CSE127, Computer Security

Course introduction, definitions, reflections on trusting trust

UC San Diego

whoami

Deepak Kumar
Assistant Professor in CSE

undergrad → grad → postdoc

Background on me (research)

- I work primarily on applied computer security research, my research interests are in *sociotechnical security* (computer security + human-computer interaction)
  - Interests primarily in ways that technology + society interact, and where security or safety problems arise (e.g., online harassment, mis/disinformation, AI generated deepfakes, trust on the Internet, etc.)
- Mostly, I'm a data + systems guy…. e.g.,
  - "What does the marketplace for nonconsensual sexual deepfake creation look like?"
  - "How much toxic content there on Reddit, and what can we learn about attack patterns than inform defenses?"
  - "How can we build better defenses for journalists facing online harassment on social media?"

How’d I get into security?

- I took the UMich version of this class 11 years ago (Winter 2015)
- Two versions of my story (both are true)
    - I’ve always been very interested in technology + society, and computer security is a field that by definition gets to impact both of those interests
    - I *wanted* to do computer architecture research in undergrad, but I didn’t get a good enough grade. The security group was looking for students and I got an A in the 127-equivalent, so they took me, and the rest is history
- Started in network security, moved my way towards more human-centered work

I also do other things…

Baked! The Musical

THE OLD GLOBE

Playwright Deepak Kumar, whose "House of India" play will make its world premiere at the Old Globe in 2025. (The Old Globe)

'House of India'

This world premiere play was written by San Diego resident Deepak Kumar, who is both a playwright and a computer science assistant professor at UC San Diego. The play is set in a family-owned Indian restaurant in a Cleveland-area strip mall, where Ananya has reluctantly taken over after her husband's death. She's faced with mounting bills and disagreement between her children over whether to stick to her husband's traditional menu or modernize with a quick-service, fusion concept. "It's about the American question of holding on to our traditions versus assimilating into the mainstream," Edelstein said. It will be directed by Zi Alikhan (Pasadena Playhouse's "Sanctuary City"). *May 10-June 1.*

What are we (UCSD) known for in security?

- Measurement (cybercrime, malware, spam, captchas, fraud, etc.)
- Defenses (threat intelligence, cyber hygiene, etc.)
- Embedded Security (hacking cars, voting machines, airplanes, credit card skimmers, medical devices)
- Web security + PLsec (cookies, information flow, wasm runtime shenanigans)
- Intersection of crypto + security (turns out, implementing crypto is very hard)
- Lots of faculty here working on stuff — more here
  - https://cryptosec.ucsd.edu
  - http://sysnet.ucsd.edu

Course Staff

Bella Jeong
TA
ljeong@ucsd.edu

Manan Patel
Tutor
mbp001@ucsd.edu

5-Minute Introductions

- Find two other people in the classroom you're sitting nearby (ideally people you haven't met)
- Write and send us an email introducing yourselves to us (one per group is fine):
  - Names, degree programs + year progress (e.g., junior, senior)
  - Why did you enroll in this class?
  - What is something you do for fun?
  - Attach a selfie!
- Send introductions to cse-127staff-g@ucsd.edu

Let's recap the last 5-minutes of your life

Let’s recap the last 5-minutes of your life

- Some guy (me) who most of you just met stood up and claimed to be the professor of the class

Let’s recap the last 5-minutes of your life

- Some guy (me) who most of you just met stood up and claimed to be the professor of the class
- He asked you to do a task that required some effort (and therefore took up your precious time)

Let’s recap the last 5-minutes of your life

- Some guy (me) who most of you just met stood up and claimed to be the professor of the class
- He asked you to do a task that required some effort (and therefore took up your precious time)
- He asked you to *send a compilation of information about yourselves* to a random email address, including a picture!?

Let’s recap the last 5-minutes of your life

- Some guy (me) who most of you just met stood up and claimed to be the professor of the class
- He asked you to do a task that required some effort (and therefore took up your precious time)
- He asked you to *send a compilation of information about yourselves* to a random email address, including a picture!?

Why did you do that?

Security is all about trust

You can’t have security if you trust no one

Security is all about trust

You can't have security if you trust no one

- With those same groups (no tricks this time), answer the following questions:
  - What is security?
  - What is *computer* security?
  - What is *trust*?

Security is all about trust

You can't have security if you trust no one

- With those same groups (no tricks this time), answer the following questions:
  - **What is security?**
  - What is *computer* security?
  - What is *trust*?

Definitions: Security

- Merriam-Webster online dictionary:
    - The quality or state of being <u>secure:</u> such as
        - Free from *danger* : safety
        - Freedom from fear or anxiety
        - Freedom for the prospect of being laid off (job security)

Definitions: Security

- Merriam-Webster online dictionary:
  - The quality or state of being secure: such as
    - Free from *danger* : safety
    - Freedom from fear or anxiety
    - Freedom for the prospect of being laid off (job security)
- Note: Security is about *freedom* (from some entity, force, or otherwise)

Security is all about trust

You can't have security if you trust no one

- With those same groups (no tricks this time), answer the following questions:
  - What is security?
  - **What is *computer* security?**
  - What is *trust*?

Definitions: Computer Security

- Most of computer science is about **functionality:**
  - UX/UI
  - Architecture
  - AI / ML development
  - Operating Systems / Networking / Databases
  - Compilers / PL
  - Microarchitecture
- Computer security is ***not*** about functionality
- Computer security is the ***study of a computer system in the presence of an adversary***
- Holistic property:
  - "Software security is about integrating security practices into the way you build software, not integrating security features into your code" – Gary McGraw, ex-VP of Synopsys

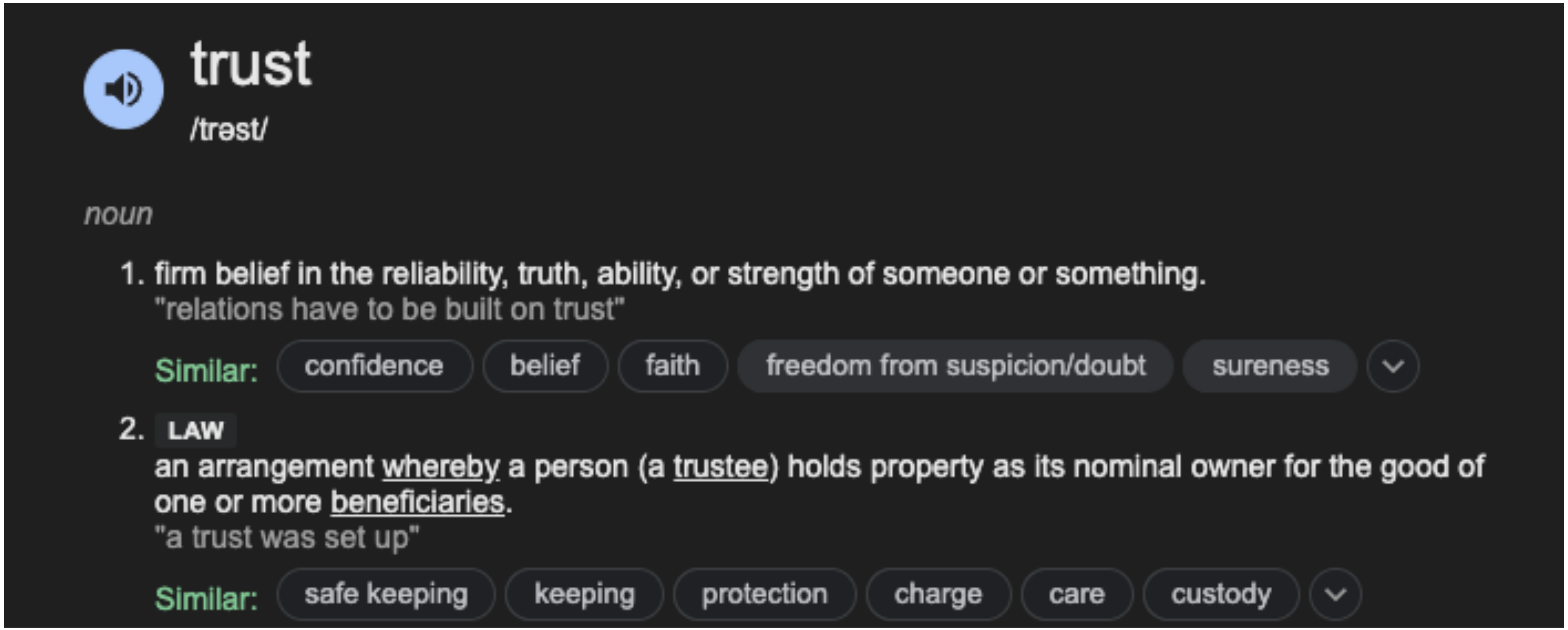
Security is all about trust

You can't have security if you trust no one

- With those same groups (no tricks this time), answer the following questions:
  - What is security?
  - What is *computer* security?
  - **What is *trust*?**

