ESC101N
Fundamentals of Computing

Arnab Bhattacharya
arnabb@iitk.ac.in

Indian Institute of Technology, Kanpur
http://www.iitk.ac.in/esc101/

1st semester, 2010-11

Tue, Wed, Fri 0800-0900 at L7

Statements and blocks

- A statement is an expression followed by a semicolon (;)
  - `i = 12 + 6 / 3;`
  - Assignment (=) is an expression
  - `i++;`
- A series of statements grouped together using braces ({}) is a block of statements or a compound statement

```
{
   i = 5;
   i++;
}
```

- A block of statements is treated as a single statement

Scope of a variable

- Part of a program where a variable can be used is called its scope
- Scope is the *statement block* where it is declared
- Scope includes
  - All statements in the current block
  - All *inner* blocks, i.e., blocks contained inside the current block

Scope of a variable

- Part of a program where a variable can be used is called its scope
- Scope is the *statement block* where it is declared
- Scope includes
  - All statements in the current block
  - All *inner* blocks, i.e., blocks contained inside the current block
- Error in line 5 as `i` is not visible outside the block

```
1: {
2:    int i = 5;
3:    i--;
4: }
5: i++;
```

Scope of a variable

- Part of a program where a variable can be used is called its scope
- Scope is the *statement block* where it is declared
- Scope includes
  - All statements in the current block
  - All *inner* blocks, i.e., blocks contained inside the current block
- Error in line 5 as `i` is not visible outside the block

```
1: {
2:    int i = 5;
3:    i--;
4: }
5: i++;
```

- All right as `i` is visible in all inner blocks

```
1: int i = 5;
2: {
3:    i++;
4: }
5: i--;
```

`if` statement

- Decision making
- Find the minimum of two integers
- Algorithm
  1. Compare the two integers `x` and `y`
  2. If `x < y`, then `min = x`
  3. Otherwise, `min = y`

`if` statement

- Decision making
- Find the minimum of two integers
- Algorithm
  1. Compare the two integers `x` and `y`
  2. If `x < y`, then `min = x`
  3. Otherwise, `min = y`
- To capture the above logic in C, `if` statements are used

if statement

- Decision making
- Find the minimum of two integers
- Algorithm
  1. Compare the two integers `x` and `y`
  2. If `x < y`, then `min = x`
  3. Otherwise, `min = y`
- To capture the above logic in C, `if` statements are used

```
if (condition)
{
   statements1
}
else
{
   statements2
}
```

- Entire `if else` is a single statement

Example

- Find the minimum of two integers
- Algorithm
  1. Compare the two integers `x` and `y`
  2. If `x < y`, then `min = x`
  3. Otherwise, `min = y`
- C code

```
#include <stdio.h>
int main()
{
  int x, y;
  int min;
  scanf(``%d'', &x);
  scanf(``%d'', &y);
  if (x < y)
    min = x;
  else
    min = y;
  printf(``Minimum is %d\n'', min);
}
```

Understanding `if`

- `condition` must evaluate to a boolean value
- When it is true, the `if` part is executed
- Otherwise (i.e., when it is false), the `else` part is executed
- All numbers, characters, etc. are treated as booleans
- Any *expression* fits as `condition`

```
if (5 - 3)
   ...
```

evaluates to

Understanding `if`

- `condition` must evaluate to a boolean value
- When it is true, the `if` part is executed
- Otherwise (i.e., when it is false), the `else` part is executed
- All numbers, characters, etc. are treated as booleans
- Any *expression* fits as `condition`

```
if (5 - 3)
   ...
```

evaluates to *true*

```
if (5 - 5)
   ...
```

evaluates to

Understanding `if`

- `condition` must evaluate to a boolean value
- When it is true, the `if` part is executed
- Otherwise (i.e., when it is false), the `else` part is executed
- All numbers, characters, etc. are treated as booleans
- Any *expression* fits as `condition`

```
if (5 - 3)
   ...
```

evaluates to *true*

```
if (5 - 5)
   ...
```

evaluates to *false*

More syntax

- Block of statements may be used in `if` and `else` part

```
if (condition)
{
  statement1
  statement2
}
else
{
  statement3
  statement4
}
```

- Since block of statements is equivalent to a single statement, the above is really the same
- Important: `else` part may be omitted

Nested if

- else with more than one previous if is ambiguous

```
if ((x + y) > 0)
  if (x < y)
    printf(``x is minimum'');
  else
    printf(``y is minimum'');
```

Nested if

- else with more than one previous if is ambiguous

```
if ((x + y) > 0)
  if (x < y)
    printf(``x is minimum'');
  else
    printf(``y is minimum'');
```

- Rule: else is associated with nearest else-less if
- Comment: Indenting program correctly helps in understanding (as shown in previous code snippet)

Nested if

- else with more than one previous if is ambiguous

```
if ((x + y) > 0)
  if (x < y)
    printf(``x is minimum'');
  else
    printf(``y is minimum'');
```

- Rule: else is associated with nearest else-less if
- Comment: Indenting program correctly helps in understanding (as shown in previous code snippet)
- Use braces if intended otherwise

```
if ((x + y) > 0)
{
  if (x < y)
    printf(``x is minimum'');
}
else
  printf(``x + y is negative'');
```

else if statement

- Testing more than two conditions can be done using `else if`

```
if (x < 0)
  printf(``Negative'');
else
  if (x > 0)
    printf(``Positive'');
  else
    printf(``Zero'');
```

is equivalent to

```
if (x < 0)
  printf(``Negative'');
else if (x > 0)
  printf(``Positive'');
else
  printf(``Zero'');
```

Multiple else if

- Consider

```
if (section == 1)
  printf(``TB101'');
else if (section == 2)
  printf(``TB102'');
else if (section == 15)
  printf(``WL226'');
else
  printf(``Wrong section'');
```

- Multiple else if statements are better written using switch case
- switch case works only when the same variable is tested for equality against different values

```
switch (section)
{
  case 1: printf(``TB101''); break;
  case 2: printf(``TB102''); break;
  case 15: printf(``WL226''); break;
  default: printf(``Wrong section''); break;
}
```

switch case statement

```
switch (variable)
{
  case value1: statements1; break;
  case value2: statements2; break;
  ...
  default: statementsn; break;
}
```

- default is executed when variable evaluates to none of the other values in case

switch case statement

```
switch (variable)
{
  case value1: statements1; break;
  case value2: statements2; break;
  ...
  default: statementsn; break;
}
```

- default is executed when variable evaluates to none of the other values in case
- Important: Without break, next case is also executed

```
switch (x)
{
  case 0: printf(``0'');
  case 1: printf(``1'');
  default: printf(``2'');
}
```

- When x is 0, all of 0, 1 and 2 are printed
- When x is 1, both 1 and 2 are printed

switch case without break

- switch case without break is useful when same statement needs to be executed for multiple cases
- Suppose there are two sections 1 and 2 on Monday, two sections 3 and 4 on Tuesday, and others on Wednesday
- Output the day based on input section

```
switch (section)
{
  case 1: ;
  case 2: printf(``Monday'');  break;
  case 3: ;
  case 4: printf(``Tuesday'');  break;
  default: printf(``Wednesday'');  break;
}
```