Procedures, Parameters, Values and Variables

Steven R. Bagley

Compiling C

- Write our program (the *source code*) in a text file using a text editor
- Then need to compile it
  `gcc source.c`
- Output called `a.out`
- Use the `–o` flag to specify output name

Compiling C

- Write our program (the *source code*) in a text file using a text editor
- Then need to compile it
  ```
  gcc -o myprog source.c
  ```
- Output called `myprog`
- Use the `-o` flag to specify output name

Start me up…

- Computer needs to know where to start
- C defines that it starts with a procedure called `main()`
- Every C program will have one of these (somewhere)

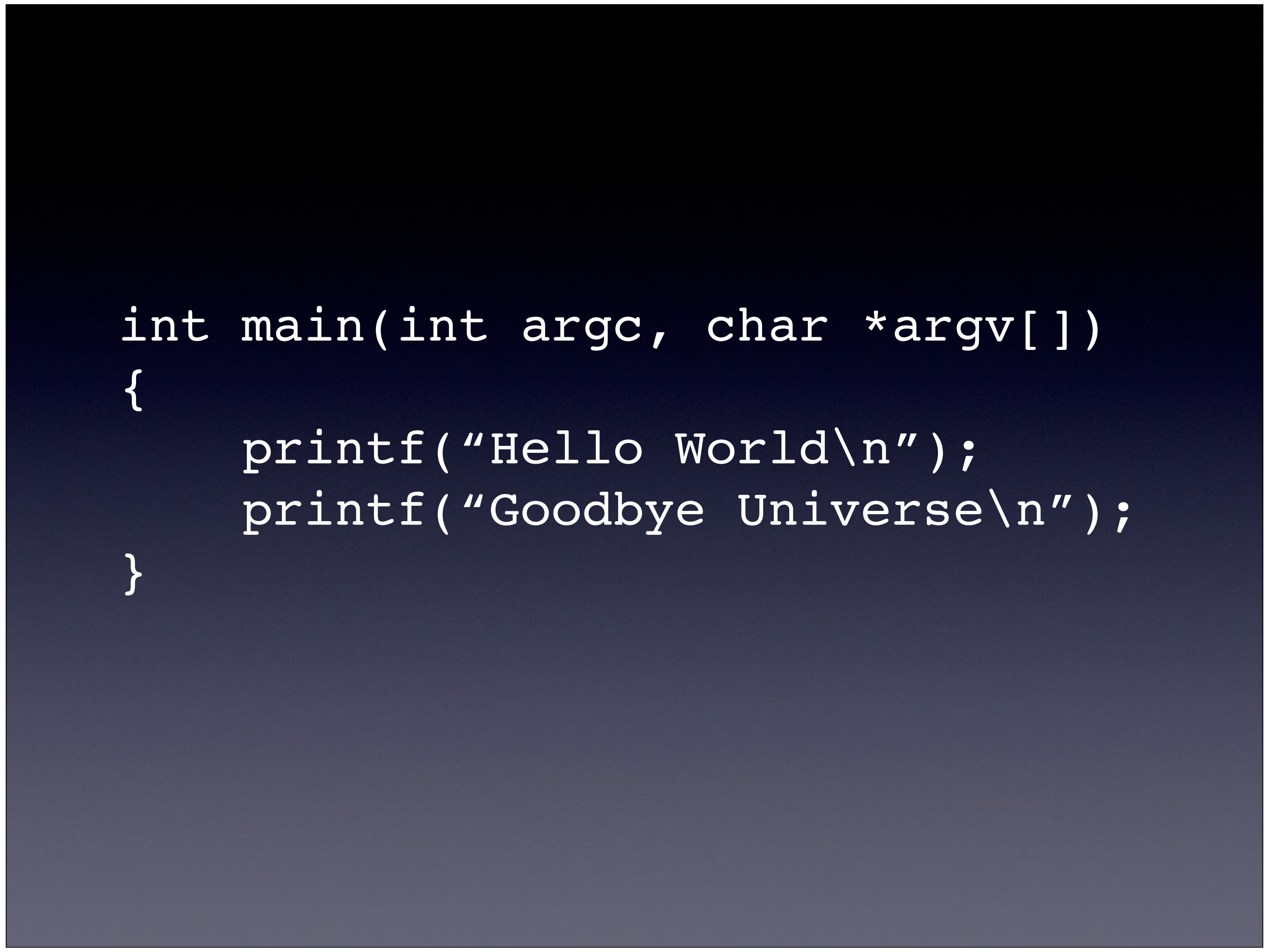
We'll look at what
these mean later

```
int main(int argc, char *argv[])
{
  /* Program goes here */
}
```