Security is all about trust

You can't have security if you trust no one



trust

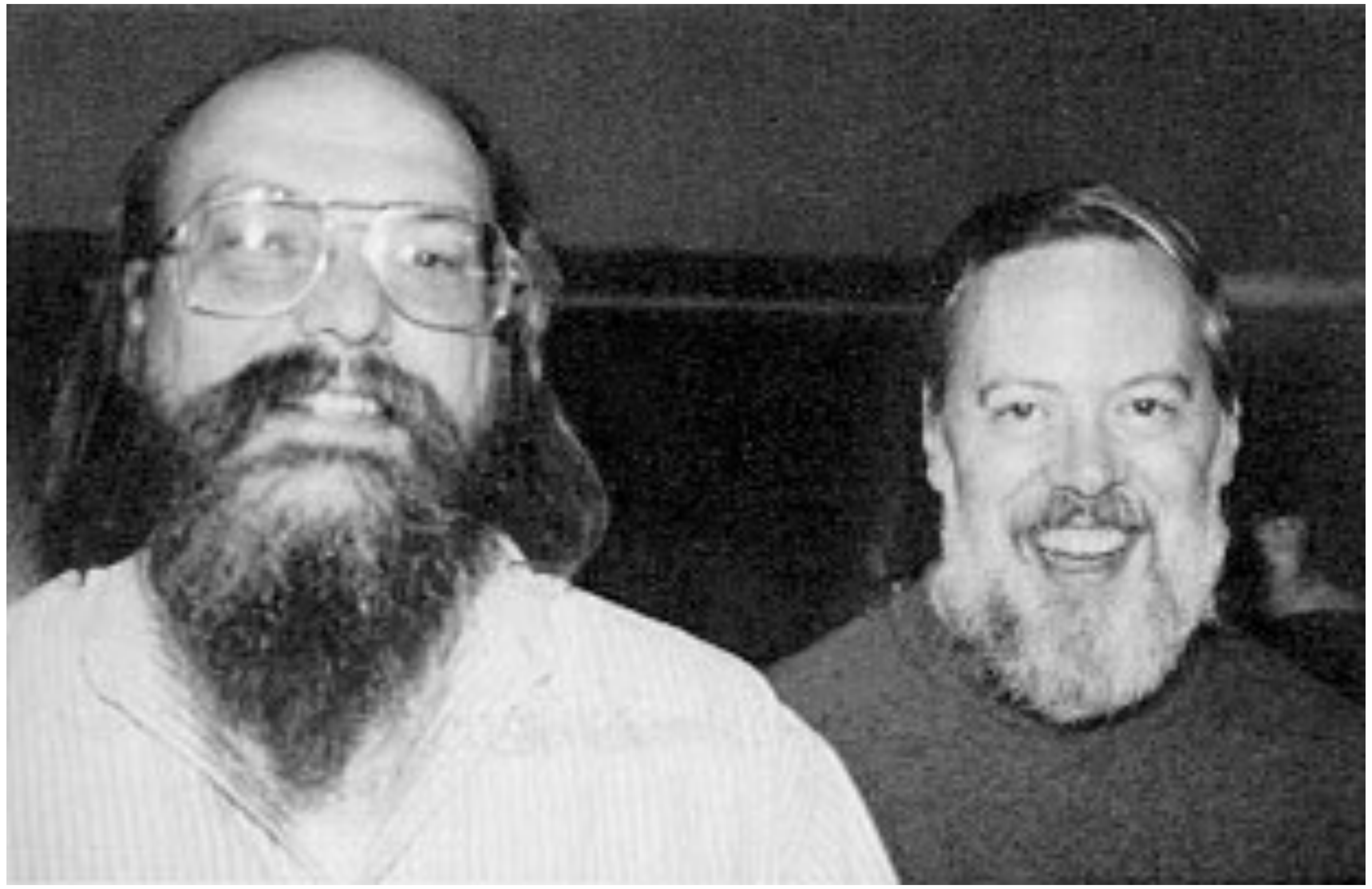
/trəst/

noun

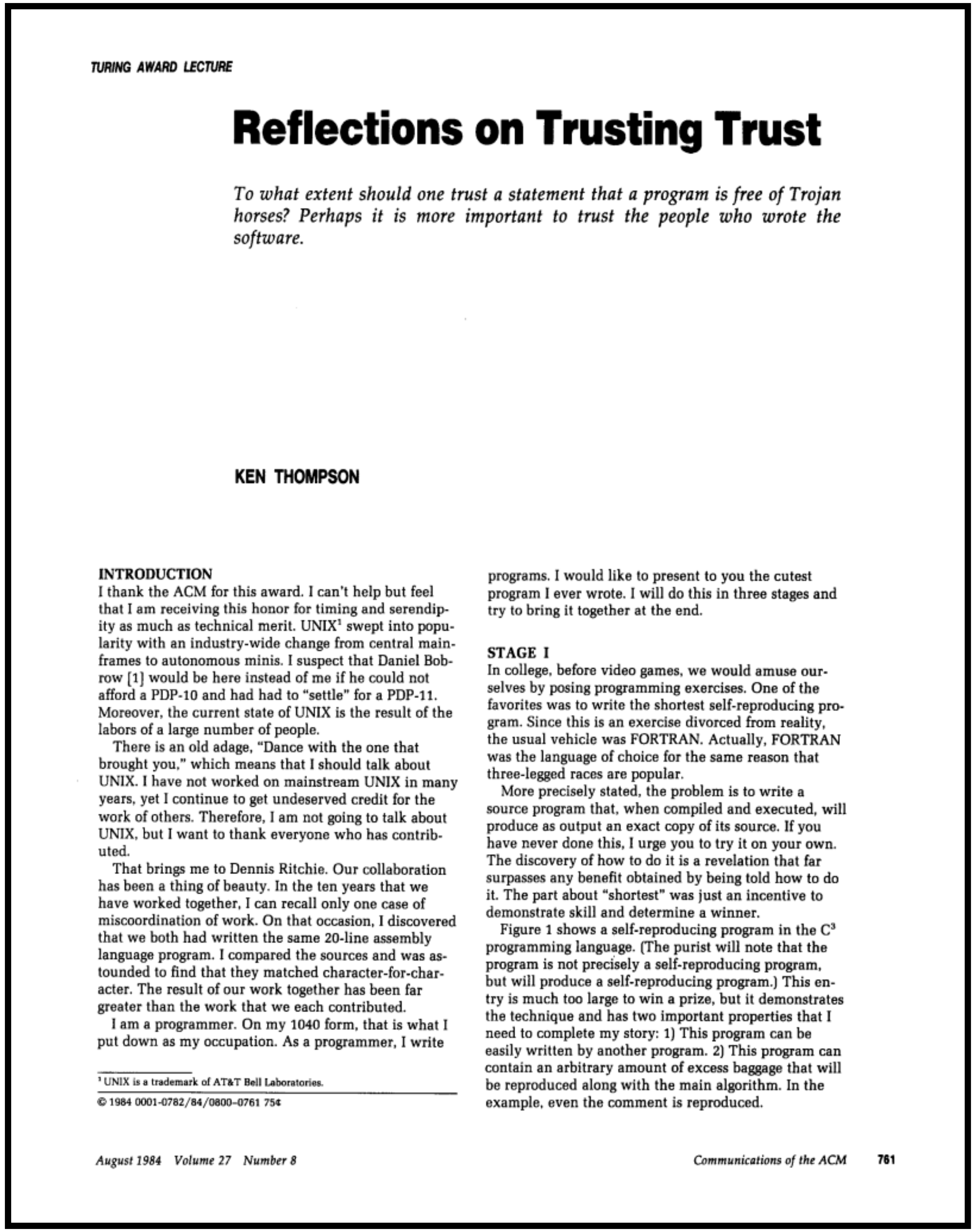
1. firm belief in the reliability, truth, ability, or strength of someone or something.
   "relations have to be built on trust"
   Similar: confidence · belief · faith · freedom from suspicion/doubt · sureness
2. LAW
   an arrangement whereby a person (a trustee) holds property as its nominal owner for the good of one or more beneficiaries.
   "a trust was set up"
   Similar: safe keeping · keeping · protection · charge · care · custody

Reflections on Trusting Trust

1984 Turing Award Lecture



Ken Thompson + Dennis Ritchie



Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

INTRODUCTION

I thank the ACM for this award. I can't help but feel that I am receiving this honor for timing and serendipity as much as technical merit. UNIX[1] swept into popularity with an industry-wide change from central mainframes to autonomous minis. I suspect that Daniel Bobrow [1] would be here instead of me if he could not afford a PDP-10 and had had to "settle" for a PDP-11. Moreover, the current state of UNIX is the result of the labors of a large number of people.

There is an old adage, "Dance with the one that brought you," which means that I should talk about UNIX. I have not worked on mainstream UNIX in many years, yet I continue to get undeserved credit for the work of others. Therefore, I am not going to talk about UNIX, but I want to thank everyone who has contributed.

That brings me to Dennis Ritchie. Our collaboration has been a thing of beauty. In the ten years that we have worked together, I can recall only one case of miscoordination of work. On that occasion, I discovered that we both had written the same 20-line assembly language program. I compared the sources and was astounded to find that they matched character-for-character. The result of our work together has been far greater than the work that we each contributed.

I am a programmer. On my 1040 form, that is what I put down as my occupation. As a programmer, I write programs. I would like to present to you the cutest program I ever wrote. I will do this in three stages and try to bring it together at the end.

STAGE I

In college, before video games, we would amuse ourselves by posing programming exercises. One of the favorites was to write the shortest self-reproducing program. Since this is an exercise divorced from reality, the usual vehicle was FORTRAN. Actually, FORTRAN was the language of choice for the same reason that three-legged races are popular.

