Control Flow, Functions and Basic Linkage

MATH 5061: Fundamentals of Computer Programming for Scientists and Engineers

Dr. Richard Berger

richard.berger@temple.edu

Department of Mathematics
Temple University

10/13/2016

TEMPLE UNIVERSITY

Outline

Decision and Control Flow Statements

- `if`- and `else`-Statement
- Conditional Expressions
- `switch`-Statement
- `while`-Statement
- `for`-Statement
- `do`−`while`-Statement

Functions

- Declaration and Definition
- `void` type
- Examples

Compilation - Part 2: Basic Linkage

TEMPLE UNIVERSITY

Outline

Decision and Control Flow Statements

- **`if`**- and **`else`**-Statement
- Conditional Expressions
- **`switch`**-Statement
- **`while`**-Statement
- **`for`**-Statement
- **`do`**−**`while`**-Statement

Functions

- Declaration and Definition
- **`void`** type
- Examples

Compilation - Part 2: Basic Linkage

TEMPLE UNIVERSITY

`if` Statement

- executing a block of code if a condition is **`true`**

Python

```python
if condition:
    statement1
```

```python
if condition:
    statement1
    statement2
    statement3
```

C++

```cpp
if (condition) {
    statement1;
}
```

```cpp
if (condition) {
    statement1;
    statement2;
    statement3;
}
```

TEMPLE UNIVERSITY

`else` Statement

- **`else`**-statement is only executed if the condition of the **`if`**-statement is **`false`**

Python

```
if condition:
    statementA
else:
    statementB
```

C++

```
if (condition)
    statementA;
else
    statementB;
```

else Statement

Python

```
if condition:
    statementA1
    statementA2
    statementA3
else:
    statementB1
    statementB2
    statementB3
```

C++

```
if (condition) {
    statementA1;
    statementA2;
    statementA3;
} else {
    statementB1;
    statementB2;
    statementB3;
}
```

Dangling `else`

```
if (condition)
    if (condition2)
        if (condition3)
            statementA;
else
    statementB;
```

```
if (condition)
    if (condition2)
        if (condition3)
            statementA;
    else
        statementB;
```

```
if (condition)
    if (condition2)
        if (condition3)
            statementA;
        else
            statementB;
```

- These 3 variants are the same in C++
- In Python indentation rules would make them different
- In C++ the parsing rules state that the `else` *belongs* to the inner-most `if`-statement
- **Recommendation:** Always define a block using curly braces, even for single line `if`-statements

Dangling else

```
if (condition) {
    if (condition2) {
        if (condition3) {
            statementA;
        } else {
            statementB;
        }
    }
}
```

```
if (condition) {
    if (condition2) {
        if (condition3) {
            statementA;
        }
    }
} else {
    statementB;
}
```

- Using curly braces makes intent explicit
- Might seem like more typing, but worth the effort vs. looking for the bug later

Common mistake if you don’t use curly braces

```
if (condition)
    statementA1;
```

- now extend with an additional statement

```
if (condition)
    statementA1;
    statementA2;
```

TEMPLE UNIVERSITY

Common mistake if you don't use curly braces

```
if (condition)
     statementA1;
```

- now extend with an additional statement

```
if (condition)
     statementA1;
     statementA2;
```

you wanted this

```
if (condition) {
     statementA1;
     statementA2;
}
```

Common mistake if you don't use curly braces

```
if (condition)
     statementA1;
```

- now extend with an additional statement

```
if (condition)
     statementA1;
     statementA2;
```

you wanted this

```
if (condition) {
     statementA1;
     statementA2;
}
```

but compiler will see this

```
if (condition) {
     statementA1;
}
statementA2;
```

TEMPLE UNIVERSITY

Conditional Expressions

```
if (a > 0)
    b = a;
else
    b = -a;
```

Alternative:

```
b = (a > 0) ? a : -a;
```

$expr_1$? $expr_2$: $expr_3$;

- an expression which evaluates to one of two alternatives based on a condition

Alternative to `if`-cascades

```
if (expr == constA) {
   ...
} else if(expr == constB) {
   ...
} else if(expr == constC) {
   ...
} else {
   ...
}
```