Lets have a go…

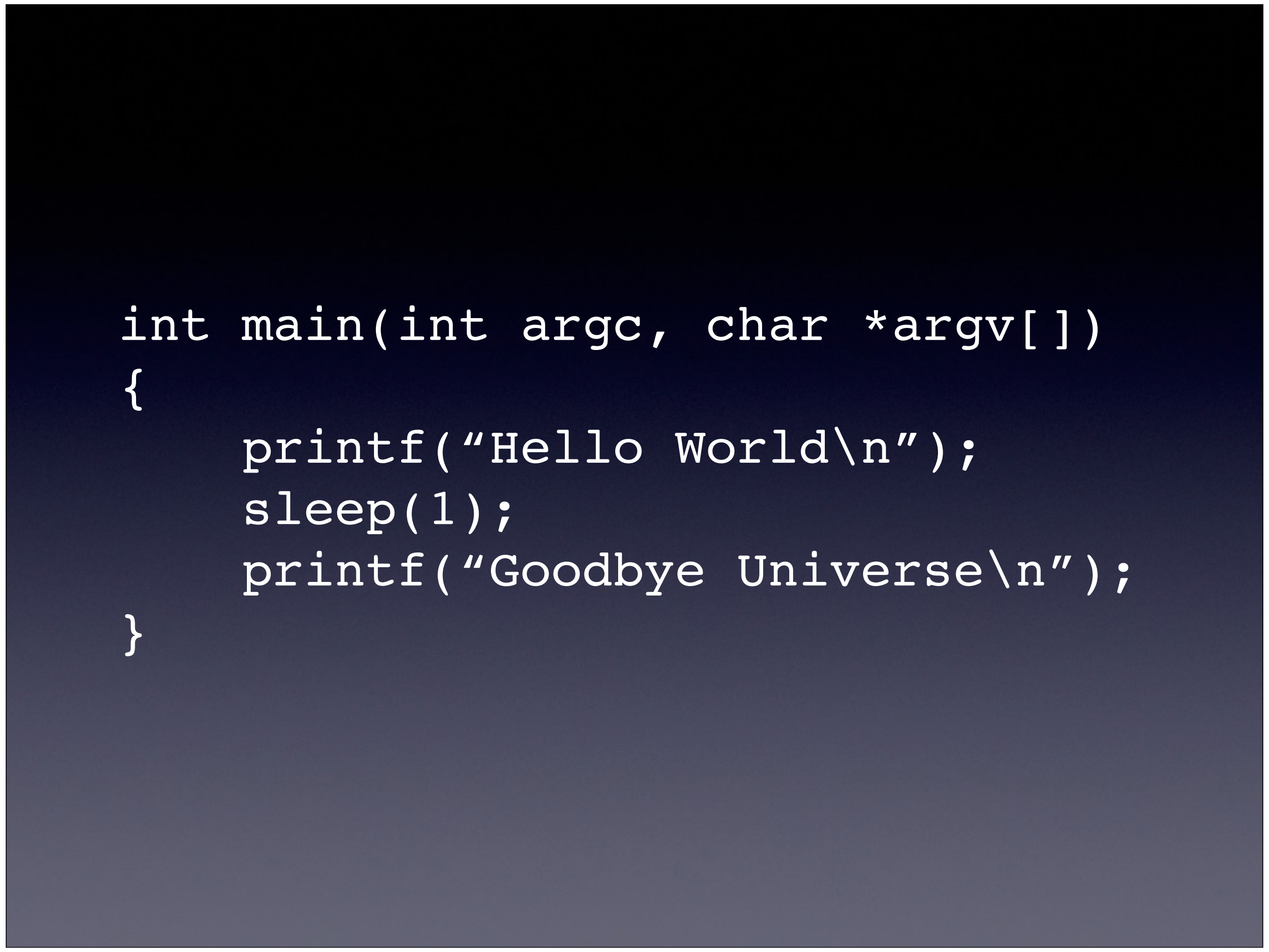
Recap

- A Program is a sequence of statements (instructions)
- Statements executed one-by-one in order
- Program starts at `main()`
- But what is `main()`?



```
int main(int argc, char *argv[])
{
    printf("Hello World\n");
    printf("Goodbye Universe\n");
}
```

Go through this step by step -- add a sleep command to show that its a sequence

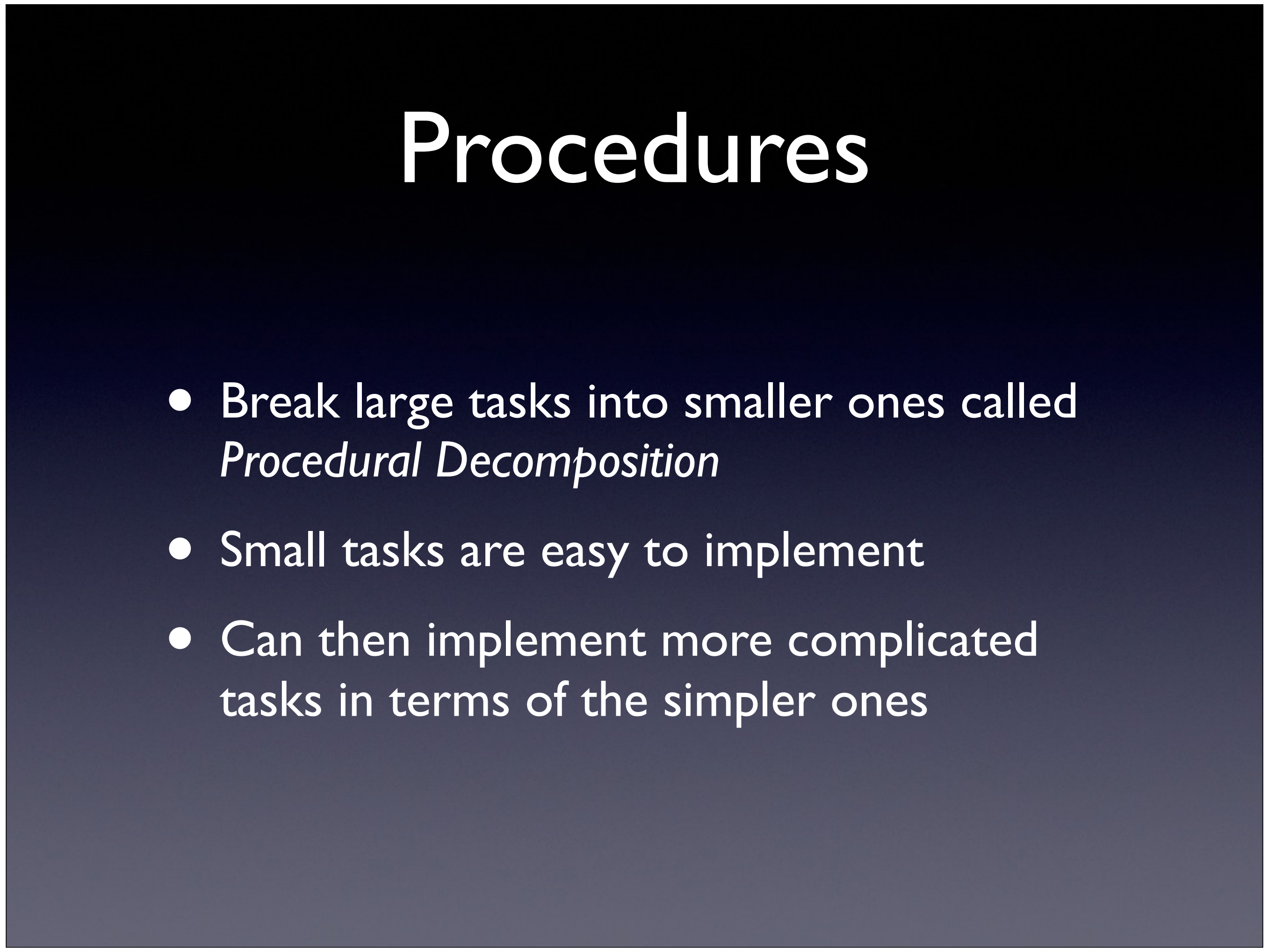


```
int main(int argc, char *argv[])
{
    printf(“Hello World\n”);
    sleep(1);
    printf(“Goodbye Universe\n”);
}
```

