

Functions

Victor Eijkhout and Charlie Dey

spring 2017

Function basics

Turn blocks of code into functions

- Code fragment with clear function:
- Turn into *subprogram*: function *definition*.
- Use by single line: function *call*.

Function definition and call

```
for (int i=0; i<N; i++) {    void report_evenness(int n) {
    cout << i;                cout << i;
    if (i%2==0)               if (i%2==0)
        cout << " is even";   cout << " is even";
    else                       else
        cout << " is odd";     cout << " is odd";
    cout << endl;             cout << endl;
}                              }

...
int main() {
    ...
    for (int i=0; i<N; i++)
        report_evenness(i);
}
```

Why functions?

- Easier to read
- Shorter code: reuse
- Maintenance and debugging

Prime function

Using a function, primality testing would look like:

```
bool isprime;  
number = 13;  
isprime = prime_test_function(number);
```

Anatomy of a function definition

- Result type: what's computed. `void` if no result
- Name: make it descriptive.
- Arguments: zero or more.
- Body: any length.
- Return statement: usually at the end, but can be anywhere; the computed result.

Program with function

```
#include <iostream>
using namespace std;

int twice_function(int n) {
    int twice_the_input = 2*n;
    return twice_the_input;
}

int main() {
    int number = 3;
    cout << "Twice three is: " << twice_function(number) << endl;
    return 0;
}
```


Function call

The function call

1. causes the function body to be executed, and
2. the function call is replaced by whatever you return.
3. (If the function does not return anything, for instance because it only prints output, you declare the return type to be void.)

Functions without input, without return result

```
void print_header() {  
    cout << "*****" << endl;  
    cout << "* Output      *" << endl;  
    cout << "*****" << endl;  
}  
  
int main() {  
    print_header();  
    cout << "The results for day 25:" << endl;  
    // code that prints results ....  
    return 0;  
}
```

Functions with input

```
void print_header(int day) {
    cout << "*****" << endl;
    cout << "* Output      *" << endl;
    cout << "*****" << endl;
    cout << "The results for day " << day << ":" << endl;
}

int main() {
    print_header(25);
    // code that prints results ....
    return 0;
}
```

Functions with return result

```
#include <cmath>
double pi() {
    return 4*atan(1.0);
}
```

Scope

Function body is a *scope*: local variables.

No local functions.

Project Exercise 1

Write a function that takes an integer input, and return a boolean corresponding to whether the input was prime.

```
bool isprime;  
isprime = prime_test_function(13);
```

Read the number in, and print the value of the boolean.

Project Exercise 2

Take the prime number testing program, and modify it to read in how many prime numbers you want to print. Print that many successive primes. Keep a variable `number_of_primes_found` that is increased whenever a new prime is found.

Scope

Function body is a *scope*: local variables.

No local functions.

Parameter passing

Mathematical type function

Pretty good design:

- pass data into a function,
- return result through `return` statement.
- Parameters are copied into the function.
- *call by value*

Results other than through return

Also good design:

- Return no function result,
- or return (0 is success, nonzero various informative statuses), and
- return other information by changing the parameters.
- *call by reference*

Call by reference example

```
int can_read_value( int &value ) {  
    int file_status = try_open_file();  
    if (file_status==0)  
        value = read_value_from_file();  
    return file_status;  
}
```

Exercise 3

Write a function `swap` of two parameters that exchanges the input values:

```
int i=2,j=3;  
swap(i,j);  
// now i==3 and j==2
```

Recursion

Functions are allowed to call themselves, which is known as *recursion*.

```
int factorial( const int n ) {  
    if (n==1)  
        return 1;  
    else  
        return n*factorial(n-1);  
}
```

Exercise 4

Write a recursive function for computing Fibonacci numbers:

$$F_0 = 1, \quad F_1 = 1, \quad F_{n+2} = F_n + F_{n+1}$$

Default arguments

Functions can have *default argument(s)*:

```
double distance( const double x, const double y=0. ) {  
    return sqrt( (x-y)*(x-y) );  
}
```

Any default argument(s) should come last in the parameter list.