Alternative to `if`-cascades

```
if (expr == constA) {
  ...
} else if(expr == constB) {
  ...
} else if(expr == constC) {
  ...
} else {
  ...
}
```

```
switch(expr) {
case constA:
    ...
    break;

case constB:
    ...
    break;

case constC:
    ...
    break;

default:
    ...
}
```

- the **`switch`** statement takes an expression and jumps to the case which matches its value
- **`case`**s have to be constants like integer literals or character literals

TEMPLE UNIVERSITY

switch Statement - with integer literals

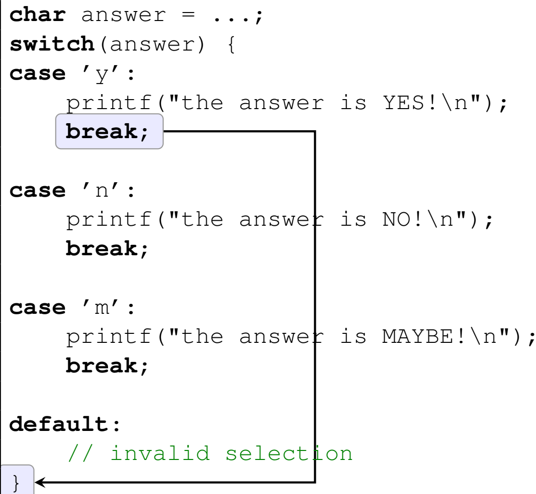
```
int dayOfWeek = ...;

switch(dayOfWeek) {
case 0:
    printf("Monday");
    break;

case 1:
    printf("Tuesday");
    break;

case 2:
    printf("Wednesday");
    break;

default:
    // in all other cases do this
}
```

switch Statement - with character literals

```
char answer = ...;
switch(answer) {
case 'y':
    printf("the answer is YES!\n");
    break;

case 'n':
    printf("the answer is NO!\n");
    break;

case 'm':
    printf("the answer is MAYBE!\n");
    break;

default:
    // invalid selection
}
```

- each **case** defines a constant and ends with a **break**

switch Statement - with character literals



- each **case** defines a constant and ends with a **break**
- after the **break**-statement execution continues after the switch statement

switch Statement - with character literals

```
char answer = ...;
switch(answer) {
case 'y':
    printf("the answer is YES!\n");
    break;

case 'n':
    printf("the answer is NO!\n");
    break;

case 'm':
    printf("the answer is MAYBE!\n");
    break;

default:
    // invalid selection
}
```

- each **case** defines a constant and ends with a **break**
- after the **break**-statement execution continues after the switch statement
- the optional **default** case is reached if none of the **cases** match

switch Statement - combining **cases**

```
char answer = ...;
switch(answer) {
case 'y':
case 'Y':
    printf("the answer is YES!\n");
    break;

case 'n':
case 'N':
    printf("the answer is NO!\n");
    break;

default:
    // invalid selection
}
```

- if a **break** statement is missing between **case**s, execution continues with the next **case**.
- No further comparison is made, execution just continues.

switch Statement - combining **cases**

```
char answer = ...;
switch(answer) {
case 'y':
case 'Y':
    printf("the answer is YES!\n");
    break;

case 'n':
case 'N':
    printf("the answer is NO!\n");
    break;

default:
    // invalid selection
}
```

- if a **break** statement is missing between **case**s, execution continues with the next **case**.
- No further comparison is made, execution just continues.

switch Statement - combining cases

```
char answer = ...;
switch(answer) {
case 'y':
case 'Y':
    printf("the answer is YES!\n");
    break;

case 'n':
case 'N':
    printf("the answer is NO!\n");
    break;

default:
    // invalid selection
}
```

- if a **break** statement is missing between **case**s, execution continues with the next **case**.
- No further comparison is made, execution just continues.