More precisely stated, the problem is to write a source program that, when compiled and executed, will produce as output an exact copy of its source. If you have never done this, I urge you to try it on your own. The discovery of how to do it is a revelation that far surpasses any benefit obtained by being told how to do it. The part about "shortest" was just an incentive to demonstrate skill and determine a winner.

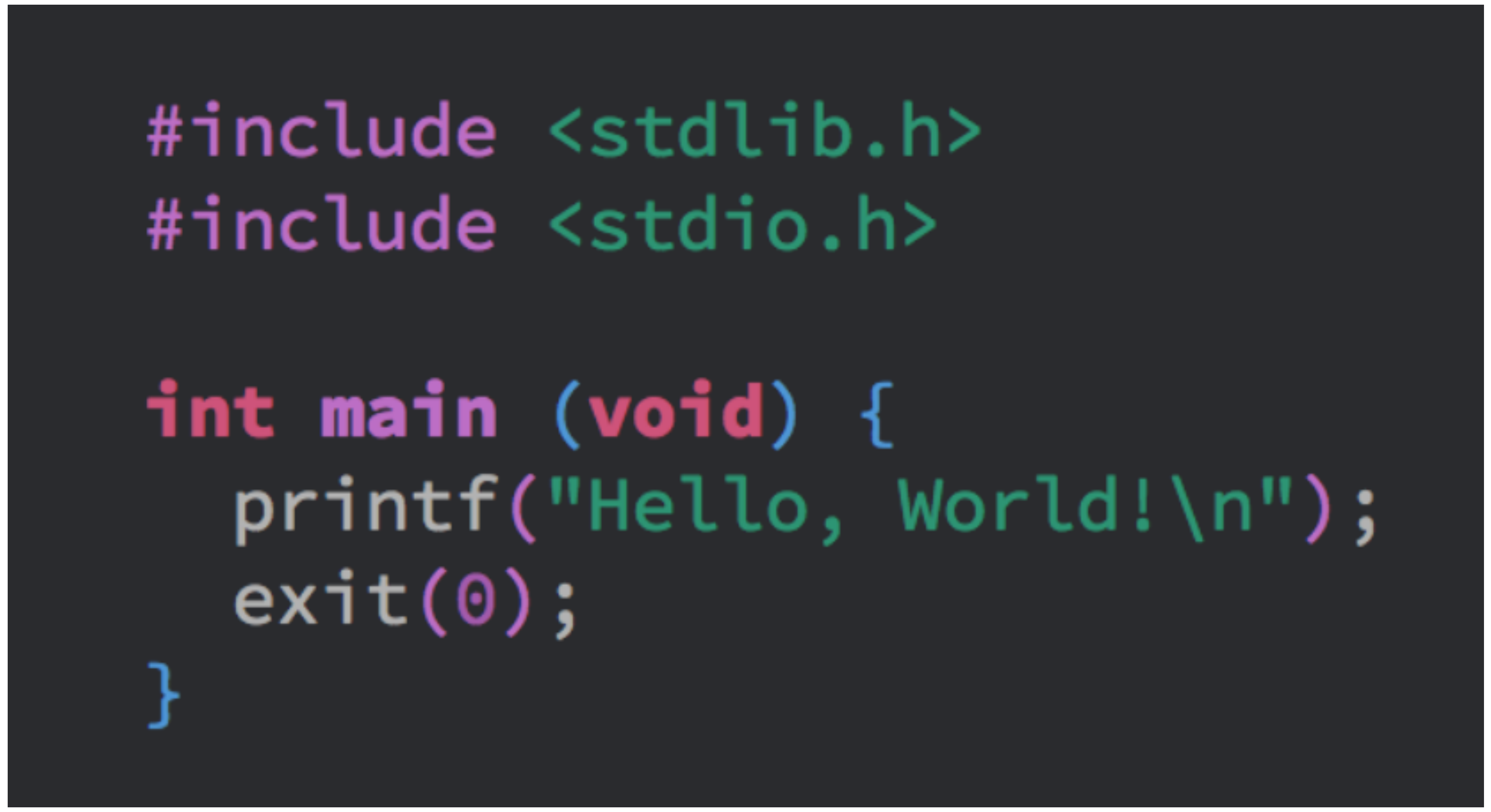
Figure 1 shows a self-reproducing program in the C[3] programming language. (The purist will note that the program is not precisely a self-reproducing program, but will produce a self-reproducing program.) This entry is much too large to win a prize, but it demonstrates the technique and has two important properties that I need to complete my story: 1) This program can be easily written by another program. 2) This program can contain an arbitrary amount of excess baggage that will be reproduced along with the main algorithm. In the example, even the comment is reproduced.

[1] UNIX is a trademark of AT&T Bell Laboratories.

Reflections on Trusting Trust

How do we run C programs?

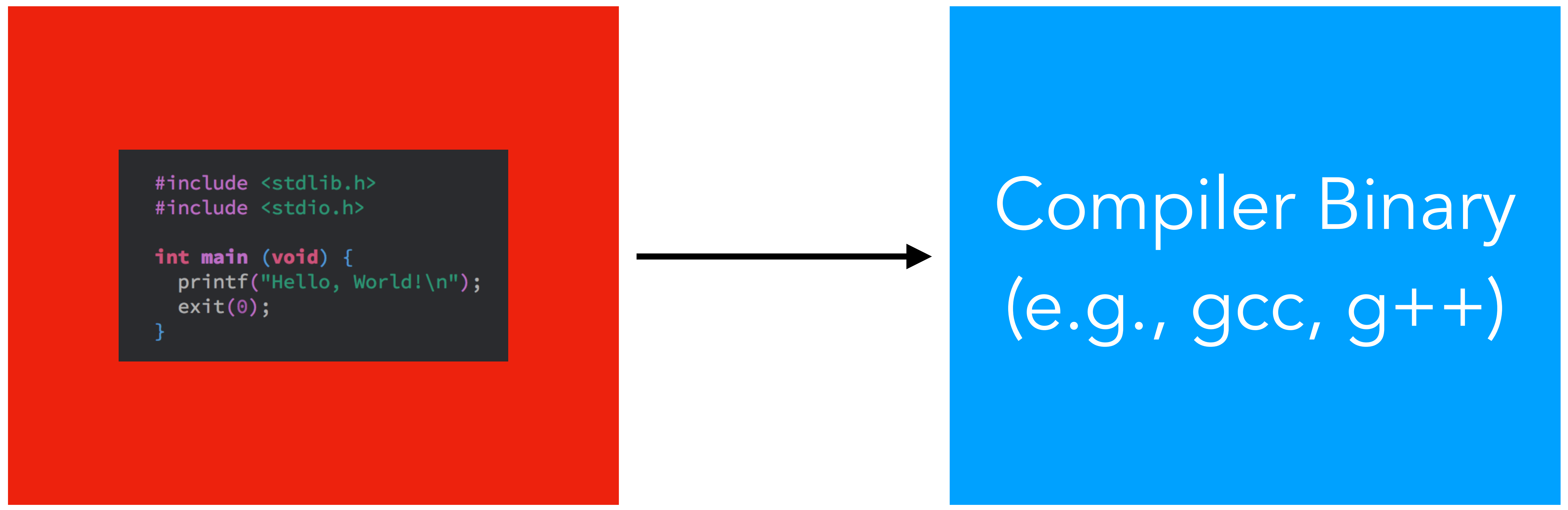
C Program



```c
#include <stdlib.h>
#include <stdio.h>

int main (void) {
  printf("Hello, World!\n");
  exit(0);
}
```

Reflections on Trusting Trust

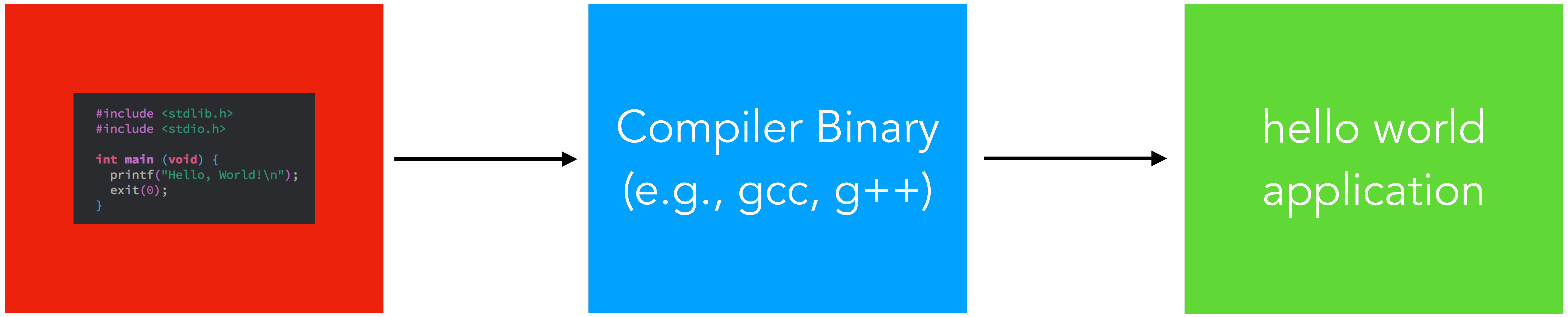
How do we run C programs?



```
#include <stdlib.h>
#include <stdio.h>

int main (void) {
  printf("Hello, World!\n");
  exit(0);
}
```

Reflections on Trusting Trust

How do we run C programs?



