Part IV

Control Flow & Statements

Statements | if | ? | switch | while | do-while | for | return

Statements in C

Statements are **code fragments** that are **sequentially executed**.

- a *function body* is a *compound statement* which is a sequence of declarations and statements

Statement **types**:

(i) **compound** statement:
- brace-enclosed sequence of statements and declarations
- create block scopes

```
{ statement }
{ declaration...}
```

(ii) **expression** statement:
- most used statement type
- used for assignments or function calls

```
expression ;
;
```

(iii) **selection** statement:
- depending on expression value, a particular statement is executed while others are ignored

```
if(expression) statement
else statement
switch(expression)
statement
```

(iv) **iteration** statement:
- repeat the execution of a statement until a particular condition is met or does not hold

```
while(expression)
statement
do statement while
(expression);
for (init ;
 expression;
 expression)
statement
```

(v) **jump** statement:

```
break;
continue;
return expression;
return ;
goto identifier;
```

: http://en.cppreference.com/w/c/language/statements

`if` Statement

Branching in executed code depending on whether a condition (*expression*) holds.

```
if (expression)
  statement_true
```

```
if (expression)
  statement_true
else
  statement_false
```

If *expression* evaluates to non-zero, $statment_{true}$ is executed.
In case an `else` branch is given, $statement_{false}$ is executed otherwise.

taken from: http://en.cppreference.com/w/c/header

? Conditional Expressions

```
" if (expression1) "
  expression2
 else
  expression3
```

Conditional Expressions

```
expression1 ? expression2 : expression3
```

Conditional expression is alternative way to express conditional choice of expression. If $expression_1$ evaluates to non-zero, then the value of $expression_2$ is taken. The value of $expression_2$ otherwise.

Example

Compute minimum of two integers a and b.

```
int a = /* ... */,
    b = /* ... */,
    x;
if (a < b)
 x = a;
else
 x = b;
```

```
int a = /* ... */,
    b = /* ... */,
    x;
x = a < b ? a : b;
```

`switch` Statement

Used for multi-way decision to test whether a given expression value equals a particular constant-expression (i.e., integer constant)

- each $case_i$ is associated with one $constant\text{-}expression_i$
  - if expression value equals $constant\text{-}expression_i$, program execution continues at $statement_i$
  - cases are implicitly *goto* labels
    - important: if $case_i$ is taken, all other cases $case_j$ ($j > i$) are executed until `switch` statement ends or `break` is reached
      - `break` immediately exits the `switch` statement
- all cases must differ in their associated constants
  - but multiple cases can execute the same statement
- If expression value is unequal to all cases, $statement_{default}$ is executed
  - `default` case is optional
- The order in which cases and default are specified is up to the programmer; typically, `default` is the last one
- declarations inside a statement are not allowed unless a new block scope is introduced

```
switch (expression) {
  case constant-expression1:
    statement1
  case constant-expression2:
    statement2
  ...
  case constant-expressionn:
    statementn
  default: statementdefault
}
```

based on: http://en.cppreference.com/w/c/language/switch

switch Statement

Examples

```
enum gecko_days_e day = /* ... */
switch (day) {
   case gecko_d_mon:
   case gecko_d_tue:
   case gecko_d_wed:
     printf("Monday – Wednesday\n“);
     break;
   case gecko_d_thu:
     printf("Thursday\n");
     break;
   default:
     printf("All the rest\n");
     break;
}
```

based on: http://en.cppreference.com/w/c/language/switch

`while` Statement

Loops the execution of a statement until expression evaluates to zero.

- termination test is done *before* each iteration
  - if expression is non-zero but loop should exit, occurrence of `break` statement will exit the loop
- an immediately continue at the end of statement („new loop round“) happens if `continue` statement is given and hit

```
while (expression)
  statement
```

`do-while` Statement

Same semantic as while statement but

- termination test is done *after* each iteration
  - statement is executed at least one time

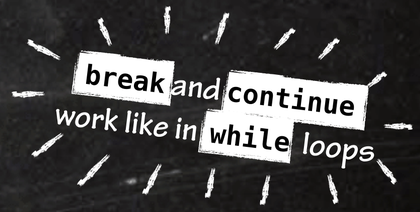
```
do statement
while (expression);
```

`for` Statement

```
for (init;
  expression_cond;
  expression_iteration)
statement
```

Shorter version of while loop: loops over statement

- *init* (optional) may be an
  - *expression* evaluated before first evaluation of $expression_{iteration}$ but which result is thrown away, or
  - *declaration* in scope of *init*, $expression_{cond}$, $expression_{iteration}$, and *statement*
    - only `auto` or `register` storage class is allowed
- $expression_{cond}$ (optional) evaluated for each iteration before *statement* is executed
  - if evaluation results in *zero*, *statement* is *not* executed and loop is exited
- $expression_{iteration}$ (optional) evaluated after execution of *statement* per iteration and result is thrown away
- control flow: *init* → $expression_{cond}$ —zero→ … , $expression_{cond}$ —non-zero→ *statement* —`break`→ … , *statement* → $expression_{iteration}$ → $expression_{cond}$, *statement* —`continue`→ $expression_{iteration}$



- if *init*, $expression_{cond}$, and $expression_{iteration}$ are all left out, it's an endless loop, *e.g.*, `for (;;) ;`

based on: http://en.cppreference.com/w/c/language/for

goto Statement

Performs an unconditional jump to the location indicated by *label*.

- location at which the control flow continues is prefixed with *label*:
  - *label* must be in same function as `goto`
    - otherwise see *non-local jumps* in standard library
- often used to break nested loops

```
goto label;

label: statement
```

```
for (/*...*/) {
 for (/*...*/) {
   if (/*...*/) {
     goto exit_loop;
   }
 }
}

exit_loop: /*...*/
```

(Example)

taken from: http://en.cppreference.com/w/c/header

return Statement

Exits the current function and returns to the caller:

- value of *expression* in case function has *non-void return type*
- nothing otherwise

```
return expression;
return ;
```

taken from: http://en.cppreference.com/w/c/header