# Flow (if/else)

These are all forward-branches. The code flow progresses from top to bottom, but the if/else allows you to skip forward over instructions.

void main() {

int a; // 1: Skip initialization because ...

cin >> a; // 2: Read a value from the keyboard

// 3: if syntax. Keyword, expression, block of code

if(a == 4) {

cout << "a is 4" << endl;

}

cout << "Done" << endl;

}

Optional “else”:

if(a == 4) {

cout << "a is 4" << endl;

} else {

cout << "a is NOT 4" << endl;

}

You can chain them like this:

if(a == 4) {

cout << "a is 4" << endl;

} else if(a==5) {

cout << "a is 5" << endl;

} else {

cout << "a is something else" << endl;

}

Without the braces the if/else works with the next statement.

int b;

int c=1;

if(a == 4) b=1;

else b=2;c=0;

But use the braces. In this case “c” is always set to 0.

## Comparison Operators (evaluate to 1 or 0)

// == operator evaluates to 1 if things are the same or 0 if not

// != evaluates to 1 if things are different

// <, > less-than, greater-than

// <=, >= less-than-equal, greater-than-equal

## Logic Operators (evaluate to 1 or 0)

if(a==1 && b==2) {}

if(a==0 || b==0 || c==3) {

}

## Short Circuit

The language definition guarantees that expressions are evaluated left to right and short circuited (true or false) as soon as possible. In the above, if a==0 then b==0 and c==3 are never checked.

This is important for null pointer checks (see later).

## Complex Expressions

if(!(a==4 || a==5)) {} // The "!" is a NOT.

if( (doSomething(2)==4) && !(a==5 && b==6) ) {}

## Be Careful

int a=1;

if(a=0) {

cout << "a is 1" << endl;

}

## While and Do/While

Loops allow you to jump the code flow backwards … to go back and repeat a block of instructions. Use braces to define the block of code and an expression to control the looping.

int a=0;

while(a<=5) {

++a; // Start example without the ++

cout << "Here" << endl;

}

The expression is evaluated at the “top” of the loop (first thing). If it evaluates to non-zero then the block of code executes. At the bottom the loop comes back to the expression at the top.

The “do/while” moves the expression check to the bottom of the loop.

int a=0;

do {

++a;

cout << a << endl;

} while (a<=5);

The “do” loop is ALWAYS executed at least once. The WHILE loop may never execute at all.

We usually want a “do” loop but almost always use a “while” loop. We like to put the controlling expression at the top where we see it first and think about it first.

## Break and Continue

The “break” keyword takes the CPU to the next instruction after the bottom of the loop. It “breaks” the CPU out of the loop. The “continue” keyword takes the CPU to the top of the loop.

int a=0;

while(a<10) {

cout << a << endl;

++a;

if(a==5) {

break;

}

}

int a=0;

while(a<10) {

++a;

if(a==5) {

continue;

}

cout << a << endl;

}

## The (nearly) Infinite Loop

while(true) {

int a;

cin >> a;

cout << "You entered " << a << endl;

if(a<0) {

break;

}

}

## Program Ideas

* How do you figure out if a number is odd or even?
* Program to average numbers (floats) entered by user