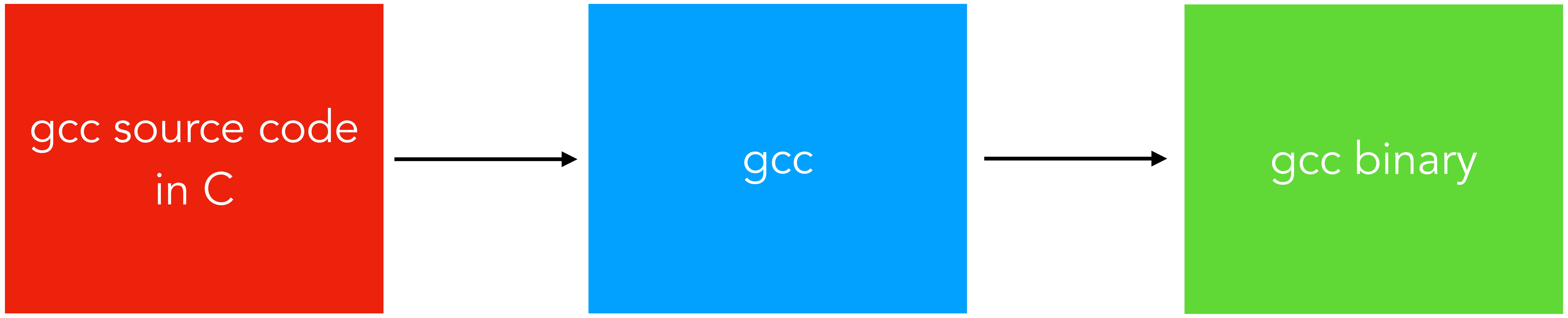
```
#include <stdlib.h>
#include <stdio.h>

int main (void) {
  printf("Hello, World!\n");
  exit(0);
}
```

What language is gcc written in?

What language is gcc written in?

Surprise! It’s C.

gcc compiles gcc

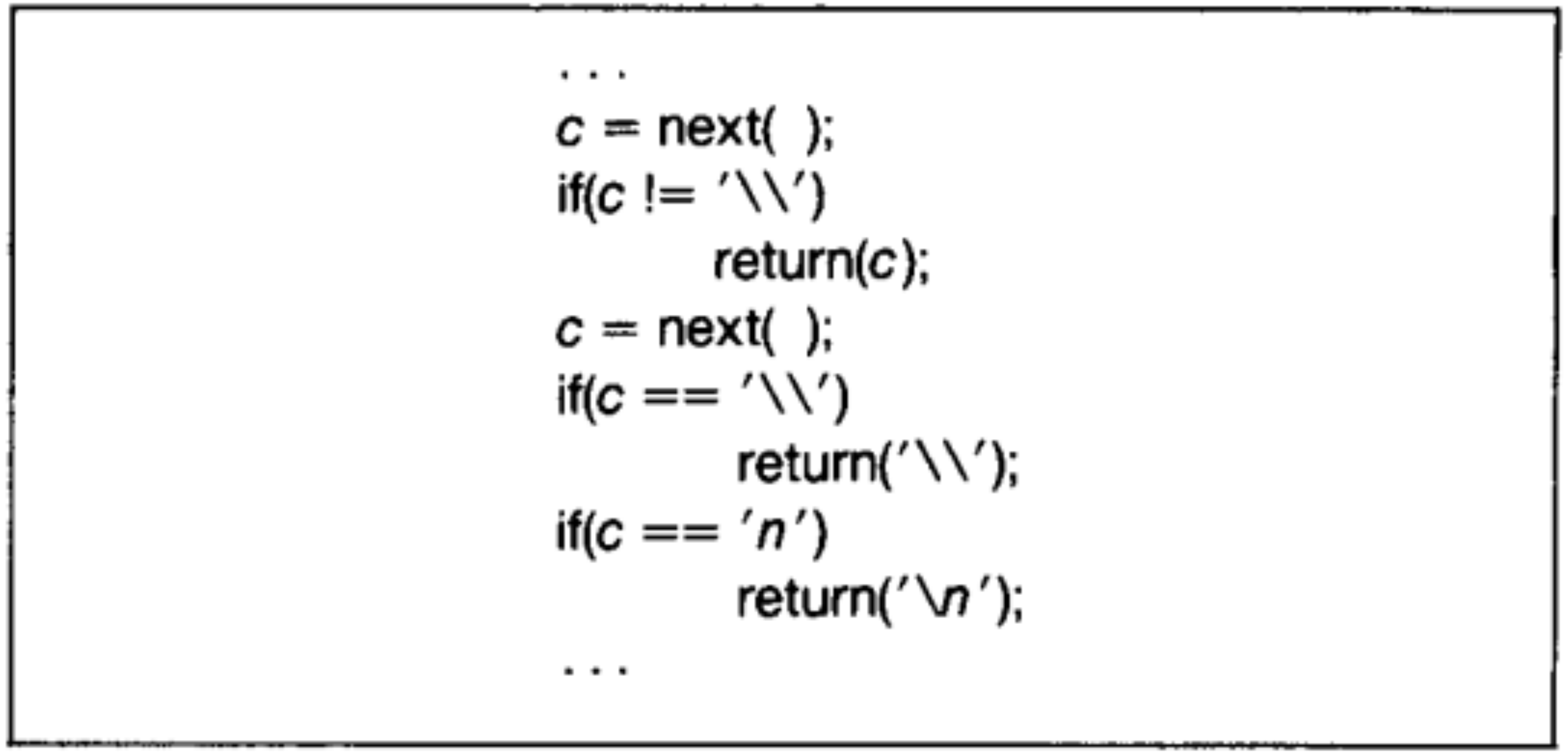


gcc compiles gcc

Simple function for parsing and escaping characters

gcc source code in C

```
char str[] = "Hello world\n";
```



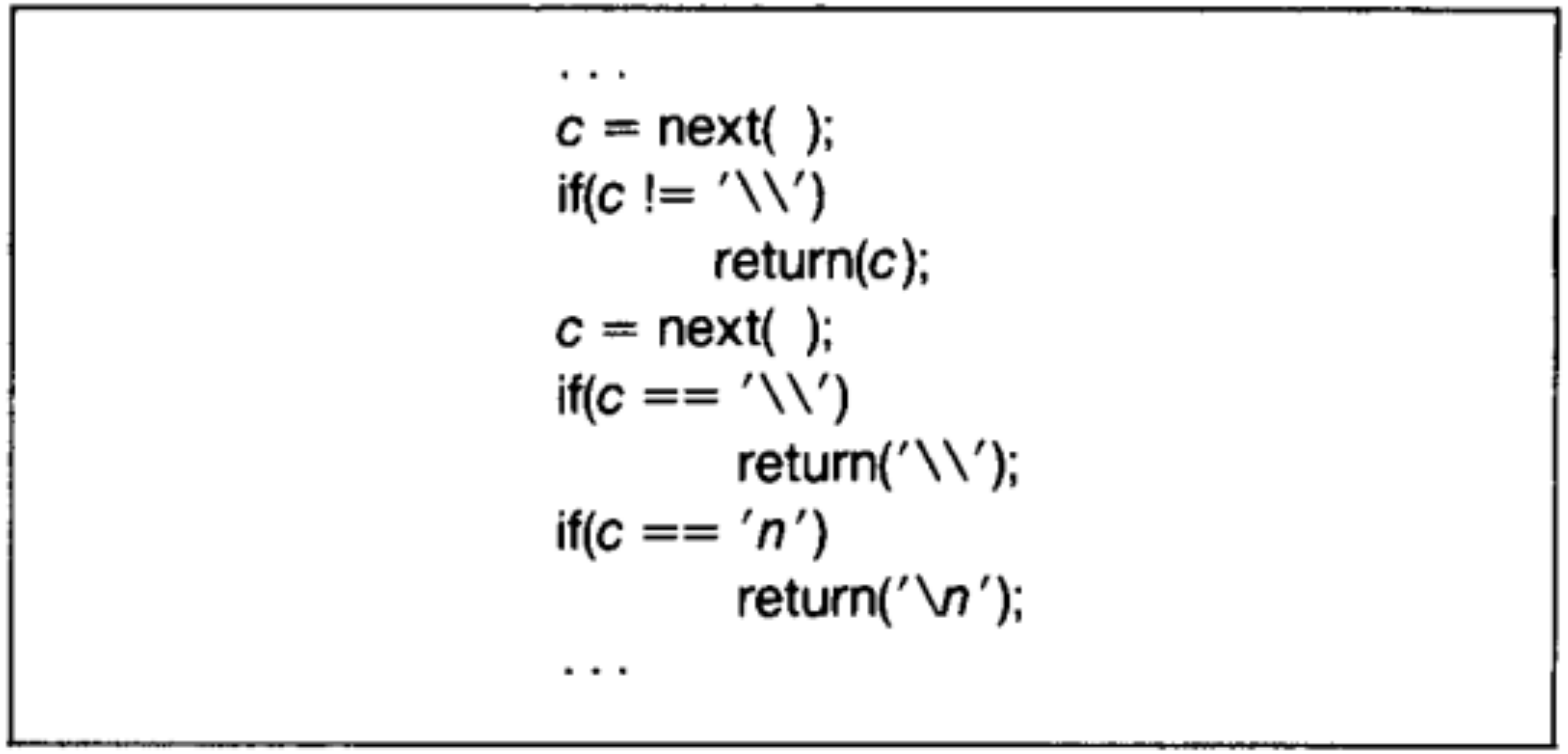
```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
...
```

gcc compiles gcc

Let’s say I wanted to add ‘\v’ – how would I do that?

