Why C?

C was designed in the early 70s as a language specifically for implementing an operating system (Unix)

Why C?

C was designed in the early 70s as a language specifically for implementing an operating system (Unix)

Thus its expertise in a range of low-level, close-to-the-machine operations

Why C?

C was designed in the early 70s as a language specifically for implementing an operating system (Unix)

Thus its expertise in a range of low-level, close-to-the-machine operations

It was subsequently found to be useful in a wide range of other areas

Why C?

C was designed in the early 70s as a language specifically for implementing an operating system (Unix)

Thus its expertise in a range of low-level, close-to-the-machine operations

It was subsequently found to be useful in a wide range of other areas

The original C, designed by Brian Kernighan and Dennis Ritchie (K&R C) was later modified and updated by the standards organisation ANSI

Why C?

C was designed in the early 70s as a language specifically for implementing an operating system (Unix)

Thus its expertise in a range of low-level, close-to-the-machine operations

It was subsequently found to be useful in a wide range of other areas

The original C, designed by Brian Kernighan and Dennis Ritchie (K&R C) was later modified and updated by the standards organisation ANSI

The current version of the standard is ANSI C11 (actually an ISO standard adopted by ANSI), was approved in December 2011

Why C?

Despite its age and origins C is used massively out there in the real world

Why C?

Despite its age and origins C is used massively out there in the real world

In terms of programming jobs, it has been in the top 5 required languages for decades; often in the top 2

Why C?

Despite its age and origins C is used massively out there in the real world

In terms of programming jobs, it has been in the top 5 required languages for decades; often in the top 2

Pretty much anything that is “close to the hardware” is programmed in C

Why C?

Despite its age and origins C is used massively out there in the real world

In terms of programming jobs, it has been in the top 5 required languages for decades; often in the top 2

Pretty much anything that is “close to the hardware” is programmed in C

It is *very* good when you need low-level, fast programs

Why C?

Despite its age and origins C is used massively out there in the real world

In terms of programming jobs, it has been in the top 5 required languages for decades; often in the top 2

Pretty much anything that is “close to the hardware” is programmed in C

It is *very* good when you need low-level, fast programs

For example, real-time controllers of cars, phones, graphics

Why C?

Despite its age and origins C is used massively out there in the real world

In terms of programming jobs, it has been in the top 5 required languages for decades; often in the top 2

Pretty much anything that is “close to the hardware” is programmed in C

It is *very* good when you need low-level, fast programs

For example, real-time controllers of cars, phones, graphics

But also in many other situations: it is very flexible and lends itself to many kinds of problem

Why C?

There is a price to pay for this, though

Why C?

There is a price to pay for this, though

C will allow you to do all kinds of stupid things

Why C?

There is a price to pay for this, though

C will allow you to do all kinds of stupid things

Unlike Java, it does not try to protect the programmer from themself

Why C?

There is a price to pay for this, though

C will allow you to do all kinds of stupid things

Unlike Java, it does not try to protect the programmer from themself

If you want to refer to memory that has not been allocated by the system to your program, you can do it

Why C?

There is a price to pay for this, though

C will allow you to do all kinds of stupid things

Unlike Java, it does not try to protect the programmer from themself

If you want to refer to memory that has not been allocated by the system to your program, you can do it (e.g., generating random numbers)

Why C?

In fact, poorly programmed C is one of the major causes of cracks (hacks) of programs and operating systems: many security updates to (say) Windows are to fix just this kind of error

Why C?

In fact, poorly programmed C is one of the major causes of cracks (hacks) of programs and operating systems: many security updates to (say) Windows are to fix just this kind of error

C does not protect the programmer

Why C?