TEMPLE UNIVERSITY

while loop

Python

```
while condition:
    statement
```

```
while condition:
    statement1
    statement2
    statement3
```

C++

```
while (condition)
    statement;
```

```
while (condition) {
    statement1;
    statement2;
    statement3;
}
```

TEMPLE UNIVERSITY

Example: Temperature Table

- Let's write a program which generates a table showing the celsius temperature for fahrenheit values from 0 to 300 in increments of 20 degrees.

$$^{\circ}C = \frac{5}{9}(^{\circ}F - 32)$$

```
F    C
0    -17
20   -6
40   4
60   15
80   26
100  37
120  48
140  60
160  71
180  82
200  93
220  104
240  115
260  126
280  137
300  148
```

Live Demo: Example 1a

Example: Temperature Table

```c
#include <stdio.h>

int main() {

  int lower = 0;   // lower limit of table
  int upper = 300; // upper limit
  int step  = 20;  // step size

  int fahrenheit = lower;
  printf("F\tC\n");

  while (fahrenheit <= upper) {
    int celsius = 5*(fahrenheit - 32) / 9;
    printf("%d\t%d\n", fahrenheit, celsius);
    fahrenheit += step;
  }

  return 0;
}
```

```
F    C
0    -17
20   -6
40   4
60   15
80   26
100  37
120  48
140  60
160  71
180  82
200  93
220  104
240  115
260  126
280  137
300  148
```

Live Demo: Example 1b

TEMPLE UNIVERSITY

Example: Temperature Table using floats

```c
#include <stdio.h>

int main() {

  int lower = 0;    // lower limit of table
  int upper = 300;  // upper limit
  int step  = 20;   // step size

  float fahrenheit = lower;
  printf("F\tC\n");

  while (fahrenheit <= upper) {
    float celsius = 5.0*(fahrenheit - 32.0) / 9.0;
    printf("%d\t%f\n", fahrenheit, celsius);
    fahrenheit += step;
  }

  return 0;
}
```

```
F    C
0.000000    -17.777779
20.000000   -6.666667
40.000000   4.444445
60.000000   15.555555
80.000000   26.666666
100.000000  37.777779
120.000000  48.888889
140.000000  60.000000
160.000000  71.111115
180.000000  82.222221
200.000000  93.333336
220.000000  104.444443
240.000000  115.555557
260.000000  126.666664
280.000000  137.777771
300.000000  148.888885
```

Controlling formatting in `printf`

- Placeholders can be further configured
- You can specify how wide a placeholder should be. By default it will add spaces when necessary

  %3d Print decimal number so that it is at least 3 characters wide, insert spaces if necessary

  %03d Print decimal number so that it is at least 3 characters wide and insert leading zeros if necessary

  %6.1f Print floating point number so that it is at least 6 characters wide and have 1 number after the decimal point

Live Demo: Example 1c

Example: Formatted Temperature Table

```
#include <stdio.h>

int main() {

  int lower = 0;    // lower limit of table
  int upper = 300;  // upper limit
  int step  = 20;   // step size

  int fahrenheit = lower;
  printf("F\tC\n");

  while (fahrenheit <= upper) {
    float celsius = 5.0*(fahrenheit - 32.0) / 9.0;
    printf("%3d %6.1f\n", fahrenheit, celsius);
    fahrenheit += step;
  }

  return 0;
}
```

```
F   C
  0  -17.8
 20   -6.7
 40    4.4
 60   15.6
 80   26.7
100   37.8
120   48.9
140   60.0
160   71.1
180   82.2
200   93.3
220  104.4
240  115.6
260  126.7
280  137.8
300  148.9
```

TEMPLE UNIVERSITY

Looping over number ranges

```
int i = 0;

while (i < 10) {
    printf("%d\n", i);
    ++i;
}
```

```
0
1
.
.
8
9
```

- very common in programming, which is why there is a special loop for it

Looping over number ranges

```
int i = 0;

while (i < 10) {
    printf("%d\n", i);
    ++i;
}
```

```
0
1
.
.
8
9
```

- very common in programming, which is why there is a special loop for it

```
for (int i = 0; i < 10; ++i ) {
    printf("%d\n", i);
}
```

TEMPLE UNIVERSITY

`for` Statement

Regular usage:

```
for ( init ; condition ; increment ) {
    // loop body
}
```

General form:

```
for ( expr1 ; expr2 ; expr3 )
{
    // loop body
}
```

```
expr1;
while ( expr2 )
{
    // loop body
    expr3;
}
```

`for` Statement

Loop with 10 iterations

```
for (int i = 0; i < 10; ++i ) {
    printf("%d\n", i);
}
```