gcc source code in C

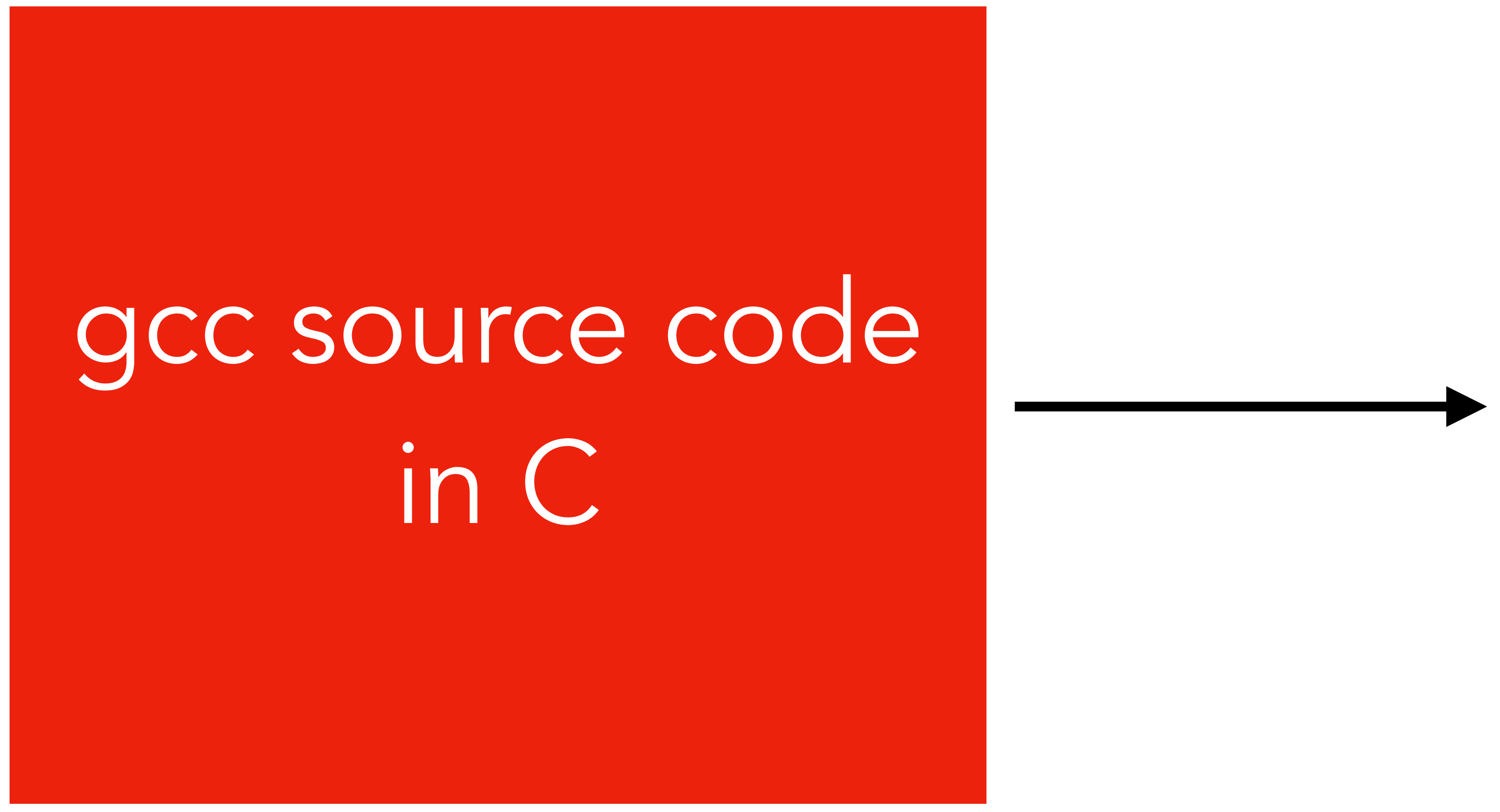
```
char str[] = “Hello world\v”;
```



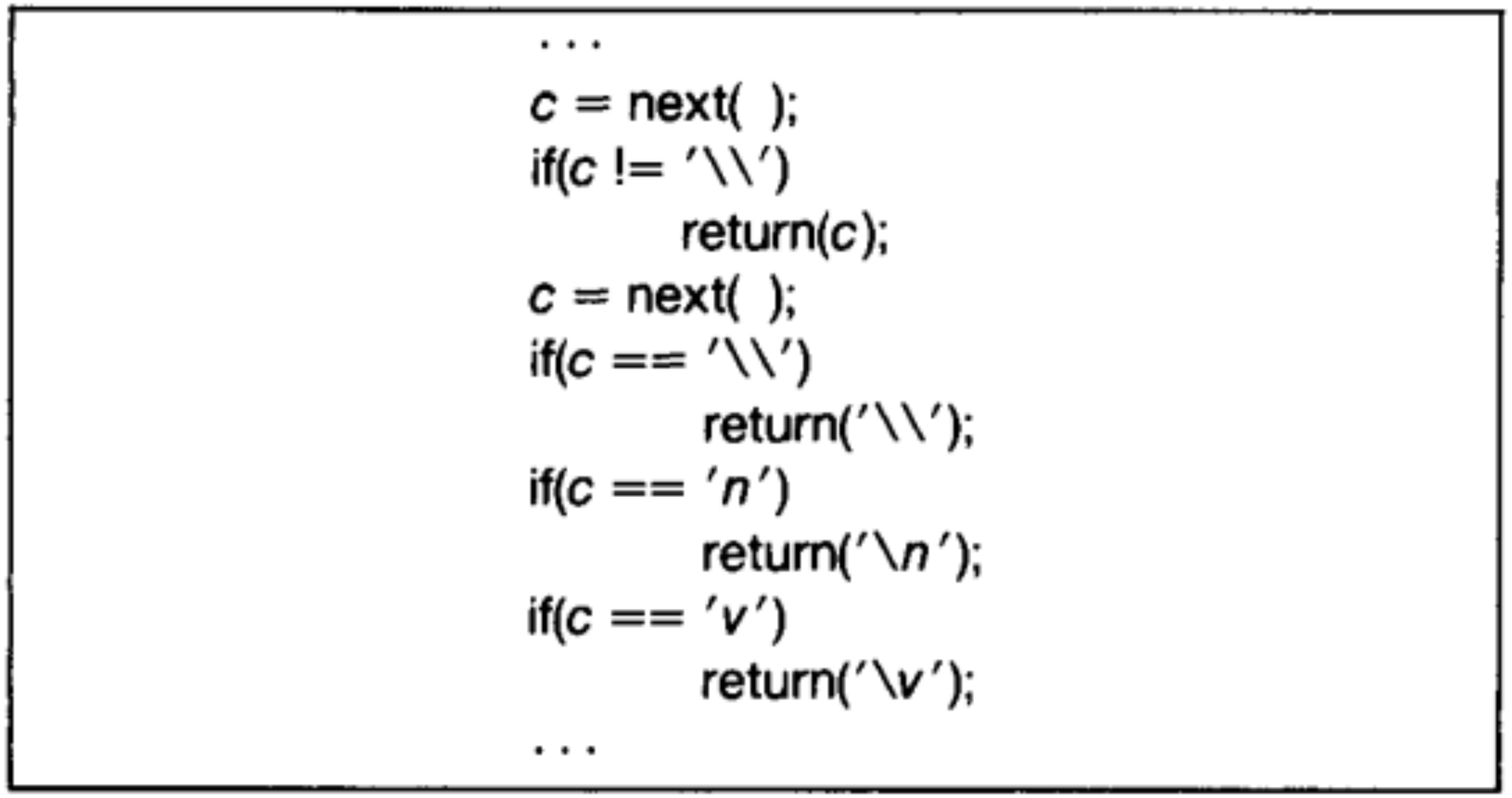
```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
...
```

gcc compiles gcc

Let’s say I wanted to add ‘\v’ – how would I do that?



```
char str[] = “Hello world\v”;
```

