Functions I

DM2111
C++ Programming

Introduction

Introduction	Break
Problem solving	Array and Strings
Basic elements of C++	Array and Strings
Basic elements of C++	Pointers
Statements	Pointers
Repetition	I/O operations
Functions	Structs
Functions	Others

Agenda

- What Are Functions
- Why Functions
- Predefined Functions
- Syntax
- Prototype
- Parameter Types

What Are Functions

- a block of code with a name
- takes 0 or more arguments
- can return a value
- can be overloaded

What Are Functions

- A function helps break the statements into logical and manageable parts
- Every sub-problem can be a function
- Every C++ program is made up of functions

```
#include <iostream>

int main (void)
{
    return 0;
}
```

Why Functions

- Not practical to put everything into main
- Each function should be written to tackle one task
- A function is written once and can be used many times
- A function can be re-written without major program changes
- Different people can work on different functions simultaneously
- Enhances readability
- Users may use function without understanding the implementation

Why Functions

TL;DR

Always use functions to organise your code

Examples?

Predefined Functions

- Predefined functions are functions that are written by someone else that we can simply use
- Examples
  - abs (x)
  - pow (x, y)
  - printf (…)
  - scanf (…)

Syntax

```
type name (parameter1, parameter2, ...)
{
   statement
}
```

type - data type of the value returned by the function

name - name of the function

parameter - parameters consists of a type and an identifier

statement - statements that make up the function body

Return types can be void

```
void makeBeatBoxSounds()
{
    // code for producing beatbox sounds
    // wipe saliva
}
```

This function do not need to return a value, however, we still need to specify a return type.

```
int rollDice()
{
    return 4;
}
```

This function returns a number, of type `int`.

You can have many parameters or none at all

```
void goHome(void)
{
    // code for looking up directions to
    // go home
}
```

This function do not need to take in any parameters.
You can explicitly put a void, or put nothing at all.

```
void stareAtWall(int wallID)
{
    // locate wall and stare at it
}
```

This function requires a variable of type int, named wallID.

You can have many parameters or none at all

```
double pow(double base, double exponent)
{
    // do some mathematical magic
    return result;
}
```

This function takes in 2 parameters.

Each parameter is in the form of a variable declaration

type and variable name

```
double x
```

Separated with a comma

Getting out of a function

A function returns when

- It reaches the end
- The return statement is encountered

```
void printStars (int num)
{
    if (num <= 0)
    {
        // how to print negative or no stars?
        return;
    }
    // print stars!
    for (int i = 0; i < num; i++)
    {
        cout << "*";
    }
}
```

You promised to return!

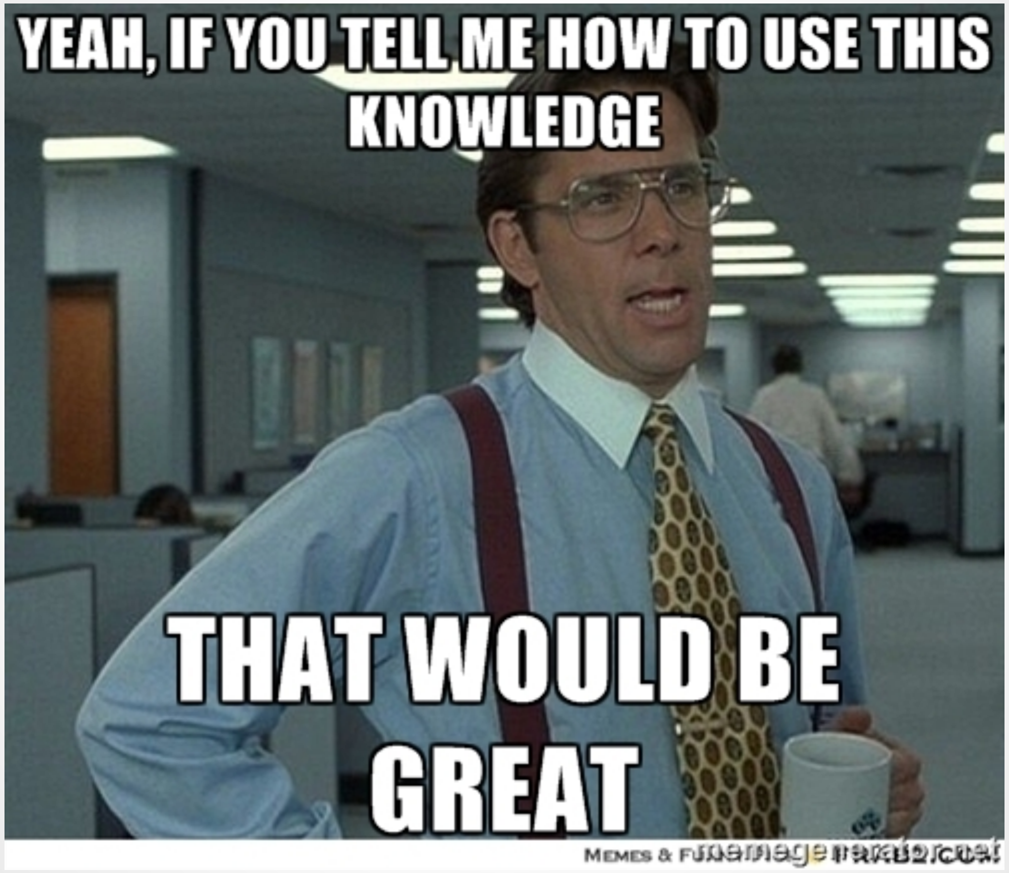
A function that is expected to return a value MUST return a value

```
int triple (int num)
{
    if (num == 0)
    {
        return;
    }
    else
    {
        return 3 * num;
    }
}
```

✗

```
int triple (int num)
{
    if (num == 0)
    {
        return 0;
    }
    else
    {
        return 3 * num;
    }
}
```

✓


YEAH, IF YOU TELL ME HOW TO USE THIS KNOWLEDGE
THAT WOULD BE GREAT
memegenerator.net

Calling calling...

```
int triple (int num);
```

```
triple(4);
triple(0);
triple();
triple(4, 5);
triple(9.4);
triple('1');

9 = triple(3);

int num = 6;
int result = triple(num);

float kill = 4.5f;
double nooo = 6.8;
kill = triple(triple(kill));
nooo == triple(kill);
```

“A wizard will never provide more parameters, nor will he provide fewer parameters, he gives precisely the number of parameters he needs to.”