Go through this step by step -- add a sleep command to show that its a sequence

Procedures

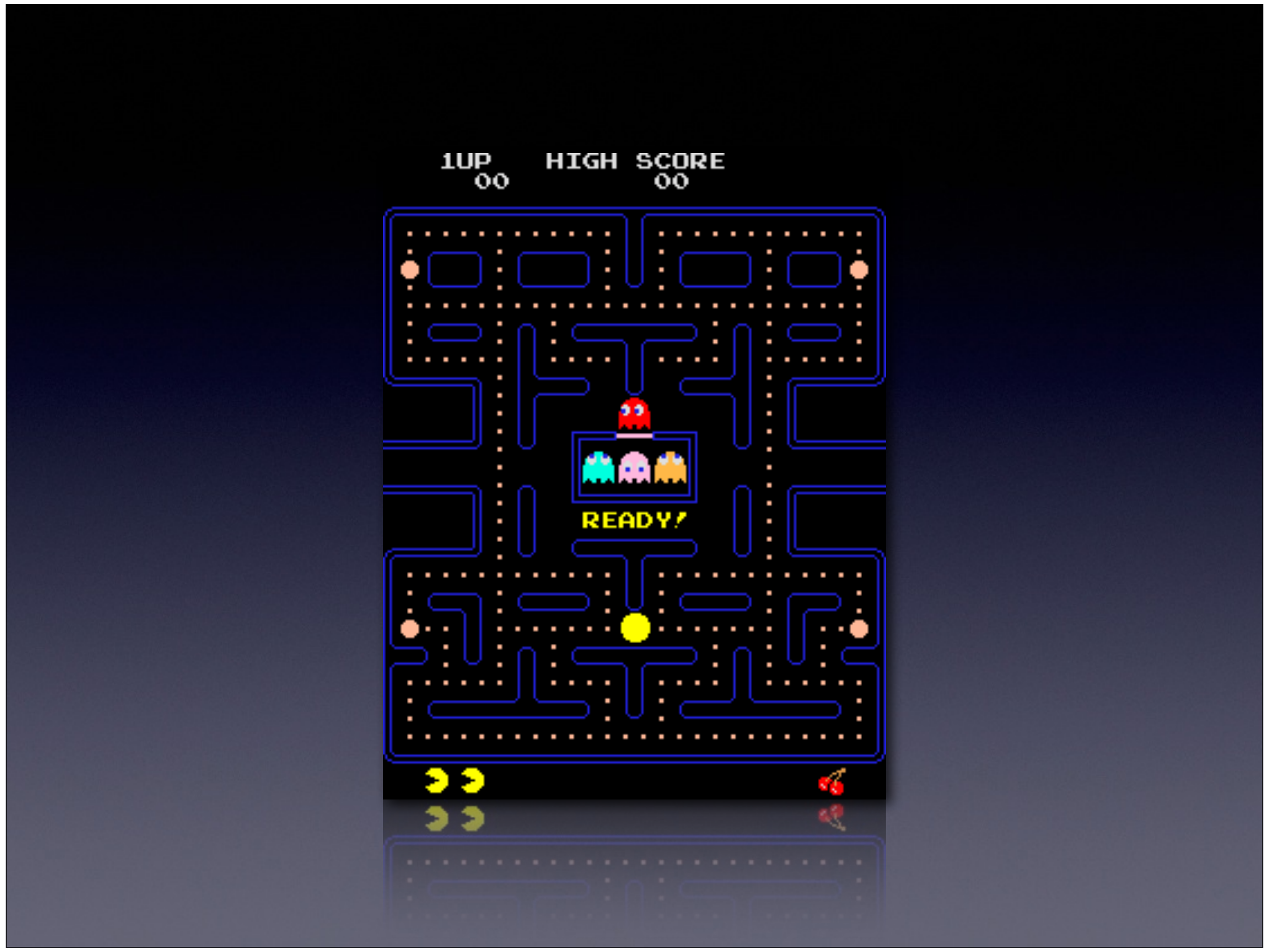
- C is a procedural language
- All instructions are inside a procedure
- Procedures are a block of instructions that are given a name
- Invoke these instructions by name



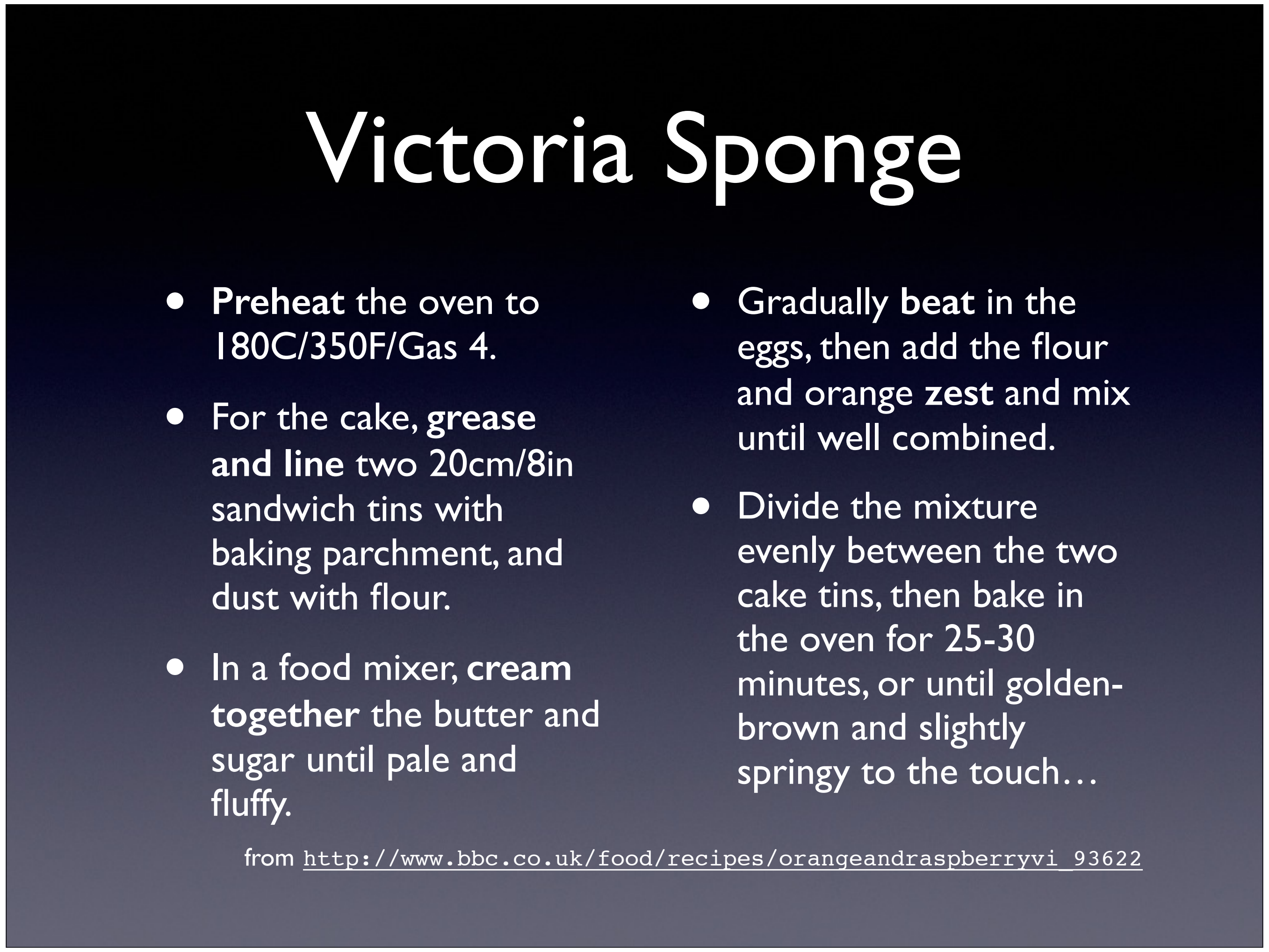
Procedures

- Break large tasks into smaller ones called *Procedural Decomposition*
- Small tasks are easy to implement
- Can then implement more complicated tasks in terms of the simpler ones

