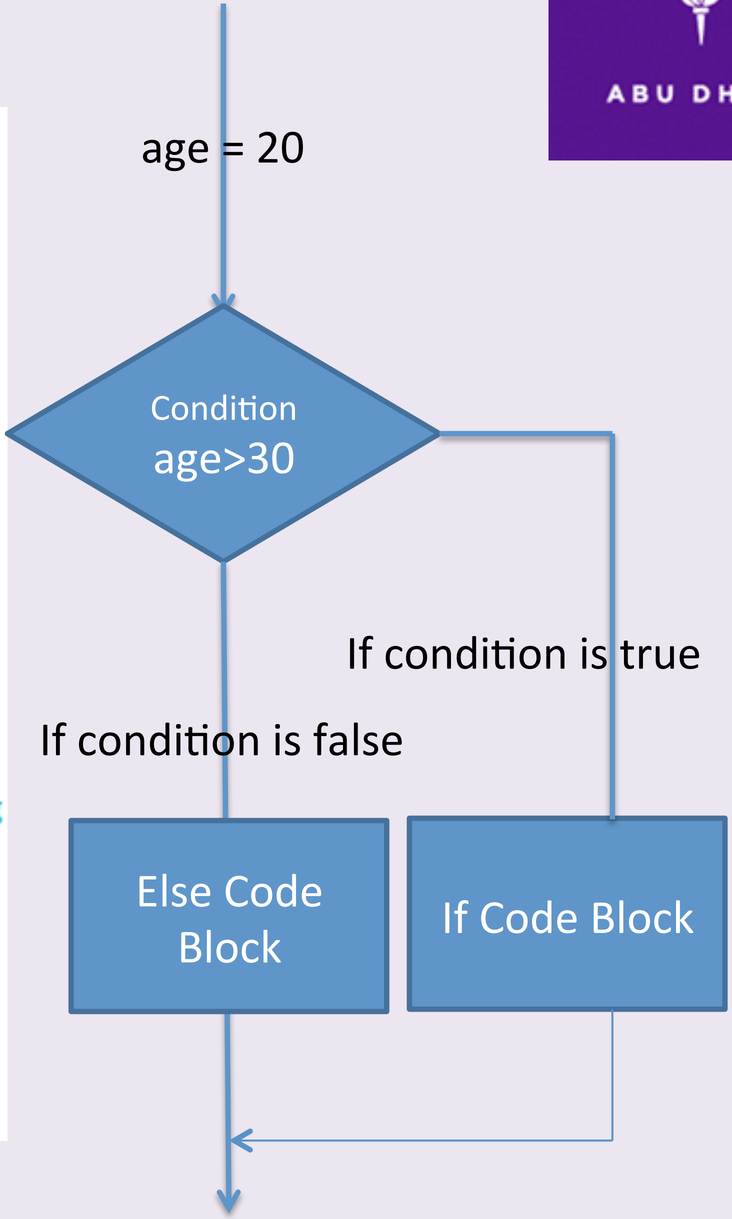
```
#include <iostream>
using namespace std;

int main ()
{
   // local variable declaration:
   int age = 20;

   // check the boolean condition
   if( age > 30 )
   {
       // if condition is true then print the following
       cout << "age is greater than 30;" << endl;
   }
   cout << "value of age is : " << age << endl;

   return 0;
}
```



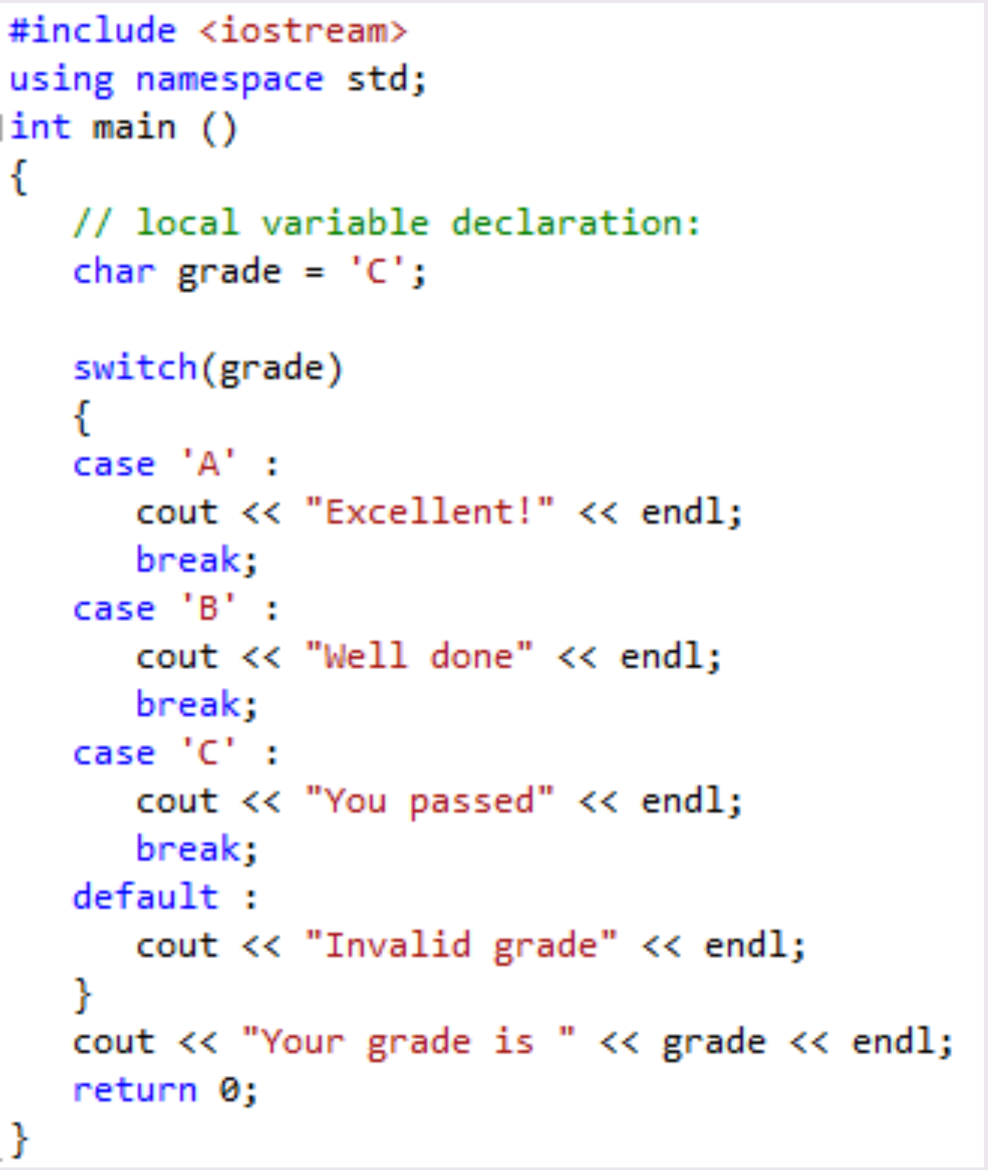
Selection statements: if else statement



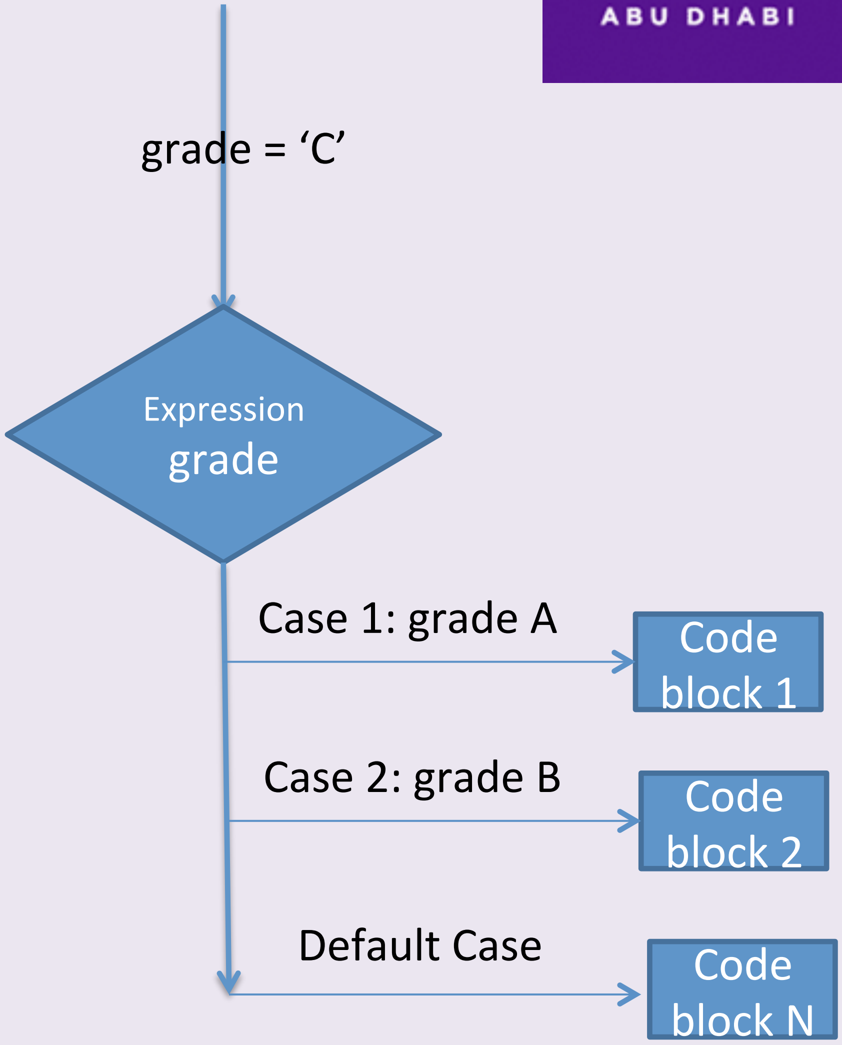
```
#include <iostream>
using namespace std;

int main ()
{