Output

for Statement

Loop with 10 iterations

Output

1

```
for (int i = 0; i < 10; ++i ) {
    printf("%d\n", i);
}
```

for Statement

Loop with 10 iterations

2

```
for (int i = 0; i < 10; ++i ) {
    printf("%d\n", i);
}
```

Output

TEMPLE UNIVERSITY

`for` Statement

Loop with 10 iterations

```
for (int i = 0; i < 10; ++i ) {
    printf("%d\n", i);
}
```

3

Output

0

TEMPLE UNIVERSITY

`for` Statement

Loop with 10 iterations

4

```
for (int i = 0; i < 10; ++i) {
    printf("%d\n", i);
}
```

Output

```
0
```

`for` Statement

Loop with 10 iterations

5

```
for (int i = 0; i < 10; ++i ) {
    printf("%d\n", i);
}
```

Output

```
0
```

TEMPLE UNIVERSITY

for Statement

Loop with 10 iterations

6

```
for (int i = 0; i < 10; ++i ) {
    printf("%d\n", i);
}
```

Output

0

1

for Statement

Loop with 10 iterations

7

```
for (int i = 0; i < 10; ++i) {
    printf("%d\n", i);
}
```

Output

```
0

1
```

`for` Statement

Loop with 10 iterations

```
for (int i = 0; i < 10; i++ ) {
    printf("%d\n", i);
}
```

Loop with an increment of 2

```
for (int i = 0; i < 10; i += 2 ) {
    printf("%d\n", i);
}
```

TEMPLE UNIVERSITY

Live Demo: Example 1d

Example: Formatted Temperature Table using **for**-loop

```
#include <stdio.h>

int main() {

  int lower = 0;    // lower limit of table
  int upper = 300;  // upper limit
  int step  = 20;   // step size

  printf("F\tC\n");

  for (int fahrenheit = lower; fahrenheit <= upper; fahrenheit += step) {
    float celsius = 5.0*(fahrenheit - 32.0) / 9.0;
    printf("%3d %6.1f\n", fahrenheit, celsius);
  }

  return 0;
}
```

TEMPLE UNIVERSITY

do–while Loop

```
do {
    statementA1;
    statementA2;
    statementA3;
    statementA4;
} while (condition);
```

- Body of the loop is executed at least once
- Loop condition is only evaluated at the end

TEMPLE UNIVERSITY

Example: number guessing game

```
Guess a number between 1 and 100
Enter a number: 50
nope... the number is higher
Enter a number: 70
nope... the number is higher
Enter a number: 90
nope... the number is higher
Enter a number: 99
nope... the number is lower
Enter a number: 95
nope... the number is lower
Enter a number: 92
Congratz! You found it!
```

Live Demo: Example 2

Example: number guessing game

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    srand(time(0)); // seed random number generator with current time

    int secret_value = (rand() % 100) + 1;   // create a random integer between 1 and 100
    int guess = 0;

    printf("Guess a number between 1 and 100\n");

    do {
        printf("Enter a number: ");
        scanf("%d", &guess);

        if (guess < secret_value) {
            printf("nope... the number is higher\n");
        } else if (guess > secret_value) {
            printf("nope... the number is lower\n");
        }
    } while(guess != secret_value);

    printf("Congratz! You found it!\n");
    return 0;
}
```

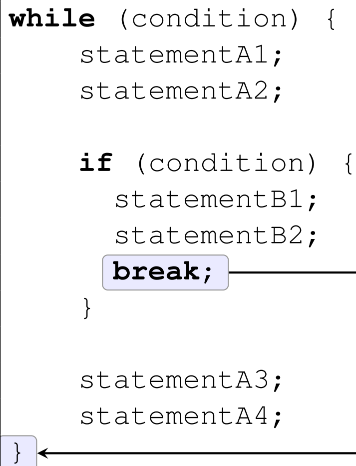
break Statement

```
while (condition) {
    statementA1;
    statementA2;

    if (condition) {
      statementB1;
      statementB2;
      break;
    }

    statementA3;
    statementA4;
}
```

Escape from the inside loop.

TEMPLE UNIVERSITY

break Statement



