

Week 3 Lecture 3

Control

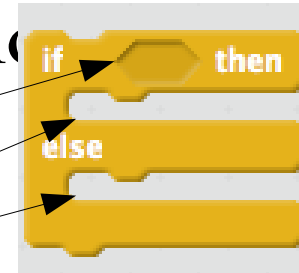
Statements and Expressions

- *Expressions* represent values. For example:
 - Variables: a, counter, avg_grade ...
 - Arithmetic: $1 + 2$, $a * 5$, $\text{total_grades} / \text{num_grades}$
 - Expression take an *operators*, such as $+$ and operands such as 1 and 2.
- *Statement* tell the computer when to do something. For example:
 - Selection: `if (a = 1) {printf(“%d”, a);}`
 - Iteration: `while (a = 1) {a = a + 1;}`

Selection and Iteration

- *Selection* chooses whether or not to perform an act (E.g., if then else)

- Basis of Choice
- Actions to choose



- *Iteration* repeats actions (E.g., repeat until)

- When to stop
- Action to repeat



Basis of Choice

- In C, the decision to execute an action in selection or to repeat an action in iteration is based *predicates*
 - A predicate is an expression that is either true or false. For example:
 - $1 > 2$
 - $1 < 2$
 - $1 == 2$
 - Predicates are constructed using relational operators
 - E.g., $>$, $<$, $==$, ...

Relational Operators

- Boolean value
 - True or false

Operator	Meaning
<code>a == b</code>	Equal
<code>a != b</code>	Not equal
<code>a > b</code>	Greater than
<code>a < b</code>	Less than
<code>a >= b</code>	Greater than or equal
<code>a <= b</code>	Less than or equal

Predicates may be combined

- Logical operators connect predicates
 - The logical operators are and, or and not
 - They take two predicates

Logical Operators

- Both a and b must be boolean values
 - Remember: booleans are ints
 - False is 0
 - True is any other number

Operat or	Meaning
!a	Not
a && b	And
a b	Or

Logical Expression Values

- Truth tables

&&	true	false
true	true	false
false	false	false

	true	false
true	true	true
false	true	false

!	
true	false
false	true

Example Logical Expressions

- $1 < 2 \ \&\& \ 2 < 3$ is true
- $1 < 2 \ || \ 2 < 3$ is true
- $1 > 2 \ || \ 2 < 3$ is true
- $1 > 2 \ \&\& \ 2 < 3$ is false
- $! (1 > 2)$ is true

Statements

- Tell the computer to do something
 - Have no value
 - Always followed by a ';'
- Putting a ';' after an expression turns it into a statement.
- An expression with followed by a ';' is a simple statement.

Block or Compound Statement

- {<declaration> <declaration> ... <statement>
<statement> ...}
 - i.e., a list of declarations (e.g. `int a;`) followed by a list of statements (e.g. `a = a + 1;`)
- Executed left-to-right, top to bottom.
 - Like reading English
- Local declarations are visible only in the block.
- e.g. `{int a; a = a + 1}`

Selection 1 (if then else)

- if (expression) then statement1
 - If expression is true, then do statement 1 otherwise skip statement 1.
- if (expression) then statement1 else statement2
 - If expression is true, then do statement 1 otherwise do statement 2.
- Either or both of statement1 and statement2 can be selection statements.

If then else example

```
#include <stdio.h>

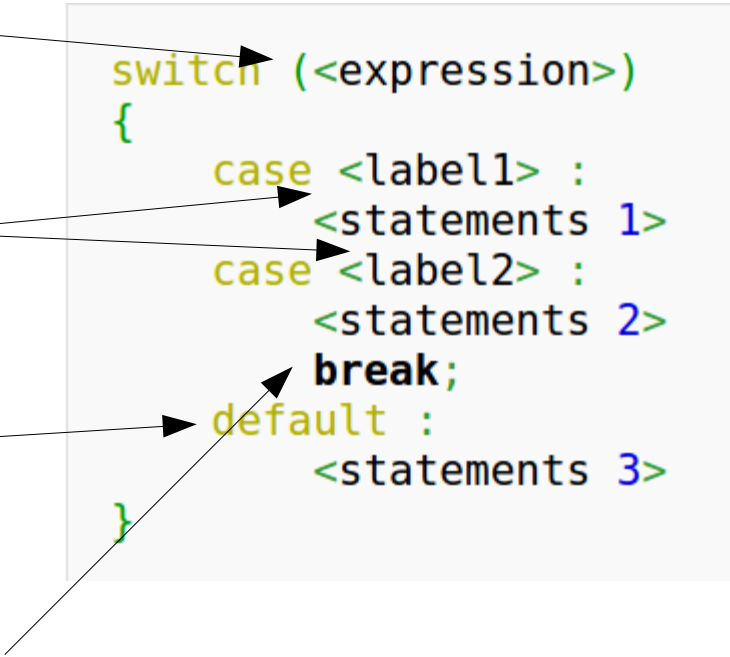
int main(int argv, char *argc[]) {
    char selector = 'x';

    printf("Enter selector> ");
    scanf("%c", &selector);
    if (selector == 'a') {
        printf("if a: selector = %c\n",
            selector);
    } else if (selector == 'b') {
        printf("else if b: selector = %c\n",
            selector);
    } else {
        printf("else: selector = %c\n",
            selector);
    }
    return 0;
}
if.c (END)
```

```
> gcc -o if if.c
> ./if
Enter selector> a
if a: selector = a
> ./if
Enter selector> b
else if b: selector = b
> ./if
Enter selector> c
else: selector = c
```