In fact, poorly programmed C is one of the major causes of cracks (hacks) of programs and operating systems: many security updates to (say) Windows are to fix just this kind of error

C does not protect the programmer

This is an explicit trade-off of speed of C programs against the (relative) safety of managed languages (like Java)

C

Note: if you know any Java, forget it

C

Note: if you know any Java, forget it

C and Java look similar on the page, but are very different languages

Note: if you know any Java, forget it

C and Java look similar on the page, but are very different languages

C and Java are very different

Note: if you know any Java, forget it

C and Java look similar on the page, but are very different languages

C and Java are very different

A lot of what you learned in Java simply does not apply to C

Note: if you know any Java, forget it

C and Java look similar on the page, but are very different languages

C and Java are very different

A lot of what you learned in Java simply does not apply to C

If you don't know Java, you are so much better off (in many ways)

C

C puts very little between the programmer and the machine

C

C puts very little between the programmer and the machine

Unlike Java, which tries to hide the hardware from the programmer

C

C puts very little between the programmer and the machine

Unlike Java, which tries to hide the hardware from the programmer

Note: this is not a judgement of which language is “better”

C

The only question is “which language is better suited **to the problem in hand**”

C

The only question is “which language is better suited **to the problem in hand**”

Exercise. Which is the better tool: a screwdriver or a hammer?

C

There is only one way to learn C (or any language)

C

There is only one way to learn C (or any language)

Write C programs

C

There is only one way to learn C (or any language)

Write C programs

Lots of C programs

C

There is only one way to learn C (or any language)

Write C programs

Lots of C programs

If you invest time *now* in writing lots of little example programs, you will reap *hugely* later

C

There is only one way to learn C (or any language)

Write C programs

Lots of C programs

If you invest time *now* in writing lots of little example programs, you will reap *hugely* later

If you don't bother, it will cost you dear later

C

There is only one way to learn C (or any language)

Write C programs

Lots of C programs

If you invest time *now* in writing lots of little example programs, you will reap *hugely* later

If you don't bother, it will cost you dear later

I've lost count of the number of students who have handed in pitiful coursework while saying "I wish I'd had more practice before starting this"

Practice!

C

We start with a couple of example C programs

```
main(t,_,a )
char
*
a;
{
return!

0<t?
t<3?

main(-79,-13,a+
main(-87,1-_,
main(-86, 0, a+1 )


+a)):

1,
t<_?
main( t+1, _, a )
:3,

main ( -94, -27+t, a )
&&t == 2 ?_
<13 ?

main ( 2, _+1, "%s %d %d\n" )
```

```
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#;#q#
:'d*'3,}{w+K w'K:'+}e#';dq#'l
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw'
i;# ){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw'
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n'
')# }'+}##(!!/")
:
t<-50?
_==*a ?
putchar(31[a]):

main(-65,_,a+1)
:
main((*a == '/') + t, _, a + 1 )
:

0<t?

main ( 2, 2 , "%s")
:*a=='/'||

main(0,

main(-61,*a, "!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m
.vpbks,fxntdCeghiry")
```

C

Written by Ian Phillipps

C

Written by Ian Phillipps

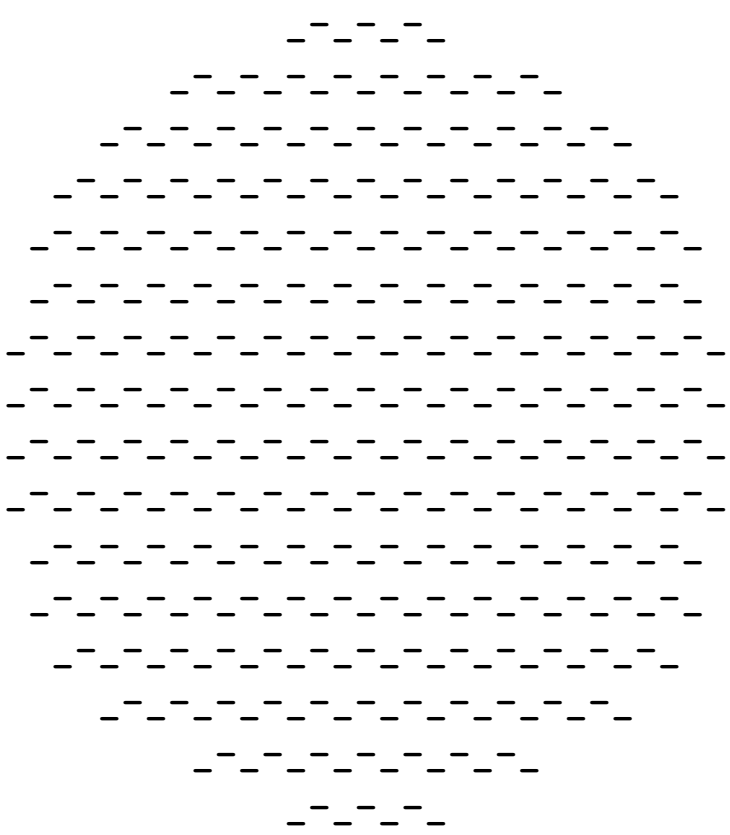
When run this produces...