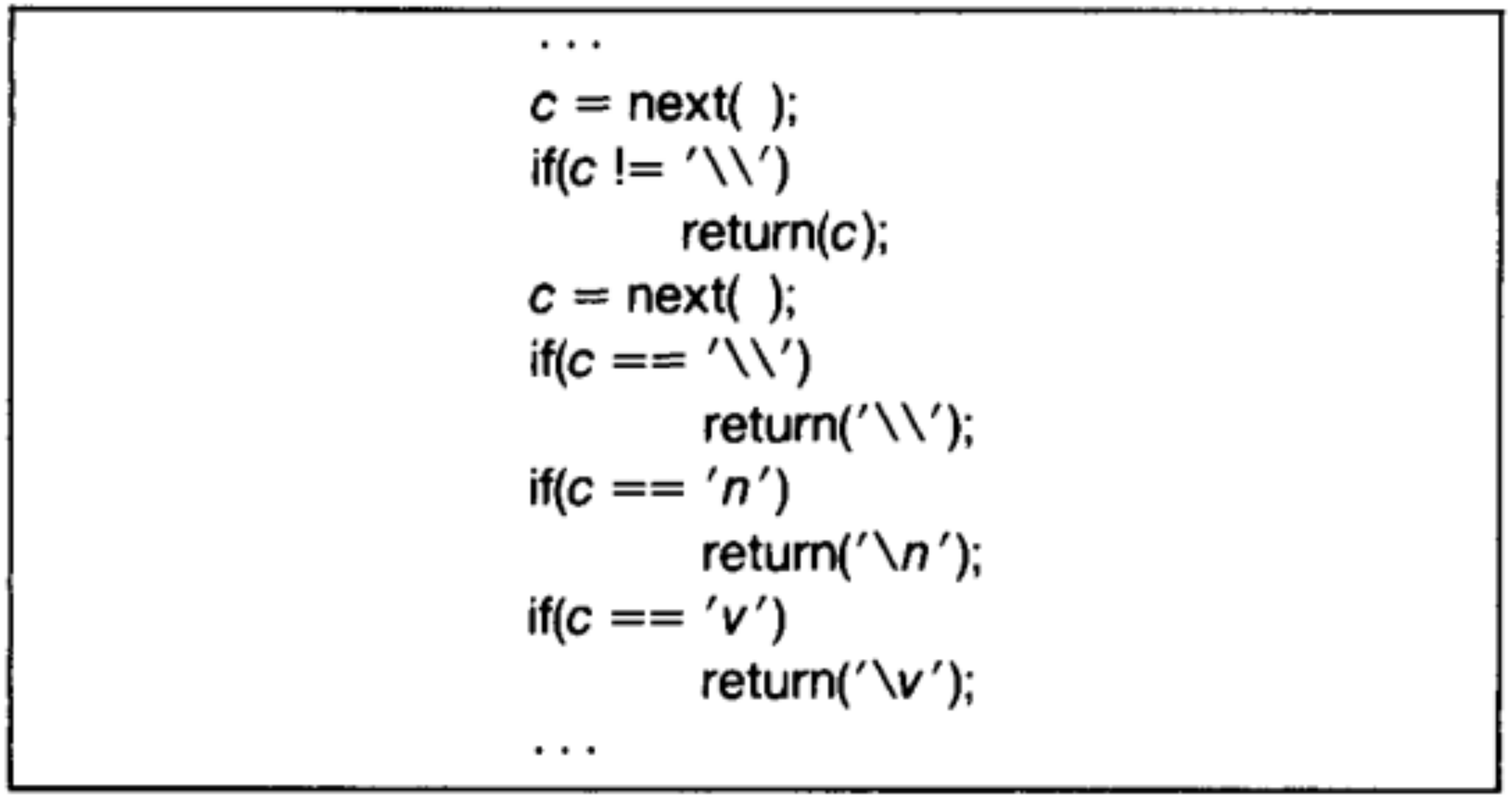


```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return('\v');
...
```

gcc compiles gcc

Let’s say I wanted to add ‘\v’ – how would I do that?

gcc source code in C

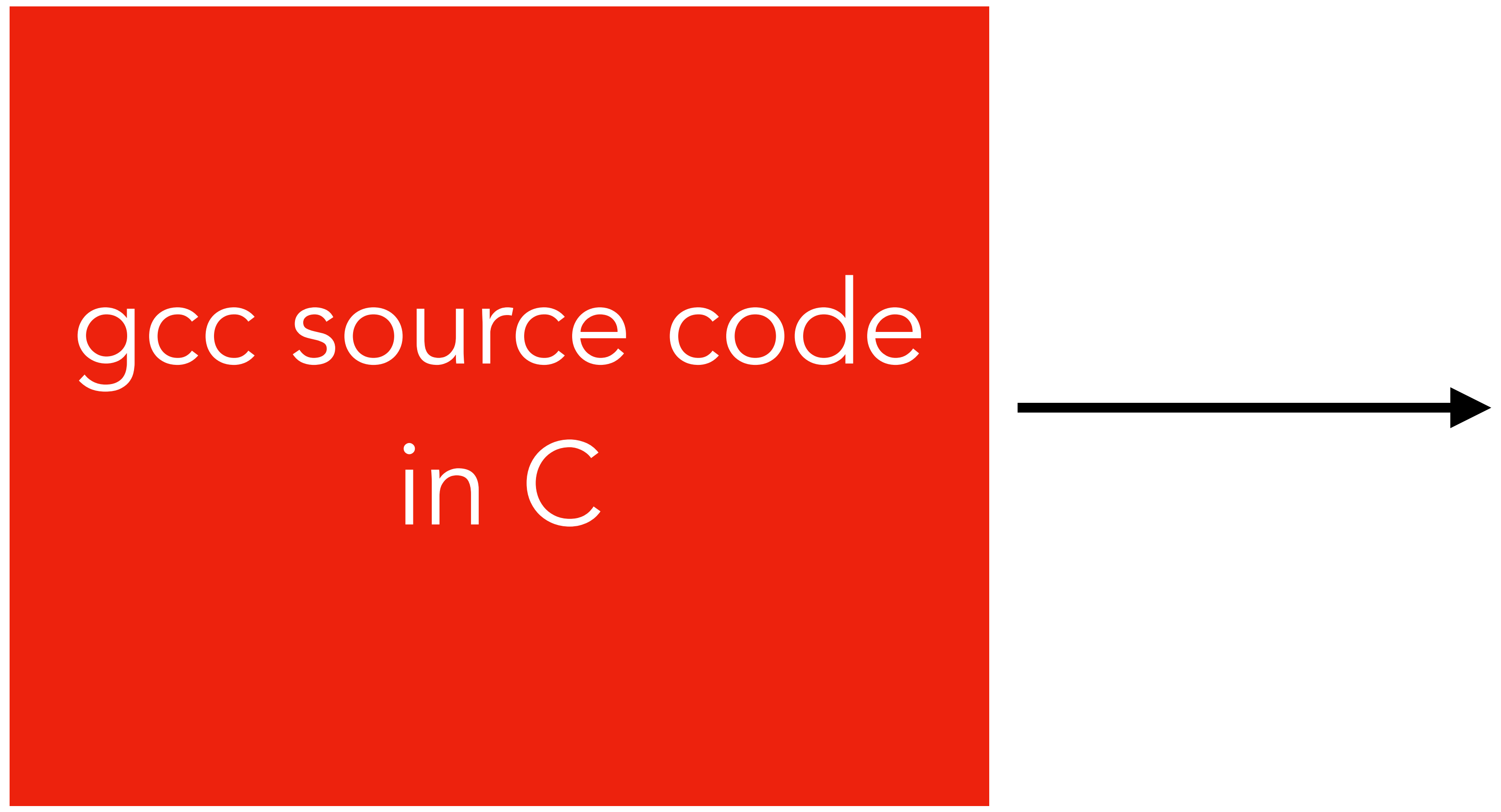


```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return('\v');
...
```

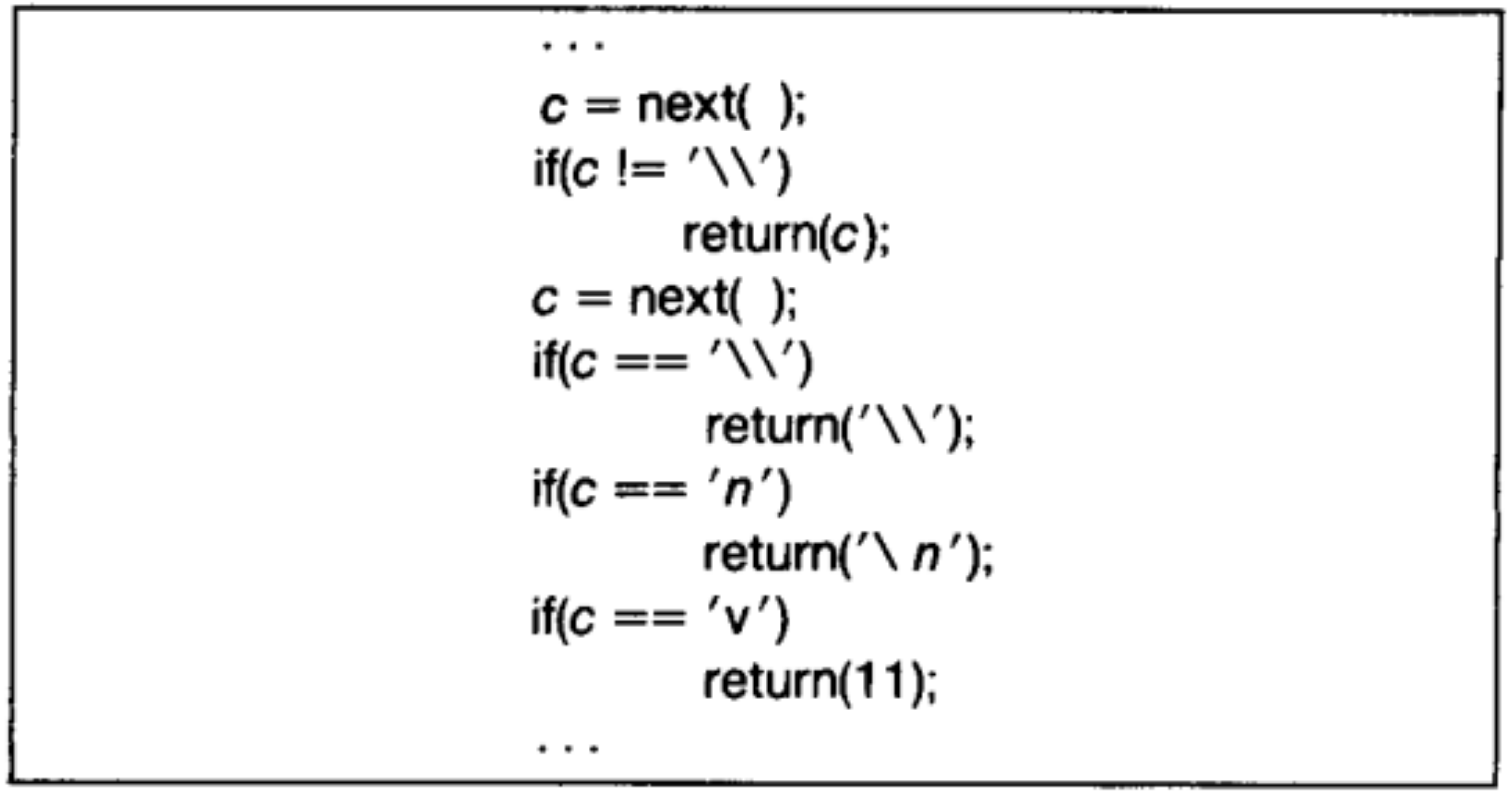
This will throw a compilation error.
Why?