Function to draw lines, can then define a function to draw square in terms of four four lines



Talk about the different procedures that you might need for a pacman game

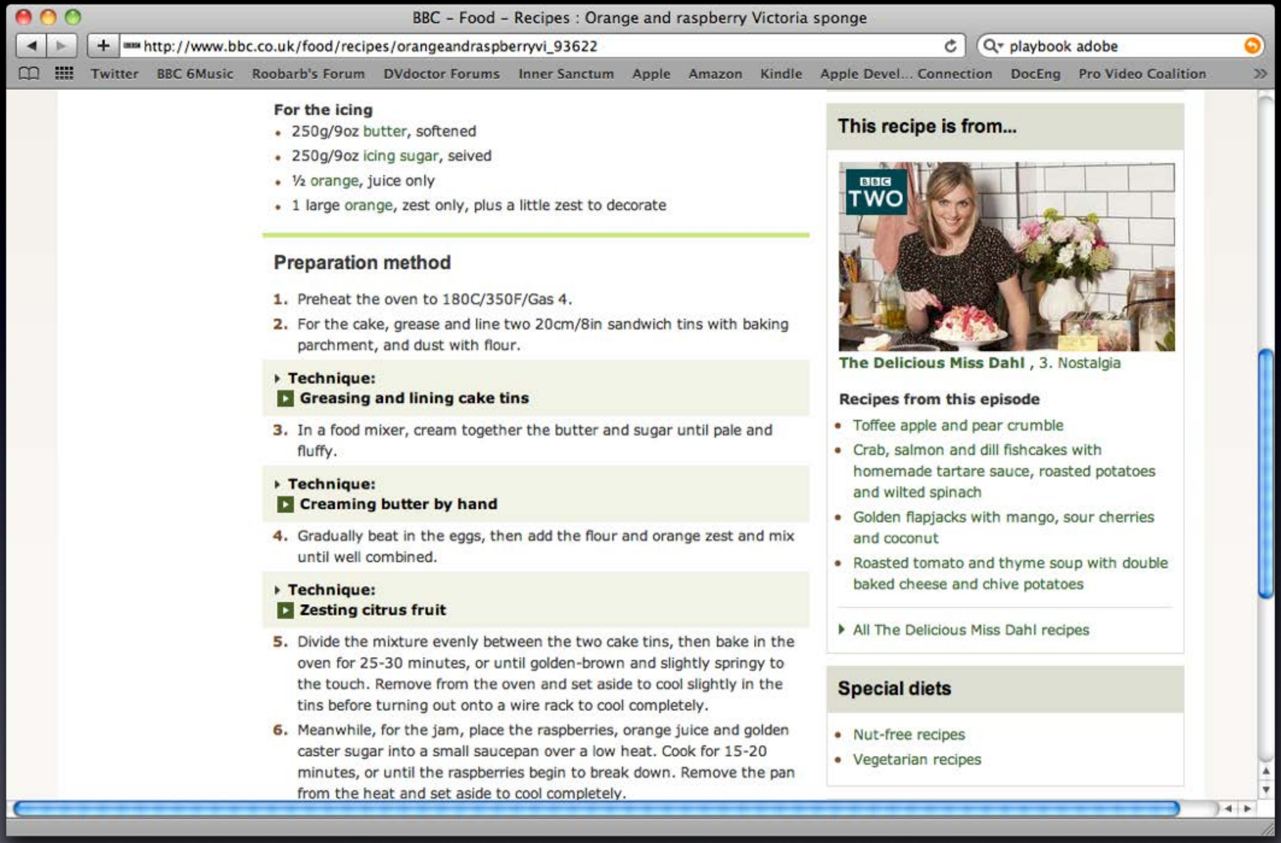


Victoria Sponge

- **Preheat** the oven to 180C/350F/Gas 4.
- For the cake, **grease and line** two 20cm/8in sandwich tins with baking parchment, and dust with flour.
- In a food mixer, **cream together** the butter and sugar until pale and fluffy.
- Gradually **beat** in the eggs, then add the flour and orange **zest** and mix until well combined.
- Divide the mixture evenly between the two cake tins, then bake in the oven for 25-30 minutes, or until golden-brown and slightly springy to the touch…

from `http://www.bbc.co.uk/food/recipes/orangeandraspberryvi_93622`

We see this all the time in cooking — the recipe uses terms like ‘cream’ or ‘fold’ that tell you to do a specific set


BBC – Food – Recipes : Orange and raspberry Victoria sponge
http://www.bbc.co.uk/food/recipes/orangeandraspberryvi_93622
playbook adobe
Twitter
BBC 6Music
Roobarb's Forum
DVdoctor Forums
Inner Sanctum
Apple
Amazon
Kindle
Apple Devel... Connection
DocEng
Pro Video Coalition

For the icing

- 250g/9oz butter, softened
- 250g/9oz icing sugar, seived
- ½ orange, juice only
- 1 large orange, zest only, plus a little zest to decorate

Preparation method

1. Preheat the oven to 180C/350F/Gas 4.
2. For the cake, grease and line two 20cm/8in sandwich tins with baking parchment, and dust with flour.

Technique:
Greasing and lining cake tins

3. In a food mixer, cream together the butter and sugar until pale and fluffy.

Technique:
Creaming butter by hand

4. Gradually beat in the eggs, then add the flour and orange zest and mix until well combined.

Technique:
Zesting citrus fruit

5. Divide the mixture evenly between the two cake tins, then bake in the oven for 25-30 minutes, or until golden-brown and slightly springy to the touch. Remove from the oven and set aside to cool slightly in the tins before turning out onto a wire rack to cool completely.
6. Meanwhile, for the jam, place the raspberries, orange juice and golden caster sugar into a small saucepan over a low heat. Cook for 15-20 minutes, or until the raspberries begin to break down. Remove the pan from the heat and set aside to cool completely.

This recipe is from...