   // local variable declaration:
   int age = 20;

   // check the boolean condition
   if( age > 30 )
   {
       // if condition is true then print the following
       cout << "age is greater than 30;" << endl;
   }
   else
   {
        // if condition is false then print the following
       cout << "age is not greater than 30;" << endl;
   }

   cout << "value of age is : " << age << endl;

   return 0;
}
```



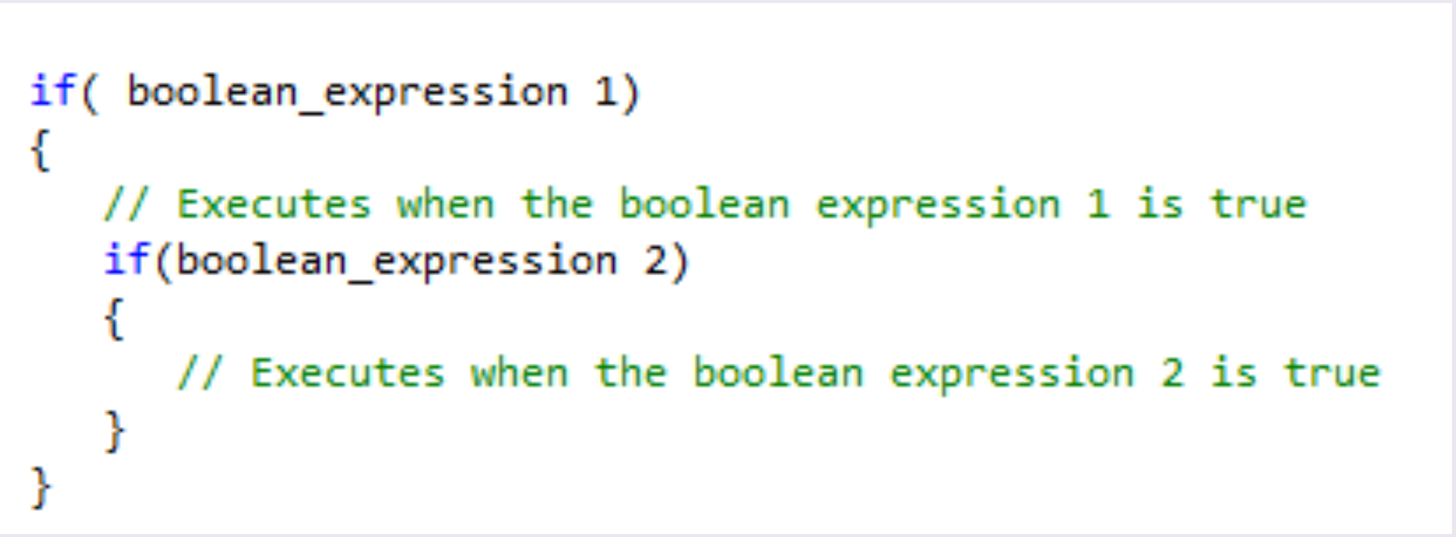
Selection statements: switch statement



```
#include <iostream>
using namespace std;
int main ()
{
   // local variable declaration:
   char grade = 'C';

   switch(grade)
   {
   case 'A' :
      cout << "Excellent!" << endl;
      break;
   case 'B' :
      cout << "Well done" << endl;
      break;
   case 'C' :
      cout << "You passed" << endl;
      break;
   default :
      cout << "Invalid grade" << endl;
   }
   cout << "Your grade is " << grade << endl;
   return 0;
}
```



Selection statements: nested if statement



```
if( boolean_expression 1)
{
   // Executes when the boolean expression 1 is true
   if(boolean_expression 2)
   {
      // Executes when the boolean expression 2 is true
   }
}
```