§4 Functions

ENGG1111

Computer Programming and Applications

Dirk Schnieders

Outline

- Top-down design approach
- Function
- Local and global variables
- Scope of variables

Top-down design approach

Top-Down Design Approach

- A good way to design a program is to break down the task to be accomplished into a few sub-tasks
- Each sub-task might be further decomposed into smaller sub-tasks, and this process is repeated until all sub-tasks are small enough that their implementations become manageable
- This approach is called top-down design (aka divide and conquer)

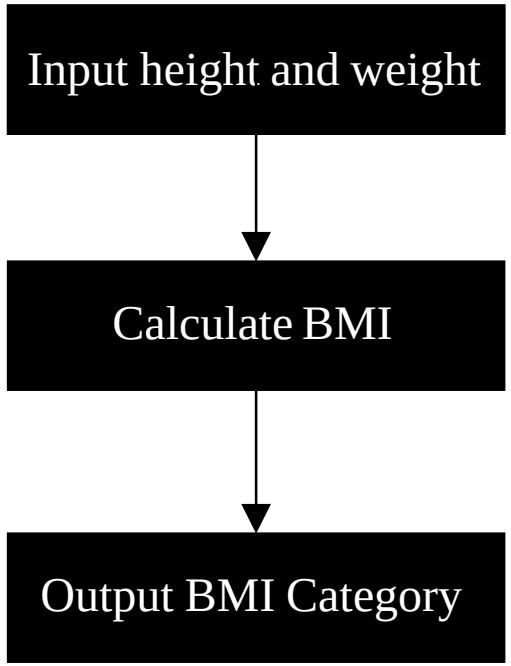
Example - BMI

- Write a program that reads in the weight (in kg) and height (in meter) of a user, and outputs the Body Mass Index (BMI)
  - BMI = weight / height^2

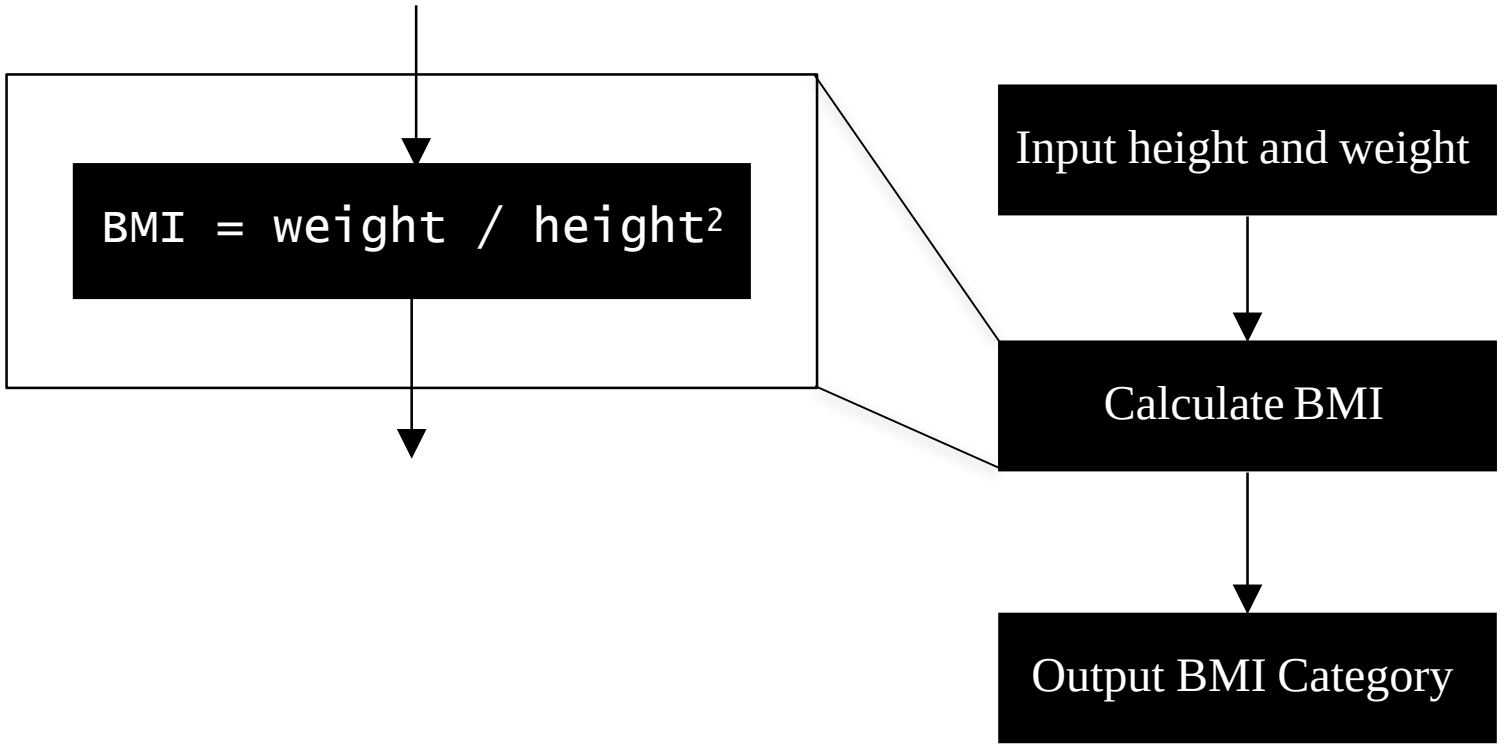
BMI	Category
<18.5	Underweight
[18.5, 22.9]	Normal
>22.9	Overweight