The Delicious Miss Dahl, 3. Nostalgia

Recipes from this episode

- Toffee apple and pear crumble
- Crab, salmon and dill fishcakes with homemade tartare sauce, roasted potatoes and wilted spinach
- Golden flapjacks with mango, sour cherries and coconut
- Roasted tomato and thyme soup with double baked cheese and chive potatoes

All The Delicious Miss Dahl recipes

Special diets

- Nut-free recipes
- Vegetarian recipes

A Recipe’s Procedure

- Recipe is a procedure for making a cake
- Defined in terms of other *‘procedures’* for doing specific smaller tasks
- Can see how to make the cake clearly
- Smaller tasks are common across several recipes
- *Procedures* are reused across several recipes

Program’s Procedures

- Programs are defined in terms of procedures that do specific tasks
- Some tasks are common across many programs
  e.g. printing out data
- These procedures can be reused
- Others specific to a particular program

Functions and Procedures

- In C, procedures tend be to called *functions*
- As they can return a value
- Called by giving their name followed by brackets
  e.g. `printf()`
- C programs start with the `main()` function

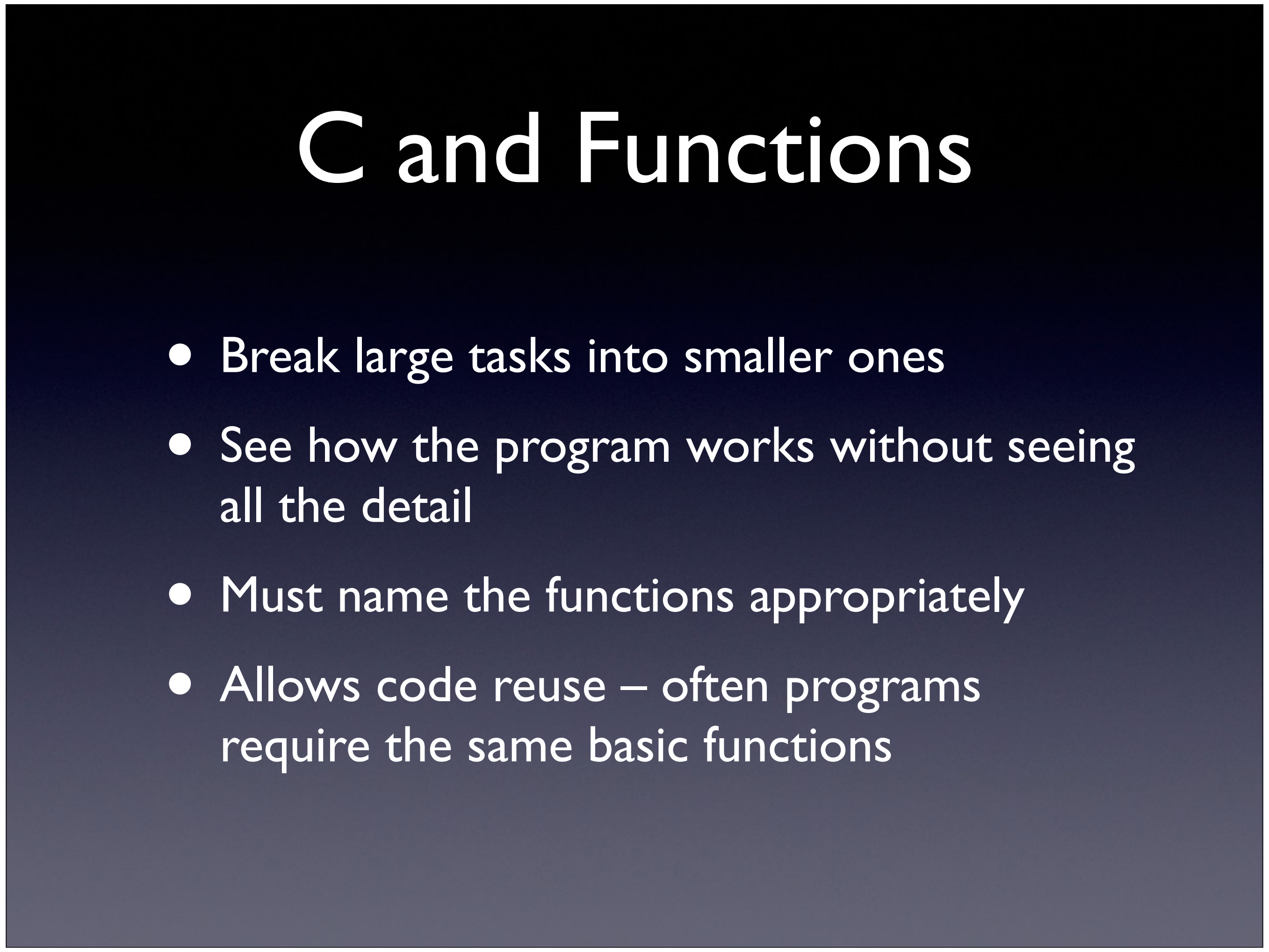
C Language

- C provides a minimal set of operations
- Everything else is handled by functions
- Including printing things out — `printf()` is a function written in C
- Library of functions to do common tasks
- Create new ones to do specific tasks

C Statements

```
break      case     continue  default
 do        else       for       goto
 if       return     sizeof    switch
while
```