Selection 2 (switch)

- Tests `<expression>`
 - Must be char or int
- Starts at `<lable?>` that matches `<expression>`
- Starts at default: if no labels match.
- Exits execution on `break`.



```
switch (<expression>
{
    case <label1> :
        <statements 1>
    case <label2> :
        <statements 2>
        break;
    default :
        <statements 3>
}
```

The diagram illustrates the mapping between the bullet points and the syntax of a switch statement. Arrows point from the following elements to the code: from '`<expression>`' to the switch's opening parenthesis; from '`<lable?>`' to the first 'case' label; from 'if no labels match' to the 'default' label; and from '`break`' to the 'break;' statement.

From Wikipedia: C_syntax

Switch example

```
#include <stdio.h>

int main(int argv, char *argc[]) {
    char selector = 'x';

    printf("Enter selector> ");
    scanf("%c", &selector);
    switch (selector)
    {
        case 'a' :
            printf("case a: selector = %c\n",
                selector);
        case 'b' :
            printf("case b: selector = %c\n",
                selector);
            break;
        default :
            printf("default: selector = %c\n",
                selector);
    }
    return 0;
}
```

switch.c (END)

```
> gcc -o switch switch.c
> ./switch
Enter selector> a
case a: selector = a
case b: selector = a
> ./switch
Enter selector> b
case b: selector = b
> ./switch
Enter selector> c
default: selector = c
```

Iteration (for)

- `for (<init>; <test>; <increment>) statement`
 - Init: initialize the loop before the first time through
 - Test: boolean expression, if true execute statement
 - Increment: expression evaluated after the statement
- e.g., `for (int i=0; i<5; i++) {a[i]=i;}`
 - Given an array `int a[5]`; this statement initialized the elements of the array to 0, 1, 2, 3, 4.

For example

```
#include <stdio.h>

int main(int argv, char *argv[]) {
    const int array_len = 5;
    int array[5] = {};

    printf("array == { ");
    for (int i = 0; i < array_len; i++) {
        printf ("%d ", array[i]);
    }
    printf("}\n");

    for (int i = 0; i < array_len; i++) {
        array[i] = i;
        printf(" i: %d%s", i,
               (i == array_len-1) ? "\n" : ", ");
    }

    printf("array == { ");
    for (int i = 0; i < array_len; i++) {
        printf ("%d ", array[i]);
    }
    printf("}\n");
    return 0;
}
for.c (END)
```

```
> gcc -std=c11 -o for for.c
> ./for
array == { 0 0 0 0 0 }
i: 0, i: 1, i: 2, i: 3, i: 4
array == { 0 1 2 3 4 }
```

Iteration (while)

- While is syntactic sugar
 - while (<expression>) <statement>
- Means the same thing as
 - for (; <expression>;) <statement>

Example (while)

```
#include <stdio.h>

int is_same_string (char *s1, char *s2) {
    while (*s1 == *s2 && *s1 != '\0' && *s2 != '\0') {
        s1++;
        s2++;
    }
    return *s1=='\0' && *s2=='\0';
}

int main(int argv, char *argc[]) {
    char passwd[80] = "xxx";

    while (!is_same_string(passwd, "nat")) {
        printf("Password: ");
        scanf("%s", passwd);
    }

    return 0;
}
while.c (END)
```

```
> gcc -std=c11 -o while while.c
> ./while
Password: foo
Password: bar
Password: baz
Password: nat
```

Iteration (do while)

- Do while is (mostly) syntactic sugar
 - `<stmnt> while (<expr>);`
- Puts the test after the statement
 - The statement is done at least once.
- Means (mostly) the same thing as
 - `for (; ; <stmnt> if <expr> break;);`
 - But `<stmnt> if <expr> break;` is illegal in a for statement

Example (do while)

```
#include <stdio.h>

int is_same_string (char *s1, char *s2) {
    while (*s1 == *s2 && *s1 != '\0' && *s2 != '\0') {
        s1++;
        s2++;
    }
    return *s1=='\0' && *s2== '\0';
}

int main(int argv, char *argc[]) {
    char passwd[80] = "xxx";

    do {
        printf("Password: ");
        scanf("%s", passwd);
    } while (!is_same_string(passwd, "nat"));

    return 0;
}
```

```
> gcc -std=c11 -o do-while do-while.c
> ./do-while
Password: foo
Password: bar
Password: baz
Password: nat
```