Example - BMI

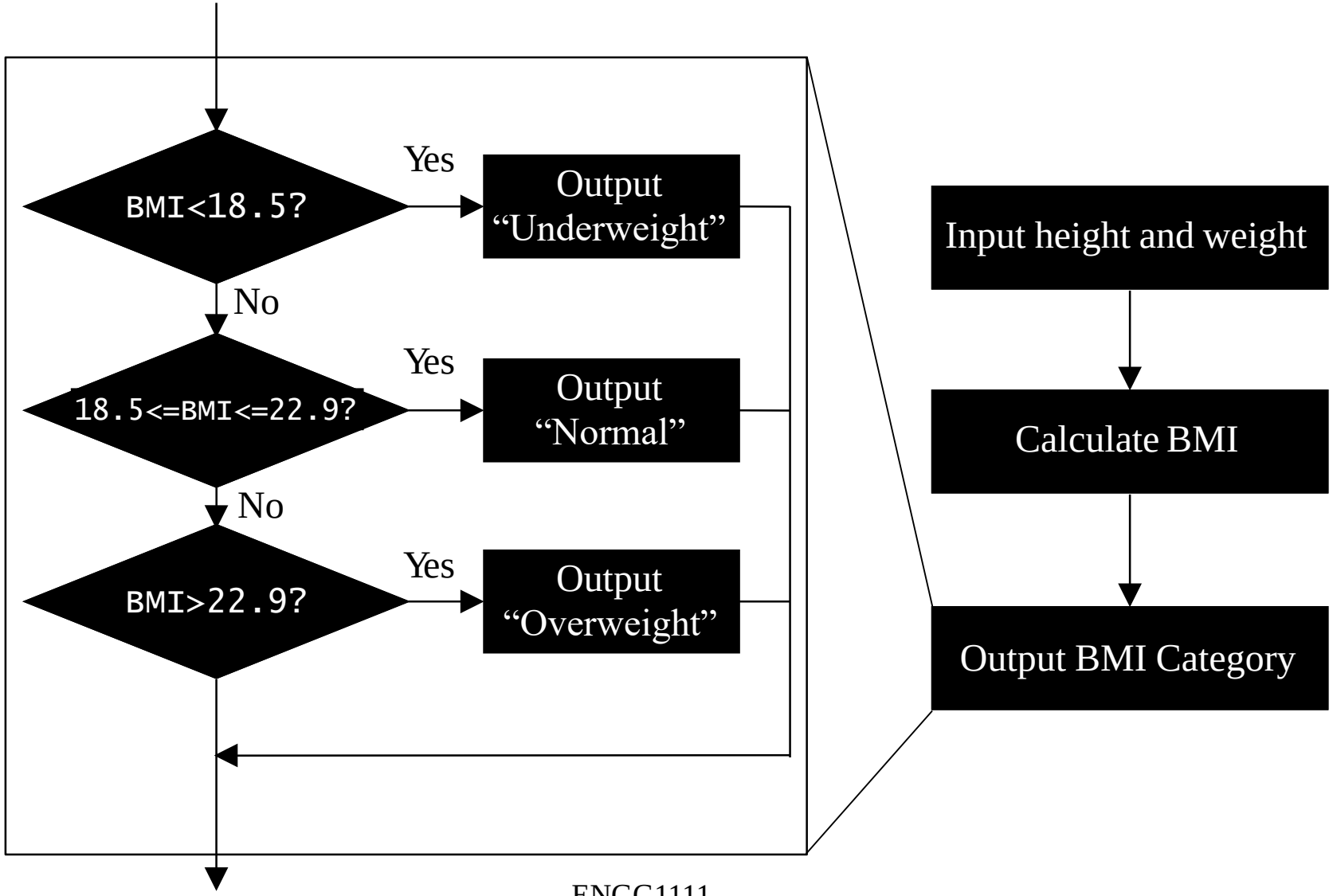
- We tackle the problem by dividing our program into three sub-tasks
- We will treat each part as a smaller problem and tackle (conquer) the smaller problems one by one