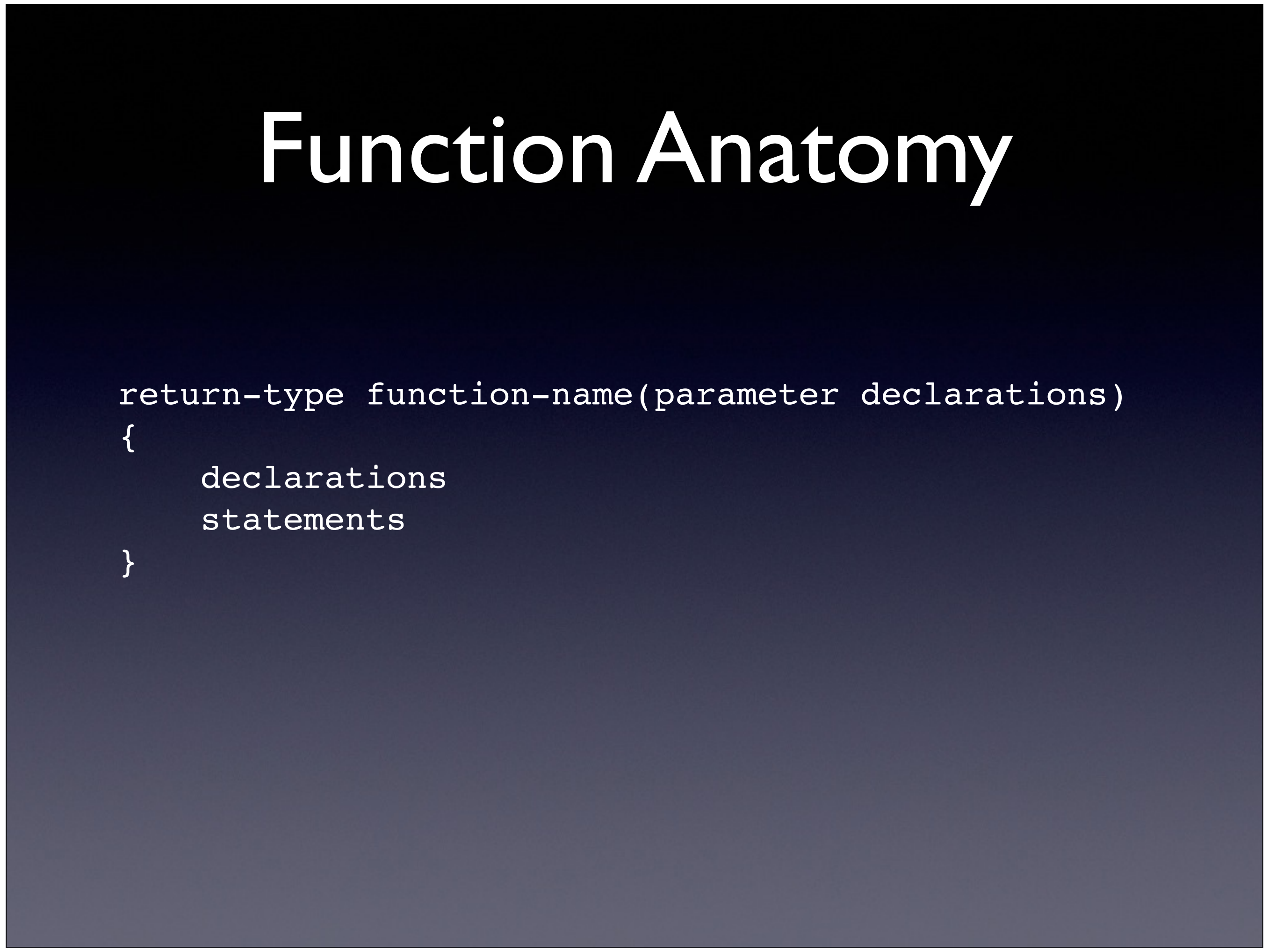
- Mathematical operations
- Variable and memory manipulation
- And, of course, defining new functions



C and Functions

- Break large tasks into smaller ones
- See how the program works without seeing all the detail
- Must name the functions appropriately
- Allows code reuse – often programs require the same basic functions

Generic sprite drawing procedure instead of a draw pacman or ghost procedure



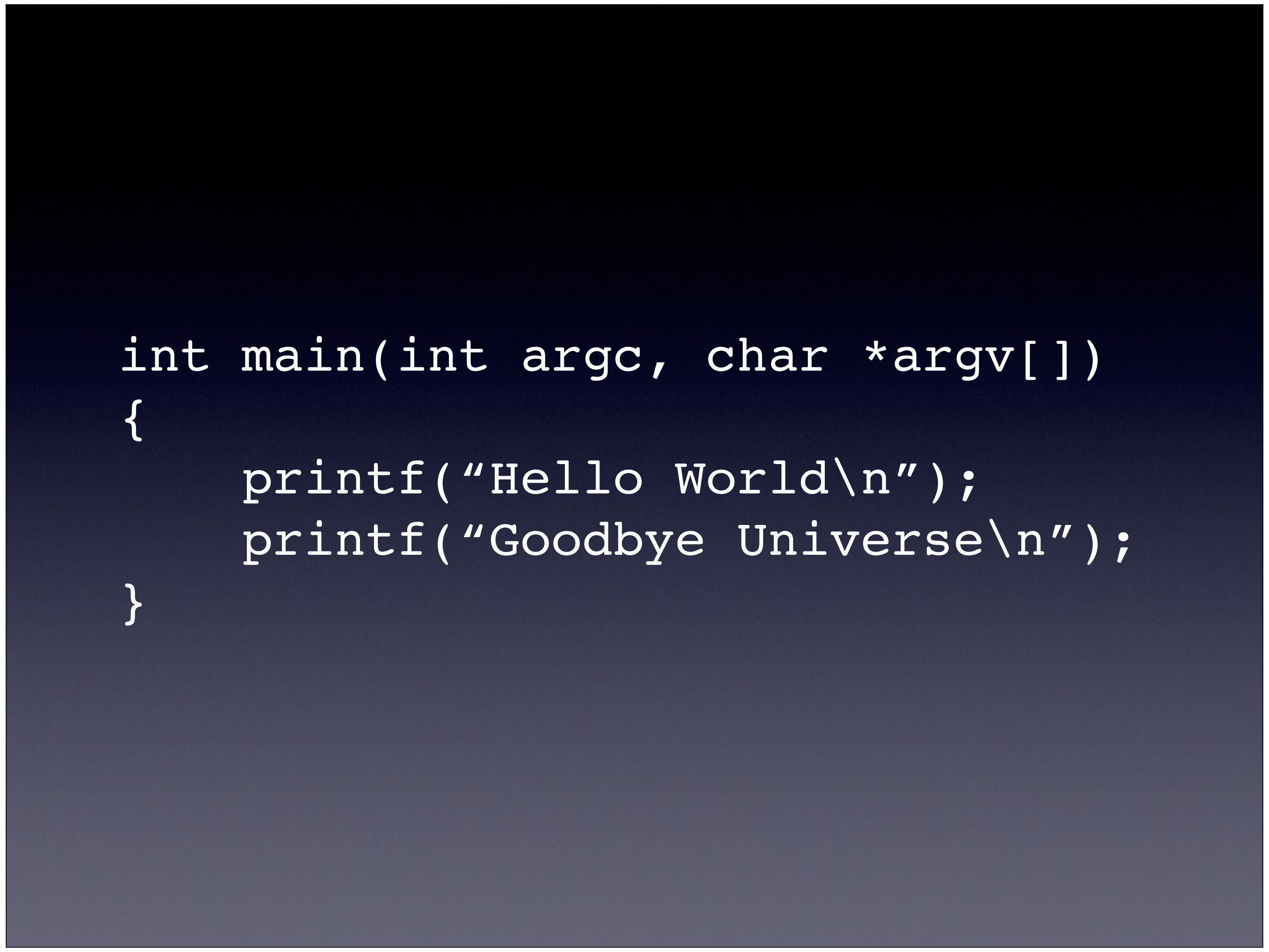
Function Anatomy

```
return-type function-name(parameter declarations)
{
    declarations
    statements
}
```

All statements end with a semicolon in C -- forget them and you’ll get compile errors (and not sensible ones)