Well… not really true

Flow of a program

```
void printStar (void)
{
    cout << "*";
}

void big (int a, int b)
{
    return a > b? a : b;
}

float getPi (void)
{
    return 3.14159;
}

float area (float rad)
{
    return getPi() * rad * rad;
}
```

```
void main (void)
{
    prtStar ();

    int n;
    n = big (6, 7);

    float cArea;
    cArea = area (5);

    cout << area(6);
}
```

Function Prototype

- Compilation works top down
- When the compiler comes across a function call, it must know about the function
- Hence, a function has to be declared before being used
- Prototype - function heading without the body
- Prototype identifies the function name, return type and parameter types.

Declare the function before you use!

```
void main (void)
{
    int val;

    val = triple (4);
    cout << val << endl;
}

int triple (int num)
{
    return 3 * num;
}
```

what the function?

✗

```
int triple (int num);

void main (void)
{
    int val;

    val = triple (4);
    cout << val << endl;
}

int triple (int num)
{
    return 3 * num;
}
```

I know you!

✓

You can omit variable names in function prototype

```
int triple (int num);
int triple (int);
```

Since prototype needs only the parameter types, the variable names can be omitted

Function names follow same rules as identifier names

Function names ARE identifiers

```
int triple (int);
int triple;
```

✗

You can’t have a variable with the same name as a function.

Argument Passing

- A variable can be passed to a function by:
  - Value
  - Reference

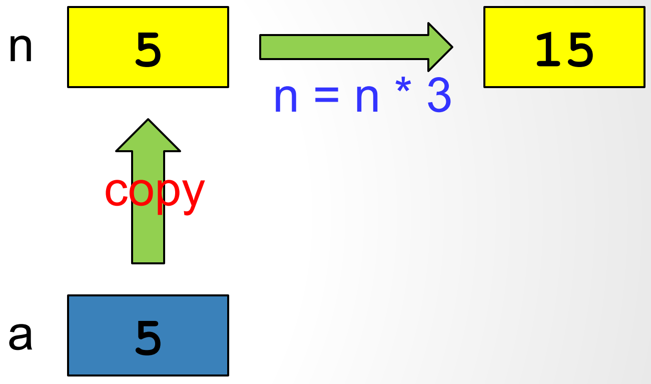
Pass by value receives a copy

The function receives a *copy* of the parameter

```
void triple (int n)
{
    n = n * 3;
    cout << n << endl; // 
   15
}

void main (void)
{
    int a = 5;

    triple (a);
    cout << a << endl; // 5
}
```



```
15
5
```

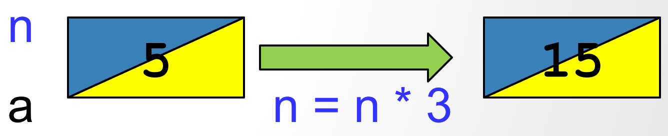
Reference parameters refer directly to the objects

The function refers to the *actual* object

```
void triple (int & n)
{
    n = n * 3;
    cout << n << endl; // 15
}

void main (void)
{
    int a = 5;

    triple (a);
    cout << a << endl; // 15
}
```



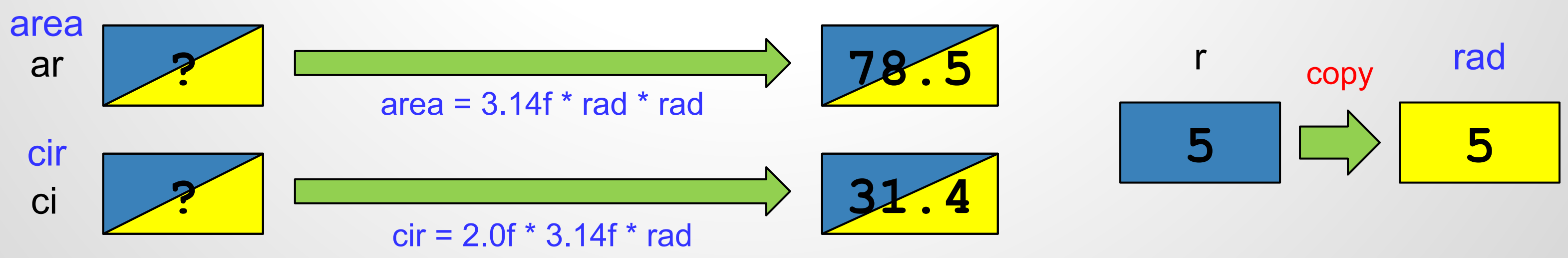
```
15
15
```

Pass by reference to return additional information

Function needs to return more than 1 variable

```
void calc (int rad, float& area, float& cir)
{
    area = 3.14f * rad * rad;
    cir = 2.0f * 3.14f * rad;
}

void main (void)
{
    float ar, ci, r = 5;
    calc (r, ar, ci);
}
```



Use reference as much as possible

Passing by reference do not copy values

- Saves memory and time
- Some objects cannot be copied

You need to modify the actual parameter

You need to return additional information

- At times when 1 return variable is not enough