Example - BMI


BMI = weight / height²
Input height and weight
Calculate BMI
Output BMI Category

Example - BMI


BMI<18.5?
Yes
Output "Underweight"
No
18.5<=BMI<=22.9?
Yes
Output "Normal"
No
BMI>22.9?
Yes
Output "Overweight"
Input height and weight
Calculate BMI
Output BMI Category

Function

Functions

- Preserving the top-down design structure in a program will make it easier to understand, write and change the program
- In C++ sub-tasks can be implemented as functions
  - A function is a group of statements that is executed when it is called from some point in program
- A program is composed of a collection of functions
- When a program is put into execution, it always starts at the main function, which may in turn call other functions

Function Definition

- Describes how a function computes the value it returns
- Consists of a function header followed by a function body
  - The function header specifies
    - the type of the return value
    - the function name (identifier), and
    - the list of parameters (with types and identifiers)
  - The function body consists of variable statements enclosed within a pair of braces { }

header →
body →

```
type_ret func_name(type_1 par_1, type_2 par_2, ...) {
  // statements ...
}
```

type of the return value

type of first parameter

name of first parameter

```
type_ret func_name(type_1 par_1, type_2 par_2, ...) {
  // statements ...
}
```

return statement

- The value returned by the function is determined when the function executes a return statement
- A return statement consist of the keyword return followed by an expression
- When a return statement is executed the function call ends
- Note that a function may contain more than one return statement

Applications functions02.cpp — ~/co... 04 - File Manager 10:40 engg1111

functions02.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions02.cpp

```cpp
#include <iostream>
using namespace std;
double celsiusToFahrenheit(double celsius) {
  return 9.0/5*celsius+32;
}
int main() {
}
```

functions02.cpp 8:1 LF UTF-8 C++ 0 files

return statement

- Note that the main function returns the value 0 if a return statement is missing

Applications functions03.cpp — ~/co... 04 - File Manager 10:55 engg1111

functions03.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions03.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      cout << "Hello World!" << endl;
5      return 0;
6  }
7
```

functions03.cpp 7:1 LF UTF-8 C++ 0 files

Function Call

- A function call (i.e., the process of calling a function) is made using the function name with the necessary parameters
- A function call is itself an expression, and can be put in any places where an expression is expected

functions04.cpp — ~/code/04 — Atom

```cpp
#include <iostream>
using namespace std;
double celsiusToFahrenheit(double celsius) {
  return 9.0/5*celsius+32;
}
int main() {
  cout << celsiusToFahrenheit(28.0) << endl;
}
```

functions04