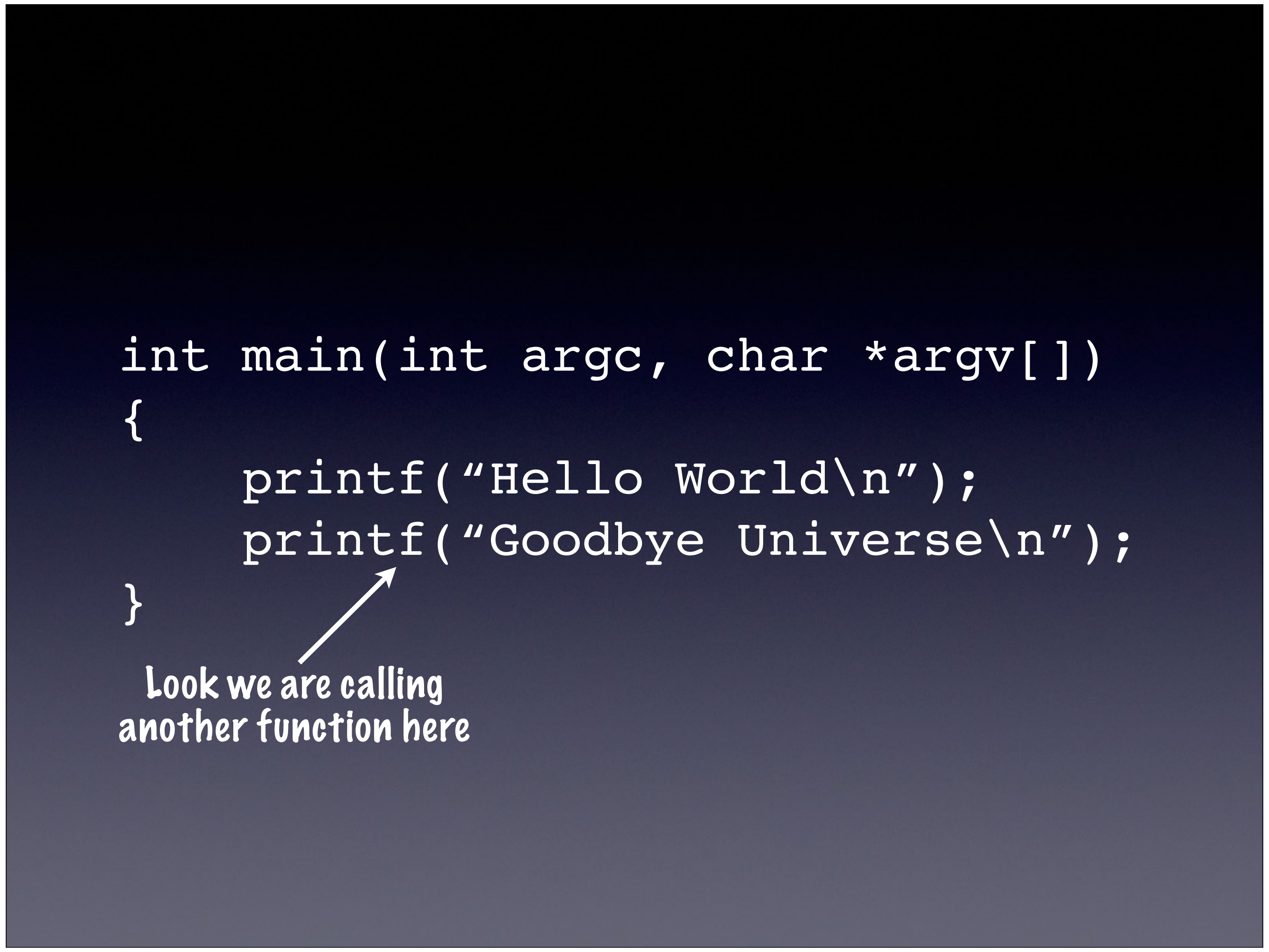
Function Anatomy

- *Return type* — states the type of the value returned from the function (if any)
- *Function name* — `main` in this case
- *Parameter declarations* — that the function uses to perform its task (if any), also have type
- { … } — groups statements together
- *Declarations* — declare any variables used in the function
- *Statements* — what the program does (almost always ended by a semicolon)
- Return to the issue of type later



```
int main(int argc, char *argv[])
{
    printf(“Hello World\n”);
    printf(“Goodbye Universe\n”);
}
```

Explain this as a function, identifying all the various different bits



```
int main(int argc, char *argv[])
{
    printf("Hello World\n");
    printf("Goodbye Universe\n");
}
```

Look we are calling
another function here

Explain this as a function, identifying all the various different bits

Calling a Function

- Already seen this with `printf`
- Give the function name followed by an open bracket
- Then the value for each parameter (separated by commas)
- Followed by a closing bracket
- Look at return value later…

Defining a Function

- Again, seen this with `main()`
- But where do we put it in our source file?
- Easy for the first function…
- Generally wants to be outside any other function
- But…

Declaring Functions

- Compiler needs to know the type signature of the function when its called
- If its not specified, it will use the default (which is almost always wrong)
- Generates a compiler error
- Must declare the function before we call it