```
while (condition) {
    statementA1;
    statementA2;

    if (condition) {
      statementB1;
      statementB2;
      break;
    }

    statementA3;
    statementA4;
}
```

Escape from the inside loop.

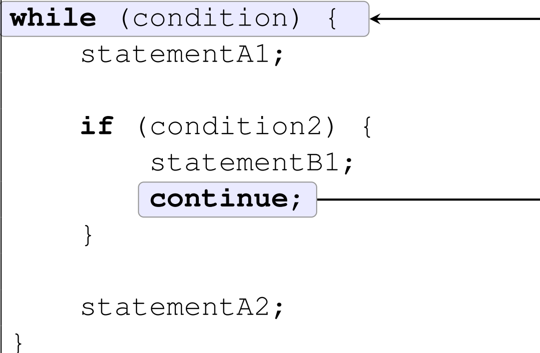
TEMPLE UNIVERSITY

`continue` Statement

```
while (condition) {
    statementA1;

    if (condition2) {
        statementB1;
        continue;
    }

    statementA2;
}
```

Abort the current iteration of the loop and continue with the next

continue Statement



```
while (condition) {
    statementA1;

    if (condition2) {
        statementB1;
        continue;
    }

    statementA2;
}
```

Abort the current iteration of the loop and continue with the next

Infinite Loops

- these are loops which do not stop unless you **`break`** or **`return`** from them

```
while (true) {
    ...
}
```

```
for ( ; ; ) {
    ...
}
```

Outline

- Decision and Control Flow Statements
  - `if`- and `else`-Statement
  - Conditional Expressions
  - `switch`-Statement
  - `while`-Statement
  - `for`-Statement
  - `do−while`-Statement
- Functions
  - Declaration and Definition
  - `void` type
  - Examples
- Compilation - Part 2: Basic Linkage

Functions and the DRY Principle

- Functions are a way to organize and structure your code
- DRY = Don't Repeat Yourself
- Avoid needless duplication of code by extracting common code into functions. This has multiple effects:
  - it makes your code shorter
  - less code, less possibility for bugs
  - with good function names, you create more readable code
  - your code becomes reusable

Declaration and Definition of Functions

```
double  rect_area ( double w, double h )
{
    return w * h;
}
```

TEMPLE UNIVERSITY

Declaration and Definition of Functions

function signature

```
double  rect_area ( double w, double h )
{
    return w * h;
}
```

TEMPLE UNIVERSITY

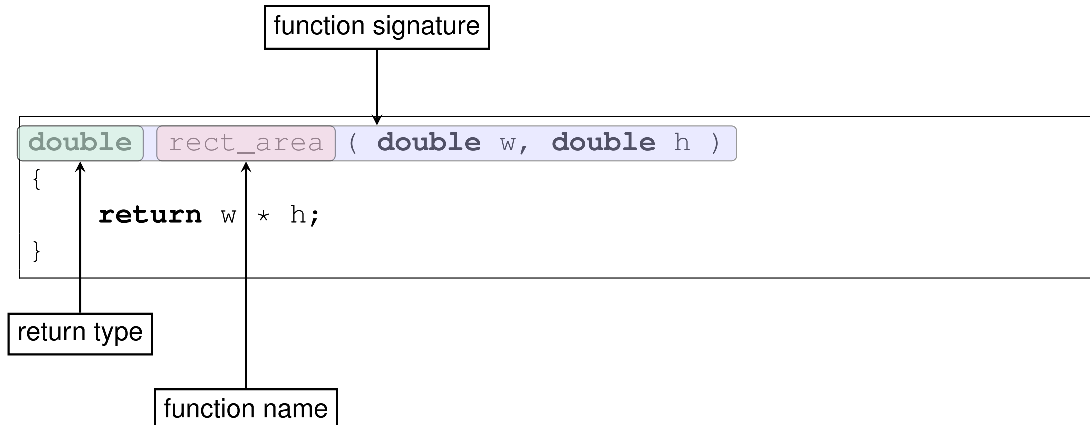
Declaration and Definition of Functions

function signature

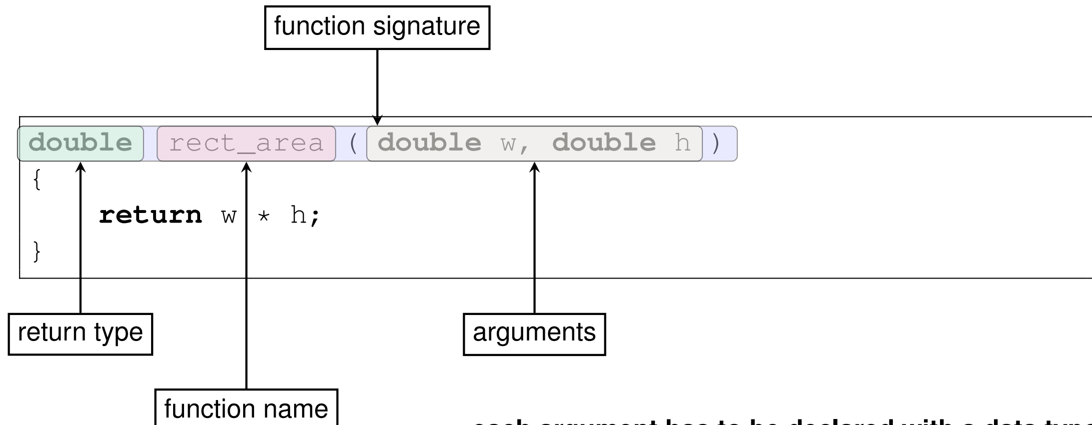
```
double rect_area ( double w, double h )
{
    return w * h;
}
```

return type

Declaration and Definition of Functions



Declaration and Definition of Functions



each argument has to be declared with a data type

TEMPLE UNIVERSITY

Declaration and Definition of Functions