```
82.4
Press any key to continue...
```

Parameters and Arguments

- Parameters are variables that are part of the function definition
- The expressions used to pass values to a function during a function call are referred to as arguments
- When a function is called the computer substitutes the first argument with the first parameter, the second argument with the second parameter, and so forth

Applications functions05.cpp — ~/co... 04 - File Manager 11:23 engg1111

functions05.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions05.cpp

```cpp
#include <iostream>
using namespace std;
int test(int a, int b, int c) {
   cout << a <<" "<< b <<" "<< c << endl;
   return a+b+c;
}
int main() {
   cout << test(1, 3, 2) << endl;
}
```

functions05.cpp 9:2 LF UTF-8 C++ 0 files

Task

- What is the output of the following program?

functions07.cpp — ~/code/04 — Atom

```cpp
#include <iostream>
using namespace std;
char test(int a) {
  if (a<10)
    return 'A';
  else
    return 'B';
}
int main() {
  cout << test(5) << endl;
}
```

Arguments

- The arguments used in a function call can be constants, variables, expressions, or even function calls

functions06.cpp — ~/code/04 — Atom

```cpp
#include <iostream>
using namespace std;
int test(int a, int b, int c) {
   cout << a <<" "<< b <<" "<< c << endl;
   return a+b+c;
}
int main() {
   cout << test(test(1, 1, 1), 3-1, 2) << endl;
}
```

void Functions

- In some situations, a function returns no value
- In this case, we use the void type specifier
- A function with no return value is called a void function
- The return statement in a void function does not specify any return value
  - It is used to return the control to the calling function
- If a return statement is missing in a void function, the control will be returned to the calling function after the execution of the last statement in the function

Applications functions08.cpp — ~/co... functions08 04 - File Manager 11:51 engg1111

functions08.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions08.cpp

```
1  #include <iostream>
2  using namespace std;
3  void test() {
4     cout << "test" << endl;
5  }
6  int main() {
7     test();
8  }
9
```

functions08.cpp 9:1

functions08