C

```
On the first day of Christmas my true love gave to me
a partridge in a pear tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a pear tree.

On the third day of Christmas my true love gave to me
three french hens, two turtle doves
and a partridge in a pear tree.

On the fourth day of Christmas my true love gave to me
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the fifth day of Christmas my true love gave to me
five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the sixth day of Christmas my true love gave to me
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
...
```

C

```
#define _ F-->00||-F-OO--;
int F=00,OO=00;main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
```



```
}
```

C

Written by Brian Westley

C

Written by Brian Westley

When run this produces...

C

Written by Brian Westley

When run this produces...

```
3.141
```

C

These are taken from the Obfuscated C Competition: a competition to see how unreadable you can make a C program: `http://www.no.ioccc.org/years.html`

C

These are taken from the Obfuscated C Competition: a competition to see how unreadable you can make a C program: `http://www.no.ioccc.org/years.html`

You can write unreadable programs in all languages: C makes it particularly easy

C

These are taken from the Obfuscated C Competition: a competition to see how unreadable you can make a C program: `http://www.no.ioccc.org/years.html`

You can write unreadable programs in all languages: C makes it particularly easy

So you need to be extra-careful on layout and presentation when writing C!

C

So here is a simpler example. In file `hello.c`

```
#include <stdio.h>

/* This is a
   block comment */
int main(void)
{
  // do something interesting
  printf("hello world\n");

  return 0;
}
```

C

Points of note

C

Points of note

- General syntax — comments, curly brackets, semicolons, quotes, etc. — may be pretty familiar

C

Points of note

- General syntax — comments, curly brackets, semicolons, quotes, etc. — may be pretty familiar
- Filename can be anything that ends `.c`, no relationship connecting filenames to classes goes on: C does not have classes

C

Points of note

- General syntax — comments, curly brackets, semicolons, quotes, etc. — may be pretty familiar
- Filename can be anything that ends `.c`, no relationship connecting filenames to classes goes on: C does not have classes
- The `#include` we shall describe later: for now just think of it as something to put at the start of every C file

C

Points of note

- General syntax — comments, curly brackets, semicolons, quotes, etc. — may be pretty familiar
- Filename can be anything that ends `.c`, no relationship connecting filenames to classes goes on: C does not have classes
- The `#include` we shall describe later: for now just think of it as something to put at the start of every C file
- The function `main` is the entry point of the program. i.e., when the program is run, it starts executing from here

C

Points of note

- General syntax — comments, curly brackets, semicolons, quotes, etc. — may be pretty familiar
- Filename can be anything that ends `.c`, no relationship connecting filenames to classes goes on: C does not have classes
- The `#include` we shall describe later: for now just think of it as something to put at the start of every C file
- The function `main` is the entry point of the program. i.e., when the program is run, it starts executing from here
- It doesn't have any fancy type: it just returns an integer

C

- In this example `main` has an empty (void) argument list

C

- In this example `main` has an empty (void) argument list
- The function `printf` prints stuff, here a string with a newline at the end (the `\n`)

C

- In this example `main` has an empty (void) argument list
- The function `printf` prints stuff, here a string with a newline at the end (the `\n`)
- The program exits when you return from `main`

C

- In this example `main` has an empty (void) argument list
- The function `printf` prints stuff, here a string with a newline at the end (the `\n`)
- The program exits when you return from `main`
- `main` returns a value back to the operating system when the program finishes. The OS can use this in various ways, if it wishes. The convention is `0` means “finished successfully”, while non-zero values can signify various kinds of error

C

We can compile this file

```
% cc -Wall -o hello hello.c
```

C

- The % is a command line prompt

C

- The `%` is a command line prompt
- `-Wall` is a option to the compiler that tells it to report all warnings. A warning is something in your code that might not be technically wrong, but is sufficiently dodgy to be worth looking at. Always use this option. You should aim to write code with no warnings (and no errors!)