```
double  rect_area ( double w, double h )
{
    return w * h;
}
```

the return value must be compatible with the return type

void Type

- functions which do not return anything, but only execute code can be declared with a special return type of **void**
- function declared as **void** can use a simple **return;** statement without any value.
- a return in a **void** function is optional

Example: print_vector function

```
void print_vector(float x, float y) {
    printf("(%f, %f)", x, y);
    return; // optional
}
```

Declaration and Definition of Functions

A: Define functions first

```
#include <stdio.h>

void print_vector(float x, float y) {
    printf("(%f, %f)\n", x, y);
}

int main() {
    print_vector(1.0f, 0.0f);
    print_vector(0.0f, 1.0f);
    return 0;
}
```

Declaration and Definition of Functions

A: Define functions first

```c
#include <stdio.h>

void print_vector(float x, float y) {
    printf("(%f, %f, %f)\n", x, y);
}

int main() {
    print_vector(1.0f, 0.0f);
    print_vector(0.0f, 1.0f);
    return 0;
}
```

This does not compile

```c
#include <stdio.h>

int main() {
    print_vector(1.0f, 0.0f);
    print_vector(0.0f, 1.0f);
    return 0;
}

void print_vector(float x, float y) {
    printf("(%f, %f)\n", x, y);
}
```

TEMPLE UNIVERSITY

Declaration and Definition of Functions

A: Define functions first

```c
#include <stdio.h>

void print_vector(float x, float y) {
    printf("(%f, %f, %f)\n", x, y);
}

int main() {
    print_vector(1.0f, 0.0f);
    print_vector(0.0f, 1.0f);
    return 0;
}
```

B: Declare function and define later

```c
#include <stdio.h>

void print_vector(float x, float y);

int main() {
    print_vector(1.0f, 0.0f);
    print_vector(0.0f, 1.0f);
    return 0;
}

void print_vector(float x, float y) {
    printf("(%f, %f)\n", x, y);
}
```

Example: Temperature Conversion functions

- let's create functions which do the conversion °C ⇔ °F for us
- this is a small reusable piece of code

```
float fahrenheit2celsius(float fahrenheit);
float celsius2fahrenheit(float celsius);
```

Example: Temperature Conversion functions