```
test
Press any key to continue...
```

Applications functions09.cpp — ~/c... functions08 04 - File Manager functions09 11:52 engg1111

functions09.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions09.cpp

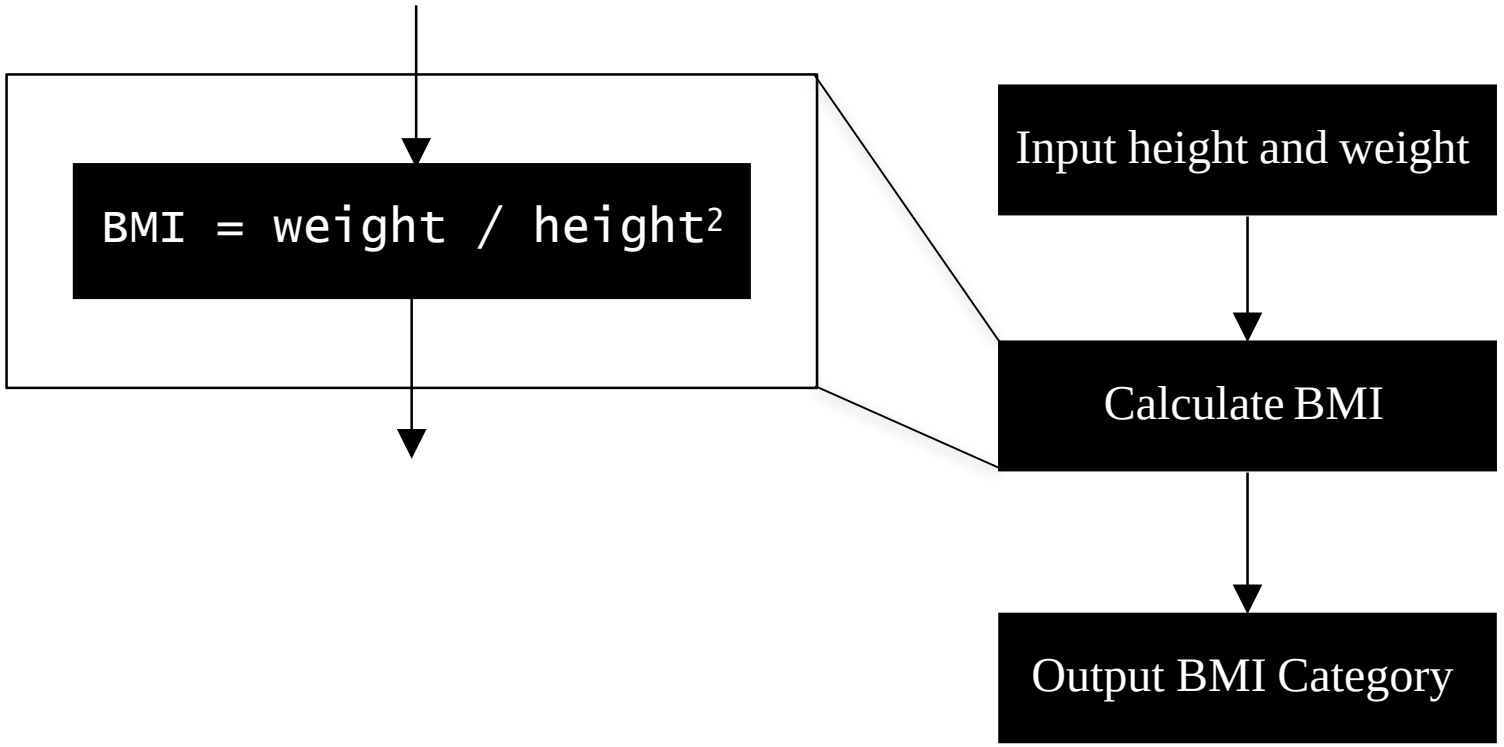
```cpp
#include <iostream>
using namespace std;
void test() {
  cout << "test" << endl;
  return;
  cout << "test" << endl;
}
int main() {
  test();
}
```

functions09.cpp 5:10

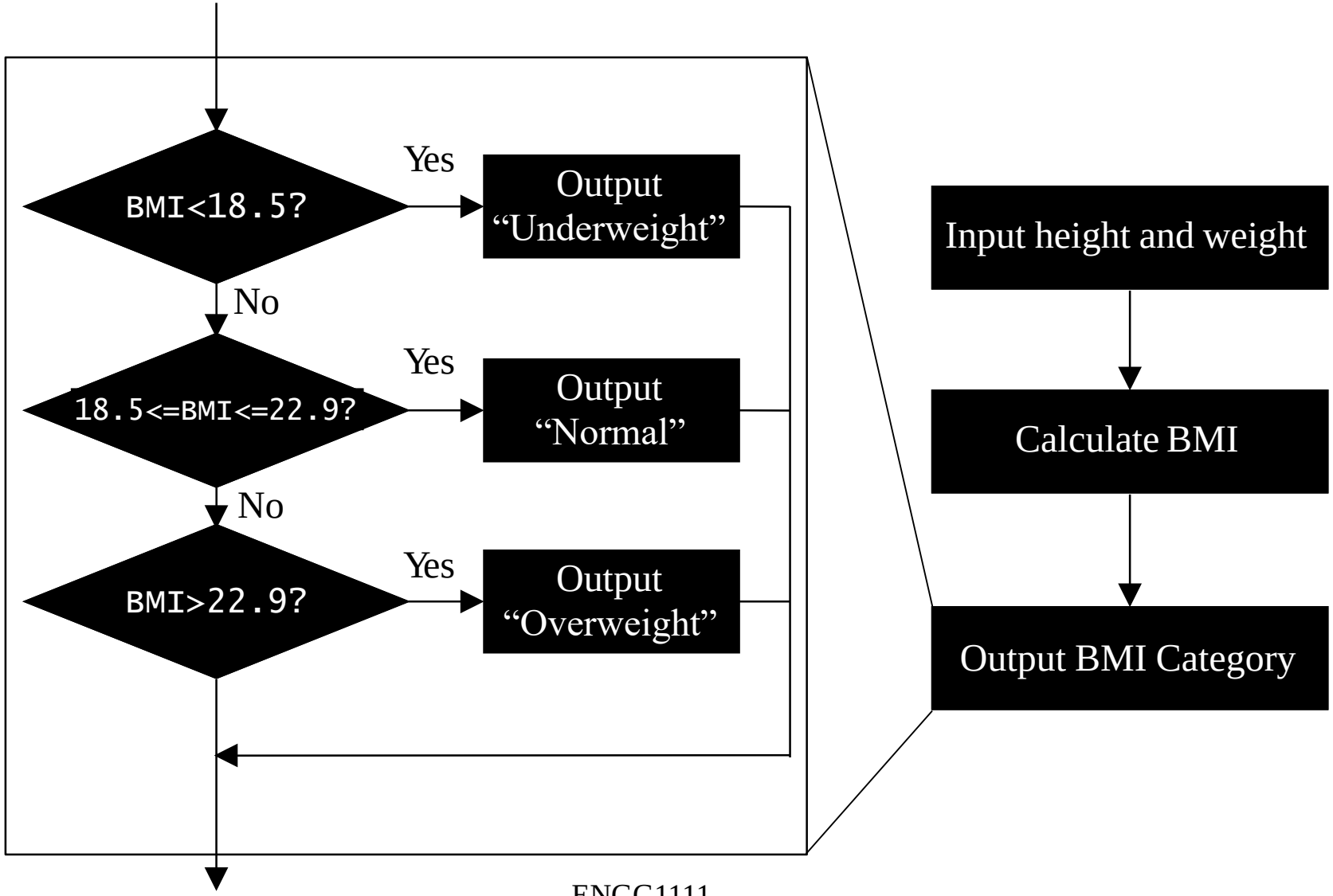
functions09

