Coding in C
Control Structures

Antonio Barbalace
antonio.barbalace@ed.ac.uk

University of Edinburgh

Index

- Conditional Control
  - if Statement
  - Conditional Expression
  - Switch Statement
- Control Loops
  - While Loop
  - Do .. While Loop
  - For Loop
- Non sequential control
  - Break
  - Continue
  - Goto

If Statements

```
    if (<expression>)
      <statement>
```

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

```
    if (<expression>) {
      <statement>
      <statement>
    }
    else {
      <statement>
      <statement>
    }
```

```
    if (<expression1>) {
      <statements>
    }
    else if (<expression2>) {
      <statements>
    }
```

- Both `if`, `if-else` and `if-else if` are available in C;
- The `<expression>` can be any valid expression;
- Parentheses around the expression are required even if it is just a single variable.

If Statements

```
if (<expression>)
  <statement>
```

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

```
if (<expression>) {
  <statement>
  <statement>
}
else {
  <statement>
  <statement>
}
```

```
if (<expression1>) {
  <statements>
}
else if (<expression2>) {
  <statements>
}
```

Example

Find the maximum of two values.

```
int x = 2;
int y = 10
int max = 0;

if (x > y) {
   max = x;
}
else {
   max = y;
}
```

if example

Conditional Expression

```
<expression1> ? <expression2> : <expression3>
```

- The conditional expression can be used as a shorthand for some `if-else` statements;
- This is an expression, not a statement, so **it represents a value**;
- The operator ? evaluates `<expression1>`:

  if it is true, it evaluates and returns `<expression2>`;

  otherwise, it evaluates and returns `<expression3>`.

Conditional Expression

```
<expression1> ? <expression2> : <expression3>
```

Example

Find the maximum of two values.

```
int x = 2;
int y = 10;
int max = 0;

max = (x > y) ? x : y;
```

conditional expression example

Switch Statement

```
switch (<expression>) {
  case <const-expression1>:
    <statement>
    break;
 case <const-expression2>:
    <statement>
    <statement>
    break;

 case <const-expression3>:
 case <const-expression4>:
    <statement>
    break;

  default:
    <statement>
}
```

- The switch statement is a sort of specialized form of `if`;
- The `switch` expression is evaluated and then the flow of control jumps to the matching `const-expression`;
- Each constant needs its own `case` keyword and a trailing colon (:);
- Once execution has jumped to a particular `case`, the program will keep running through all the cases from that point down;
- The explicit `break` statements are necessary to exit the switch.

University of Edinburgh

Switch Statement

```
switch (<expression>) {
  case <const-expression1>:
    <statement>
    break;
 case <const-expression2>:
    <statement>
    <statement>
    break;

 case <const-expression3>:
 case <const-expression4>:
    <statement>
    break;

  default:
    <statement>
}
```

Example

Set the output variable to 0 if input is 1, to 1 if input is 2 or 3, otherwise to -1.

```
int input = 11;
int output = 0;

switch(input) {
  case 1:
    output = 0;
    break;
  case 2:
  case 3:
    output = 1;
    break;
  default:
    output = -1;
}
```

switch example

While Loop

```
while (<expression>)
  <statement>

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

- In the `while` loop, the test expression is evaluated before each iteration.
- So `<statement>` may be executed zero times (if the condition is initially false);
- Parenthesis around the expression are required, as with `if`.

While Loop

```
while (<expression>)
  <statement>

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

Example 1

```
int i=0;
while (i < 100)
  i += 3;
// i is now 102
```

Example 2

```
int i=0;
while (i < 0) {
  i += 3;
}
// i is now 0
```

Compare these examples with the Do..While Loop ones.

Do..While Loop

```
do {
  <statement>
} while (<expression>)
```

```
do {
  <statement>
  <statement>
} while (<expression>)
```

- In the `do..while` loop, the test expression is evaluated at the end of each iteration;
- The loop body (`<statement>`) will be executed at least once, in any case.

Do..While Loop

```
do {
  <statement>
} while (<expression>)
```

```
do {
  <statement>
  <statement>
} while (<expression>)
```

Example 1

```
int i=0;
do {
  i += 3;
} while (i < 100);
// i is now 102
```

Example 2

```
int i=0;
do {
  i +=3;
} while (i < 0);
// i is now 3
```

Compare these examples with the While Loop ones.

For Loop

```
for (<initialization>; <continuation>; <action>)
  <statement>;

for (<initialization>; <continuation>; <action>) {
  <statement>
  <statement>
}
```

- It is the most general looping construct in C;
- The loop header contains three parts: initialization, continuation condition and action;
- The initialization is executed once before the body of the loop is entered;
- The loop continues to run as long as the continuation condition remains true;
- In each iteration of the loop, the action is executed.

For Loop

```
for (<initialization>; <continuation>; <action>)
  <statement>;

for (<initialization>; <continuation>; <action>) {
  <statement>
  <statement>
}
```

Example

Define an array of floating point values and sum its elements.

```
float values[5] = { 3.14f, 5.43f, 18.001f, 101.98f, 34.66f};
float summ = 0.0f;
int i;

for(i=0; i<5; i++)
   summ += values[i];
```

for example

Break

```
while (<expression>) {
  <statement>
  <statement>

  if (<condition>)
    break;

  <statement>
  <statement>
}
/* control jumps down here on
   the break */
```

break

- The `break` instruction will move control outside a loop or a `switch` statement;
- Stylistically speaking, it's preferable to use a straight `while` with a single test at the top whenever possible;
- Sometimes you are forced to use a `break`, because the test can occur only somewhere in the midst of the statements in the loop body.

Continue

```
while (<expression>) {
  <statement>
  <statement>

  if (<condition>)
    continue;

  <statement>
  <statement>
/* control jumps down here on
   the continue */
}
```

continue

- The `continue` instruction causes control to jump to the bottom of the loop, effectively skipping over all the loop body code that follows the `continue`;
- You can almost always get the same effect more clearly, using an if inside the loop.

Goto

```
<statements1>
goto <label>
<statements2>
<label>:
  /* control jump here after
   goto */
  <statements3>
```

goto

- Can be inserted everywhere in the code;
- `<label>` must exist locally (not in another function);
- “Real Programmers” don’t use the `goto`.

Exercises

- Write a program that counts the occurrence of the characters ‘a’, ’b’ and ‘e’ in the following sentence:

  "The overwhelming majority of program bugs and computer crashes stem from problems of memory access, allocation, or deallocation. Such memory-related errors are also notoriously difficult to debug.”