```
float fahrenheit2celsius(float fahrenheit) {
    return 5.0*(fahrenheit - 32.0) / 9.0;
}

float celsius2fahrenheit(float celsius) {
    return (celsius * 9.0 / 5.0) + 32.0;
}
```

Live Demo: Rewrite Temperature Table Example

Outline

Decision and Control Flow Statements

- `if`- and `else`-Statement
- Conditional Expressions
- `switch`-Statement
- `while`-Statement
- `for`-Statement
- `do`−`while`-Statement

Functions

- Declaration and Definition
- `void` type
- Examples

Compilation - Part 2: Basic Linkage

TEMPLE UNIVERSITY

Example: Temperature Converter

- Let’s write a small utility which converts any temperature value
- User is first asked what units should be used (enter ’c’ or ’C’ and ’f’ or ’F’)
- Then asks for a temperature and converts it

Live Demo: Example 3

temp_converter.cpp - Part 1 (Incomplete)

```
#include <stdio.h>

char ask_for_units();

int main() {
    char units = ask_for_units();
    float value, converted;

    printf("Enter the temperature (in %c): ", units);
    scanf("%f", &value);
    if (units == 'C') {
        converted = celsius2fahrenheit(value);
        printf("%6.1f C = %6.1f F\n", value, converted);
    } else { // units == 'F'
        converted = fahrenheit2celsius(value);
        printf("%6.1f F = %6.1f C\n", value, converted);
    }
    return 0;
}
```

temp_converter.cpp - Part 2

```
char ask_for_units() {
    char selected_unit;
    while(true) {
        printf("Please select unit (c or C / f or F): ");
        scanf(" %c", &selected_unit);
        switch(selected_unit) {
            case 'c':
            case 'C':
                return 'C';
            case 'f':
            case 'F':
                return 'F';
            default:
                printf("invalid input!\n");
        }
    }
}
```

Working with multiple source files

- We already implemented the conversion functions for one program
- By moving them into their own source file we can reuse that code in another application

  temp_converter.cpp main logic of the program
  temp_utilities.cpp utility functions for conversion

TEMPLE UNIVERSITY

Live Demo: Create temp_utilities.cpp

Working with multiple source files

- When we compile the program we have to compile two source files

```
g++ -o converter temp_converter.cpp temp_utilities.cpp
```

TEMPLE UNIVERSITY

Working with multiple source files

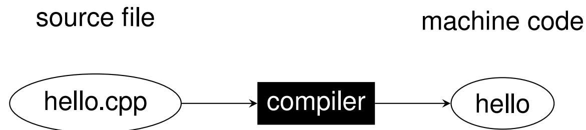
- When we compile the program we have to compile two source files

```
g++ -o converter temp_converter.cpp temp_utilities.cpp
```

However we get this compile error:

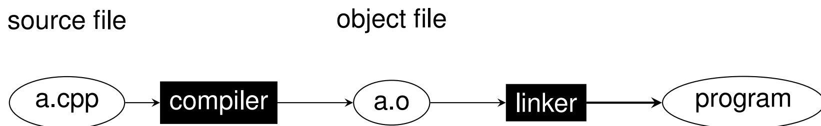
```
temp_converter.cpp: In function 'int main()':
temp_converter.cpp:14:45: error: 'celsius2fahrenheit' was
                              not declared in this scope
        converted = celsius2fahrenheit(value);
                                             ^
temp_converter.cpp:18:45: error: 'fahrenheit2celsius' was
                              not declared in this scope
        converted = fahrenheit2celsius(value);
```

Recall: Compilation



- a compiler translates C++ code (text) into machine code (binary)
- each source file corresponds to one translation unit

Compilation is a Multi-Step process



- each source file corresponds to one translation unit
- a compiler translates C++ code (text) into machine code (binary)
- each translation unit becomes one **object file**
- an object file contains machine code, but has placeholders for symbols (names) it expects from libraries or other translation units
- a separate program, the **linker**, finds these symbols, replaces the placeholders and generates the final program

TEMPLE UNIVERSITY

Compilation Steps

Single command:

```
g++ -o hello hello.cpp
```

Separate Steps:

```
g++ -o hello.o -C hello.cpp
g++ -o hello hello.o
```

Compiling multiple files

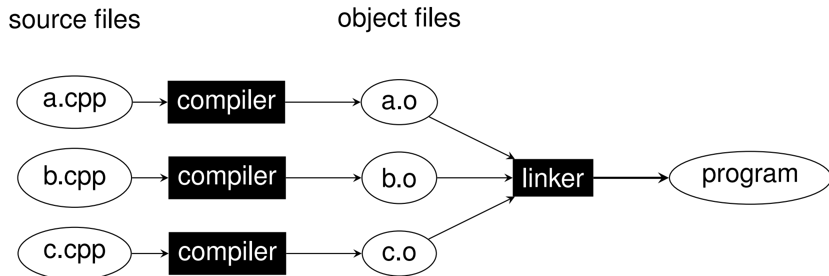
- If you compile multiple files, each is compiled separately

source file		object file
a.cpp	compiler	a.o
b.cpp	compiler	b.o
c.cpp	compiler	c.o

TEMPLE UNIVERSITY

Linking

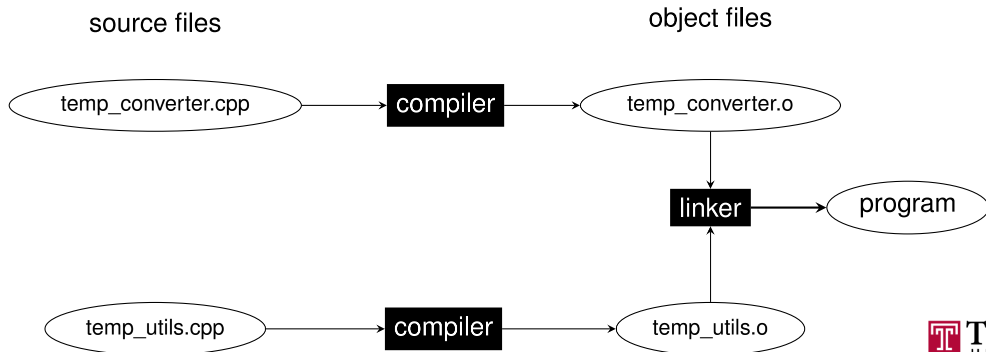
Object files are finally combined by the linker to create an executable program



TEMPLE UNIVERSITY

Example: Temperature Converter

- In our example each file is compiled separately
- The compiler of `temp_converter.cpp` does not *see* the symbols (names) of the other file
- This is why we have to declare these functions in `temp_converter.cpp` if we want to use them



TEMPLE UNIVERSITY

Header Files: sharing declarations

- while we can simply write the function declarations into the source files, it is common to move declarations of functions shared with the outside world in its own *header file*
- header files usually have the same name as their .cpp file and end with .h
- so in our case, since our functions are defined in temp_utils.cpp, we define a header temp_utils.h which only contains the function declarations
- the benefit of headers is that any other source file can simply **#include** them if needed

Note

- local headers are included with **#include** "file.h"
- system/library headers are included with **#include** <file.h>

Live Demo: Example 3 Completion

temp_utilities.h

```
float celsius2fahrenheit(float celsius);
float fahrenheit2celsius(float fahrenheit);
```

temp_utilities.cpp

```
float fahrenheit2celsius(float fahrenheit) {
    return 5.0*(fahrenheit - 32.0) / 9.0;
}

float celsius2fahrenheit(float celsius) {
    return (celsius * 9.0 / 5.0) + 32.0;
}
```

temp_converter.cpp - Part 1 (Completed)

```
#include <stdio.h>
#include "temp_utils.h"

char ask_for_units();

int main() {
    char units = ask_for_units();
    float value, converted;

    printf("Enter the temperature (in %c): ", units);
    scanf("%f", &value);
    if (units == 'C') {
        converted = celsius2fahrenheit(value);
        printf("%6.1f C = %6.1f F\n", value, converted);
    } else { // units == 'F'
        converted = fahrenheit2celsius(value);
        printf("%6.1f F = %6.1f C\n", value, converted);
    }
    return 0;
}
```

Example: Compilation Steps

Single command:

```
g++ -o converter temp_converter.cpp temp_utils.cpp
```

Separate Steps:

```
# create temp_converter.o
g++ -C temp_converter.cpp

# create temp_utils.o
g++ -C temp_utils.cpp

# create final program 'converter'
g++ -o converter temp_converter.o temp_utils.o
```