```
test
Press any key to continue...
```

calculateBMI()



outputCategory()



Applications functions10.cpp — ~/co... 04 - File Manager 12:16 engg1111

functions10.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions10.cpp

```cpp
#include <iostream>
using namespace std;
double calculateBMI(double height, double weight) {
  return weight / (height*height);
}
void outputCategory(double bmi) {
  if (bmi<18.5)
    cout << "Underweight" << endl;
  if (bmi >= 18.5 && bmi <= 22.9)
    cout << "Normal" << endl;
  if (bmi > 22.9)
    cout << "Overweight" << endl;
}
int main() {
  double height, weight;
  cin >> height >> weight;
  double bmi = calculateBMI(height, weight);
  outputCategory(bmi);
}
```

functions10.cpp 19:2 LF UTF-8 C++ 0 files

Function Definition

- A function must be defined before the function call is made

Applications functions11.cpp — ~/co... 04 - File Manager 12:18 engg1111

functions11.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions11.cpp

```
#include <iostream>
using namespace std;
void test() {

}
int main() {
  test();
}
```

gcc-make-run: Running Command... Close All

```
"g++" -pedantic-errors -std=c++11 "functions11.cpp" -o "functions11"
```

gcc-make-run: Build Success

functions11.cpp 8:2 LF UTF-8 C++ 0 files

Applications functions12.cpp — ~/co... 04 - File Manager 12:20 engg1111

functions12.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions12.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
  test();
}
void test() {

}
```

gcc-make-run: Running Command... Close All

```
"g++" -pedantic-errors -std=c++11 "functions12.cpp" -o "functions12"
```

gcc-make-run: Compile Error

```
functions12.cpp: In function ‘int main()’:
functions12.cpp:4:3: error: ‘test’ was not declared in this scope
   test();
   ^~~~
functions12.cpp:4:3: note: suggested alternative: ‘tzset’
   test();
   ^~~~
   tzset
```

functions12.cpp 8:2 LF UTF-8 C++ 0 files

Function Definition

- A function must be defined before the function call is made
  - Solution 1
    - Place the function definition before the function call in the source file
  - Solution 2
    - The function definition can be placed anywhere in the source file by including just the function declaration before the function call