gcc compiles gcc

Let's say I wanted to add '\v' – how would I do that?



```
char str[] = "Hello world\v";
```



```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return(11);
...
```

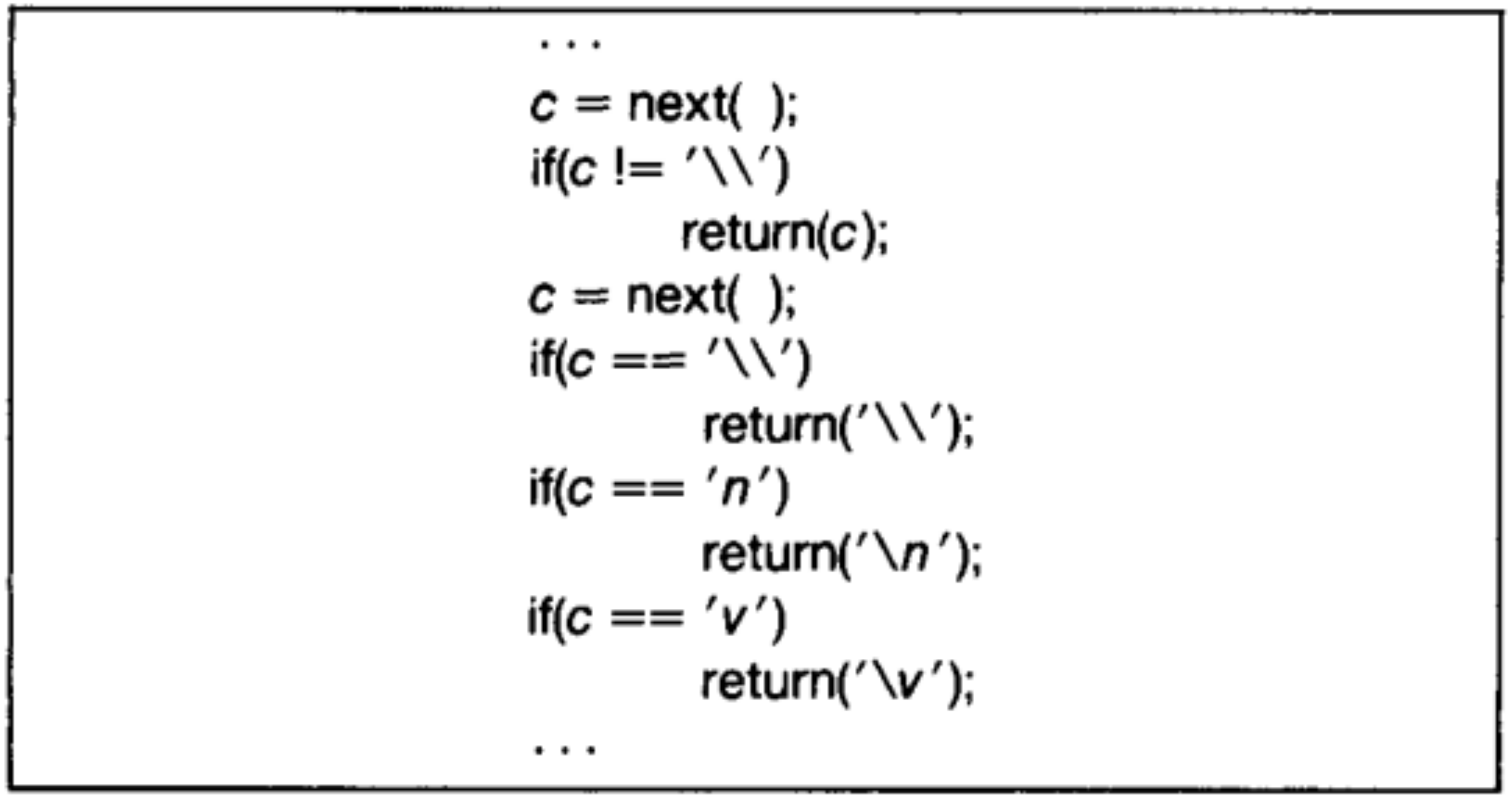
We have to tell the C compiler about `\v` before we can use `\v`

gcc compiles gcc

Let's say I wanted to add '\v' – how would I do that?