C

- The `%` is a command line prompt
- `-Wall` is a option to the compiler that tells it to report all warnings. A warning is something in your code that might not be technically wrong, but is sufficiently dodgy to be worth looking at. Always use this option. You should aim to write code with no warnings (and no errors!)
- `-Wextra` gives even more warnings

C

- The `%` is a command line prompt
- `-Wall` is a option to the compiler that tells it to report all warnings. A warning is something in your code that might not be technically wrong, but is sufficiently dodgy to be worth looking at. Always use this option. You should aim to write code with no warnings (and no errors!)
- `-Wextra` gives even more warnings
- `-Werror` makes warnings into errors: the compiler will refuse to produce any output until you fix the warnings

C

- The `%` is a command line prompt
- `-Wall` is a option to the compiler that tells it to report all warnings. A warning is something in your code that might not be technically wrong, but is sufficiently dodgy to be worth looking at. Always use this option. You should aim to write code with no warnings (and no errors!)
- `-Wextra` gives even more warnings
- `-Werror` makes warnings into errors: the compiler will refuse to produce any output until you fix the warnings
- `-o hello` says put the compiled program in the file named `hello`. This filename can be anything you like, not necessarily related to the source code file

Aside

Note: this is an example of compiling a C program using a command-line compiler (gcc in this case)

Aside

Note: this is an example of compiling a C program using a command-line compiler (gcc in this case)

Other compilers will likely take different arguments, e.g., another compiler might not recognise `-Wall` and could have something else equivalent (e.g., `-v`)

Aside

Note: this is an example of compiling a C program using a command-line compiler (gcc in this case)

Other compilers will likely take different arguments, e.g., another compiler might not recognise `-Wall` and could have something else equivalent (e.g., `-v`)

Many IDEs exist that are supposed to make the management and compilation of large programs easier (e.g., Eclipse, Visual Studio)

Aside

Note: this is an example of compiling a C program using a command-line compiler (gcc in this case)

Other compilers will likely take different arguments, e.g., another compiler might not recognise `-Wall` and could have something else equivalent (e.g., `-v`)

Many IDEs exist that are supposed to make the management and compilation of large programs easier (e.g., Eclipse, Visual Studio)

Exercise. Investigate these to find something that suits your personal taste

Aside

NB: there is a difference between an IDE and a compiler

Aside

NB: there is a difference between an IDE and a compiler

Compiler: converts a text program into executable code

Aside

NB: there is a difference between an IDE and a compiler

Compiler: converts a text program into executable code

IDE: a tool to help the programmer write better programs

Aside

NB: there is a difference between an IDE and a compiler

Compiler: converts a text program into executable code

IDE: a tool to help the programmer write better programs

They may come together wrapped up in a single package, but they are very different things

Aside

NB: there is a difference between an IDE and a compiler

Compiler: converts a text program into executable code

IDE: a tool to help the programmer write better programs

They may come together wrapped up in a single package, but they are very different things

Keep them separate in your mind

More Aside

Many C compilers exist, both paid-for and free. Different compilers may produce more or less efficient compiled code from the same source

More Aside

Many C compilers exist, both paid-for and free. Different compilers may produce more or less efficient compiled code from the same source

If all is well, a given standards-compliant C program should compile with a standards-compliant C compiler and should run and produce equivalent results independent of the compiler chosen

More Aside

Many C compilers exist, both paid-for and free. Different compilers may produce more or less efficient compiled code from the same source

If all is well, a given standards-compliant C program should compile with a standards-compliant C compiler and should run and produce equivalent results independent of the compiler chosen

Not all (any?) C compilers are fully standards-compliant

More Aside

Many C compilers exist, both paid-for and free. Different compilers may produce more or less efficient compiled code from the same source

If all is well, a given standards-compliant C program should compile with a standards-compliant C compiler and should run and produce equivalent results independent of the compiler chosen