Applications functions13.cpp — ~/co... 04 - File Manager 13:26 engg1111

functions13.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions13.cpp

```cpp
#include <iostream>
using namespace std;
void test();
int main() {
  test();
}
void test() {

}
```

gcc-make-run: Running Command... Close All

```
"g++" -pedantic-errors -std=c++11 "functions13.cpp" -o "functions13"
```

gcc-make-run: Build Success

functions13.cpp 9:3 LF UTF-8 C++ 0 files

Local and Global Variables

Local Variables

- Variables declared within a function, including parameters, are local to that particular function
  - No other function can have access to them
- Local variables in a function come into existence only when the function is called, and disappear when the function is exited
- Local variables declared within the same function must have unique identifiers, whereas local variables of different functions may use the same identifier

Applications functions14.cpp — ~/co... functions14 04 - File Manager 13:33 engg1111

functions14.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions14.cpp

```cpp
#include <iostream>
using namespace std;
void test() {
  int a = 7;
  cout << a << endl;
}
int main() {
  int a = 77;
  test();
  cout << a << endl;
}
```

functions14.cpp 11:2

functions14

```
7
77
Press any key to continue...
```

Global Variables

- Variables may also be declared outside all functions
  - Such variables are called global variables
    - They can be accessed by all functions
- Global variables remain in existence permanently
  - Retain their values even after the functions that set their values have returned
- Could be used instead of arguments to communicate data between functions
- **You must avoid using global variables**
  - Since the values of global variables can be changed by several functions they are very hard to trace

Applications functions15.cpp — ~/co... functions15 04 - File Manager 13:40 engg1111

functions15.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions15.cpp

```cpp
#include <iostream>
using namespace std;
int g = 1;
void doSomething() {
  g = 2;
}
int main() {
  g = 1;
  doSomething();
  // Programmer expects g to be 1
  // doSomething() changed it to 2
  if (g == 1) cout << "No threats detected" << endl;
  else cout << "Launch nuclear missiles" << endl;
}
```

functions15

```
Launch nuclear missiles
Press any key to continue...
```

functions15.cpp C++ 0 files

Applications functions16.cpp — ~/co... 04 - File Manager 13:47 engg1111

functions16.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions16.cpp

```cpp
#include <iostream>
using namespace std;
int g = 1;
void doSomething() {
  int g = 2;
}
int main() {
  g = 1;
  doSomething();
  // Programmer expects g to be 1
  // doSomething() changed it to 2
  if (g == 1) cout << "No threats detected" << endl;
  else cout << "Launch nuclear missiles" << endl;
}
```

functions16.cpp 5:7 LF UTF-8 C++ 0 files

Scope of Variables

Scope of Variables

- The scope of a variable is the portion of a program in which the variable can be used
  - A variable cannot be used beyond its scope
- The scope of a local variable starts from its declaration up to the end of the block
  - A block is delimited by a pair of braces { }
  - Variables declared in outer blocks can be referred to in an inner block
- Variables can be declared with the same identifier as long as they have different scopes
  - Variables in an inner block will hide any identically named variables in outer blocks

Applications functions17.cpp — ~/co... 04 - File Manager 14:06 engg1111

functions17.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions17.cpp

```cpp
#include <iostream>
using namespace std;
int a;
void test() {
  {
    int b;
  }
  int c;
}
int main() {
  int d;
  {
    int e;
  }
}
```

functions17.cpp 7:4 LF UTF-8 C++ 0 files

Applications functions18.cpp — ~/co... functions18 04 - File Manager 14:23 engg1111

functions18.cpp — ~/code/04 — Atom

File Edit View Selection Find Packages Help

functions18.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
  int i = 0;
  cout << "Outer block: i = " << i << endl;
  {
    int i = 100;
    cout << "Inner block: i = " << i << endl;
  }
  cout << "Outer block: i = " << i << endl;
}
```

functions18.cpp* 2:21 LF UTF-8 C++ 0 files