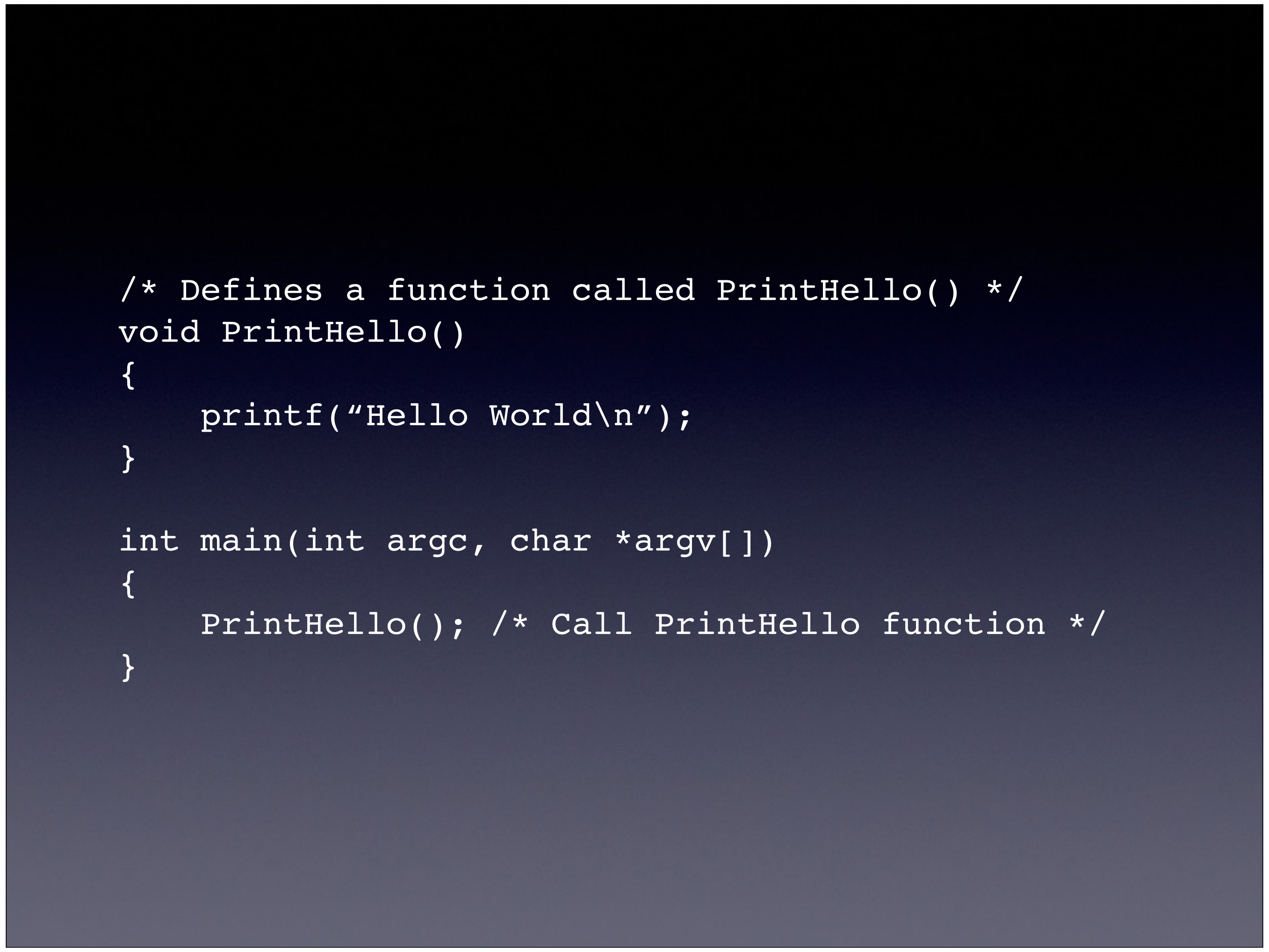
Declaring Functions

- Either, define the function in the file before it is first called
  - Not always possible…
  - Also leads to programs that have the start at the end of the source file
- Or put a function declaration in place

Think Function A calls Function B which calls Function A...

Function Declarations

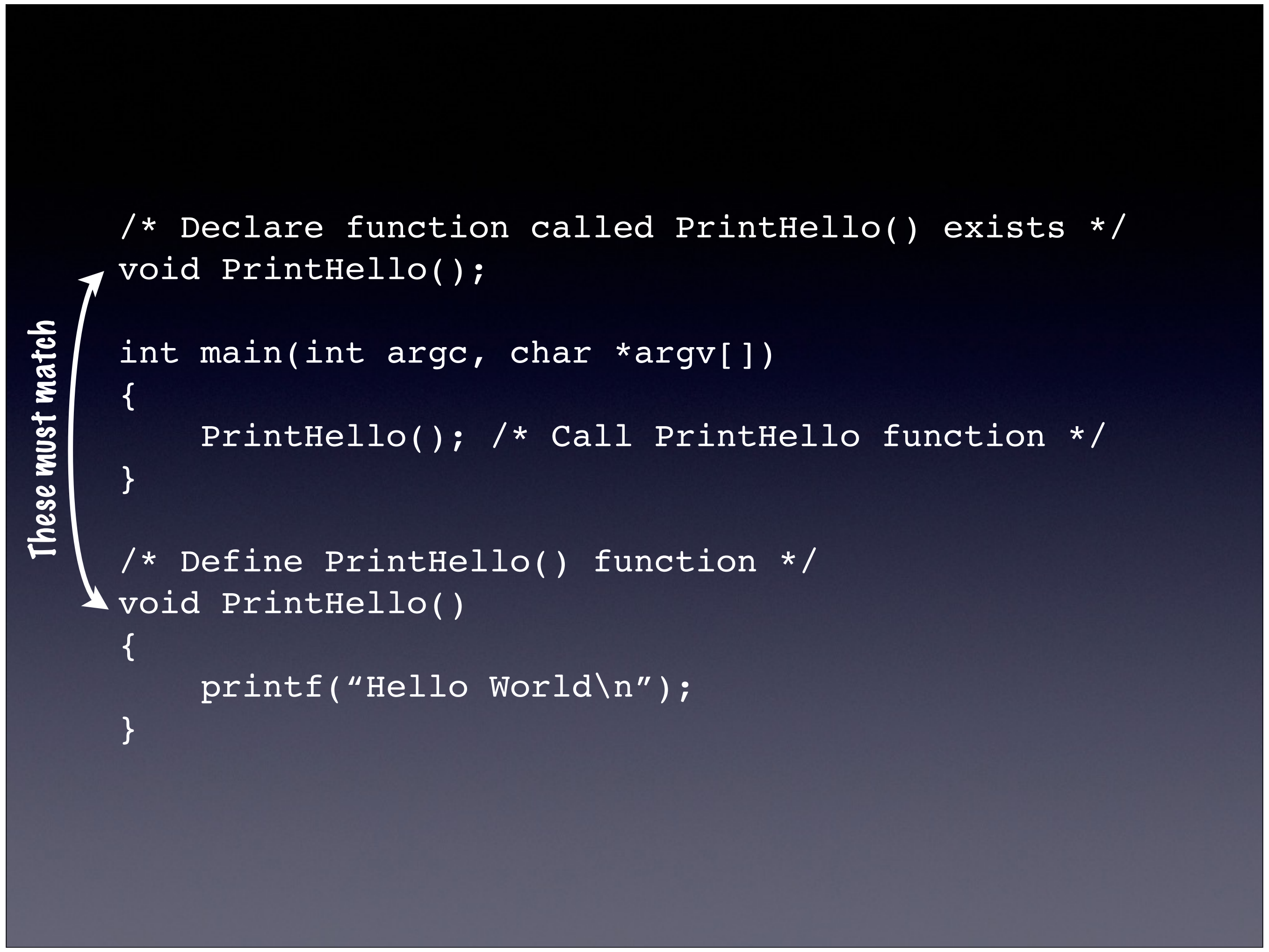
- Tell the compiler there will be a function with this type signature used
- But don't define its implementation
- That's provided later in the file…
- Declaration is identical to the first line of the function followed by a semicolon



```
/* Defines a function called PrintHello() */
void PrintHello()
{
    printf(“Hello World\n”);
}

int main(int argc, char *argv[])
{
    PrintHello(); /* Call PrintHello function */
}
```

Note the semicolons -- they are significant
void means doesn’t return anything


These must match

```
/* Declare function called PrintHello() exists */
void PrintHello();

int main(int argc, char *argv[])
{
    PrintHello(); /* Call PrintHello function */
}

/* Define PrintHello() function */
void PrintHello()
{
    printf(“Hello World\n”);
}
```

Note the semicolons -- they are significant
void means doesn’t return anything
IF they don’t match, you’ll get compile errors