Not all (any?) C compilers are fully standards-compliant

Not many C programs are fully standards-compliant

More Aside

Some compiler writers deliberately put support for non-standard things in their compilers: for reasons both good and bad

More Aside

Some compiler writers deliberately put support for non-standard things in their compilers: for reasons both good and bad

Good: to add extensions that are genuinely useful to the programmer

More Aside

Some compiler writers deliberately put support for non-standard things in their compilers: for reasons both good and bad

Good: to add extensions that are genuinely useful to the programmer

Bad: to add extensions that the programmer will become reliant on, so locking them in to using this particular compiler

More Aside

Some compiler writers deliberately put support for non-standard things in their compilers: for reasons both good and bad

Good: to add extensions that are genuinely useful to the programmer

Bad: to add extensions that the programmer will become reliant on, so locking them in to using this particular compiler

A program that follows the standards will be much more portable

More Aside

Gcc is a widely available and widely used free compiler on a large number of architectures that produces reasonable (but not the best) code

More Aside

Gcc is a widely available and widely used free compiler on a large number of architectures that produces reasonable (but not the best) code

It's not the case of paying more to get a better compiler...

More Aside

Gcc is a widely available and widely used free compiler on a large number of architectures that produces reasonable (but not the best) code

It's not the case of paying more to get a better compiler...

Many other C compilers exist: Intel; Clang; Microsoft; Norcroft; etc.

More Aside

Also, use a **text editor** (or an IDE) to write programs, not a word processor

More Aside

Also, use a **text editor** (or an IDE) to write programs, not a word processor

You are not that stupid are you?

More Aside

Also, use a **text editor** (or an IDE) to write programs, not a word processor

You are not that stupid are you?

And always use a **fixed width** font when printing out code. Layout is important in all languages, particularly in C

Running the program

```
% ./hello
hello world
```

Running the program

```
% ./hello
hello world
```

I usually include the `./` to ensure I run the program named `hello` that lives in the current directory, not some program of the same name from somewhere else in the system

C

The program is a stand-alone binary (machine instructions) which you can simply run

C

The program is a stand-alone binary (machine instructions) which you can simply run

It is compiled for a specific OS and hardware architecture, generally the machine you used the compiler on

C

The program is a stand-alone binary (machine instructions) which you can simply run

It is compiled for a specific OS and hardware architecture, generally the machine you used the compiler on

So that binary probably won't run on a different architecture: the program will need recompiling for other architectures

C

The program is a stand-alone binary (machine instructions) which you can simply run

It is compiled for a specific OS and hardware architecture, generally the machine you used the compiler on

So that binary probably won't run on a different architecture: the program will need recompiling for other architectures

There is no analogue to the `java` runtime program you need to run a Java program

C

Java: “write once, compile once, run everywhere”

C: “write once, compile everywhere, run everywhere”

C

Java: “write once, compile once, run everywhere”

C: “write once, compile everywhere, run everywhere”

Another trade-off

C

A bad program. `hello2.c`

```
#include <stdio.h>

int main(void)
{
  int n;

  n = n + 1;

  printf("hello world\n");

  return 0;
}
```

More Aside

```
% cc -Wall -o hello2 hello2.c
hello2.c: In function 'main':
hello2.c:7:5: warning: 'n' is used uninitialized
in this function
```

C

A simple example of a warning message

C

A simple example of a warning message

It is **very** important you get used to reading warning and error messages

C

A simple example of a warning message

It is **very** important you get used to reading warning and error messages

You will see loads!

C

A simple example of a warning message

It is **very** important you get used to reading warning and error messages

You will see loads!

Get used to them, and get used to fixing the problems they refer to: don’t ignore warnings

C

A simple example of a warning message

It is **very** important you get used to reading warning and error messages

You will see loads!

Get used to them, and get used to fixing the problems they refer to: don't ignore warnings

The quality of error messages varies with the compiler. Gcc produces generally reasonable messages

C

A simple example of a warning message

It is **very** important you get used to reading warning and error messages

You will see loads!

Get used to them, and get used to fixing the problems they refer to: don't ignore warnings

The quality of error messages varies with the compiler. Gcc produces generally reasonable messages

(In this case, the compiler happens to generate an executable; for more serious errors it wouldn't)