gcc source code in C

```
char str[] = "Hello world\v";
```

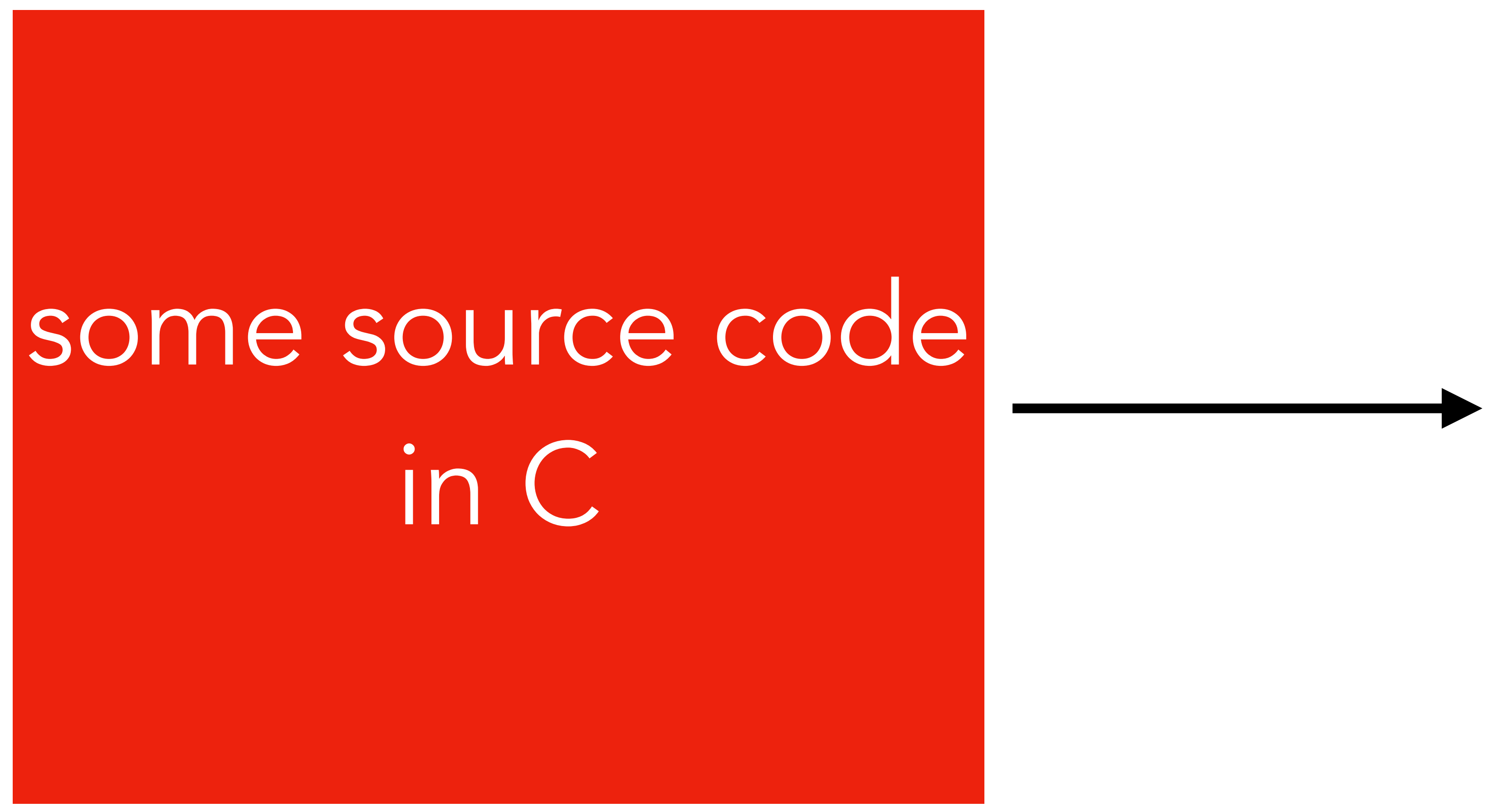


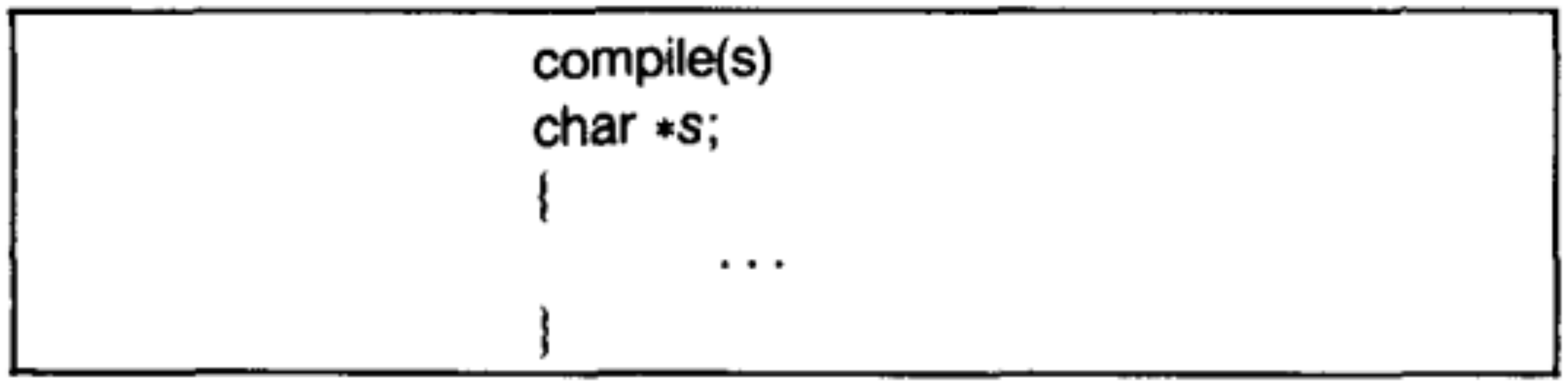
```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return('\v');
...
```

Now, this will compile!

If you own gcc, you can backdoor anything

Trojan Horsing the C compiler to return malicious binaries





```
compile(s)
char *s;
{
        ...
}
```

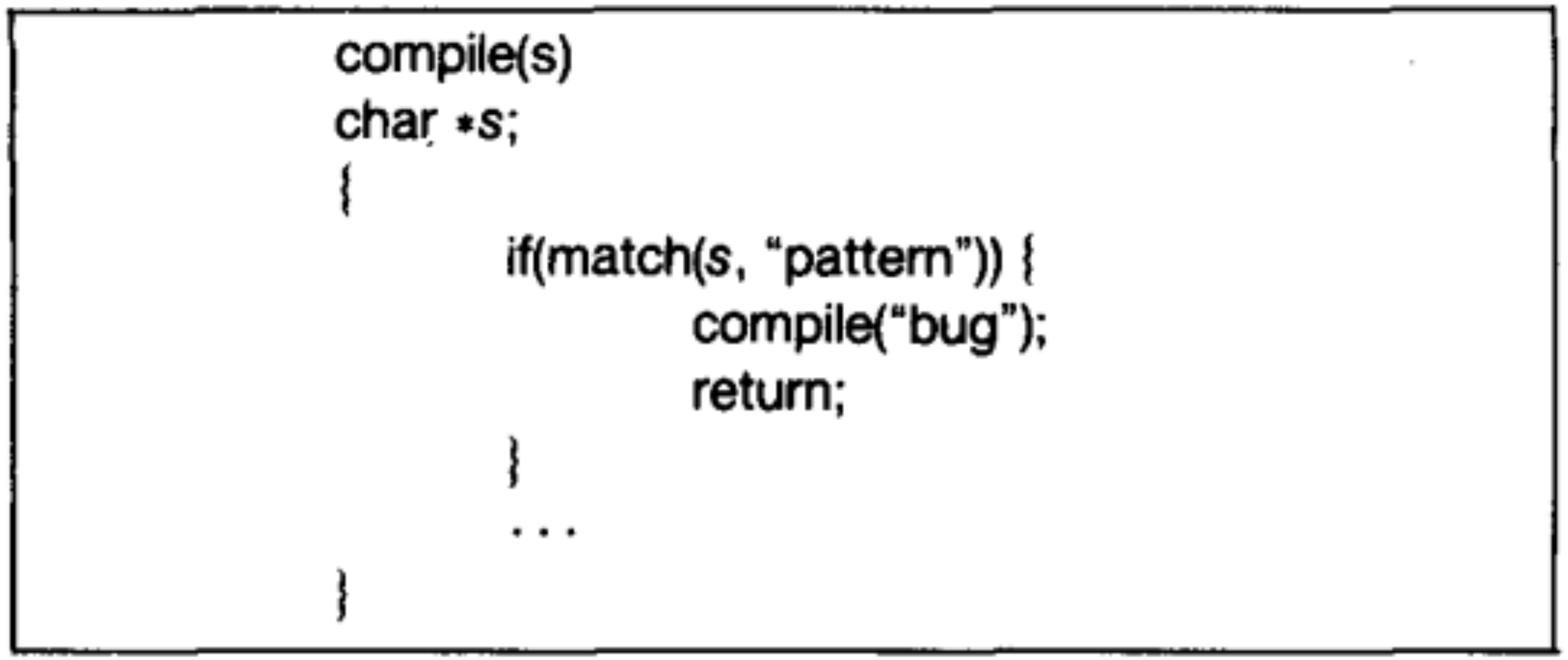
compile(s) compiles the next line of source code

If you own gcc, you can backdoor anything

Trojan Horsing the C compiler to return malicious binaries

some source code in C

$\longrightarrow$

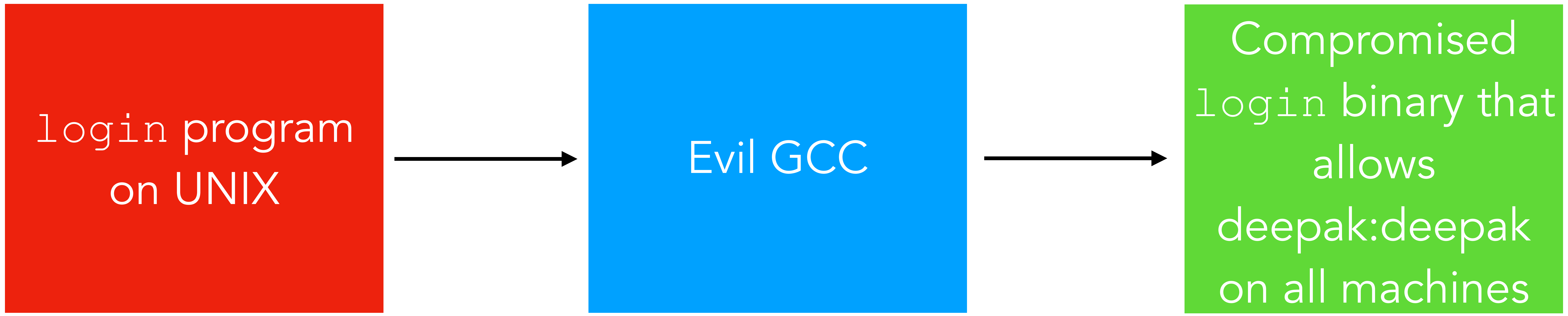


```
compile(s)
char *s;
{
        if(match(s, "pattern")) {
                compile("bug");
                return;
        }
        ...
}
```

If I owned the C compiler, I *could* add logic that introduces malicious bugs when certain patterns appear, e.g., if trying to compile "login" program

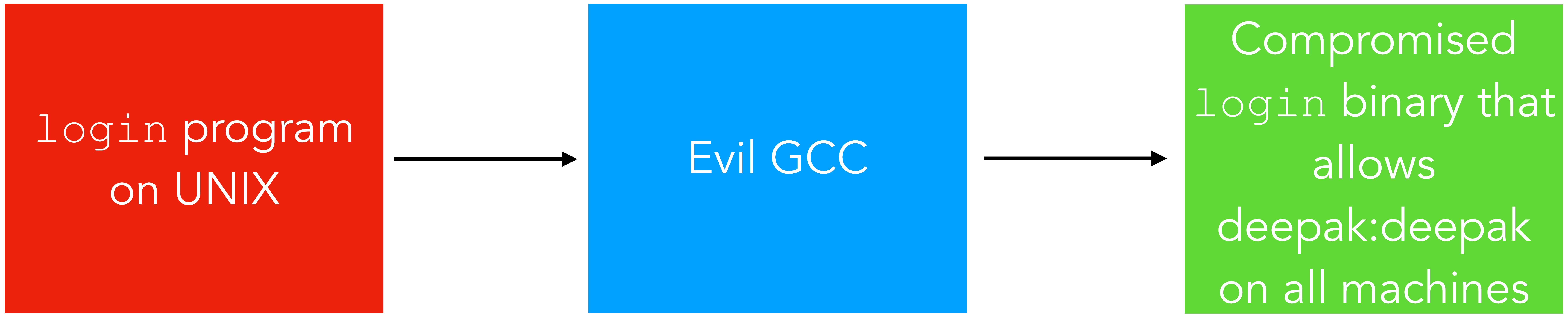
If you own gcc, you can backdoor anything

Trojan Horsing the C compiler to return malicious binaries


`login` program on UNIX
Evil GCC
Compromised `login` binary that allows deepak:deepak on all machines

If you own gcc, you can backdoor anything

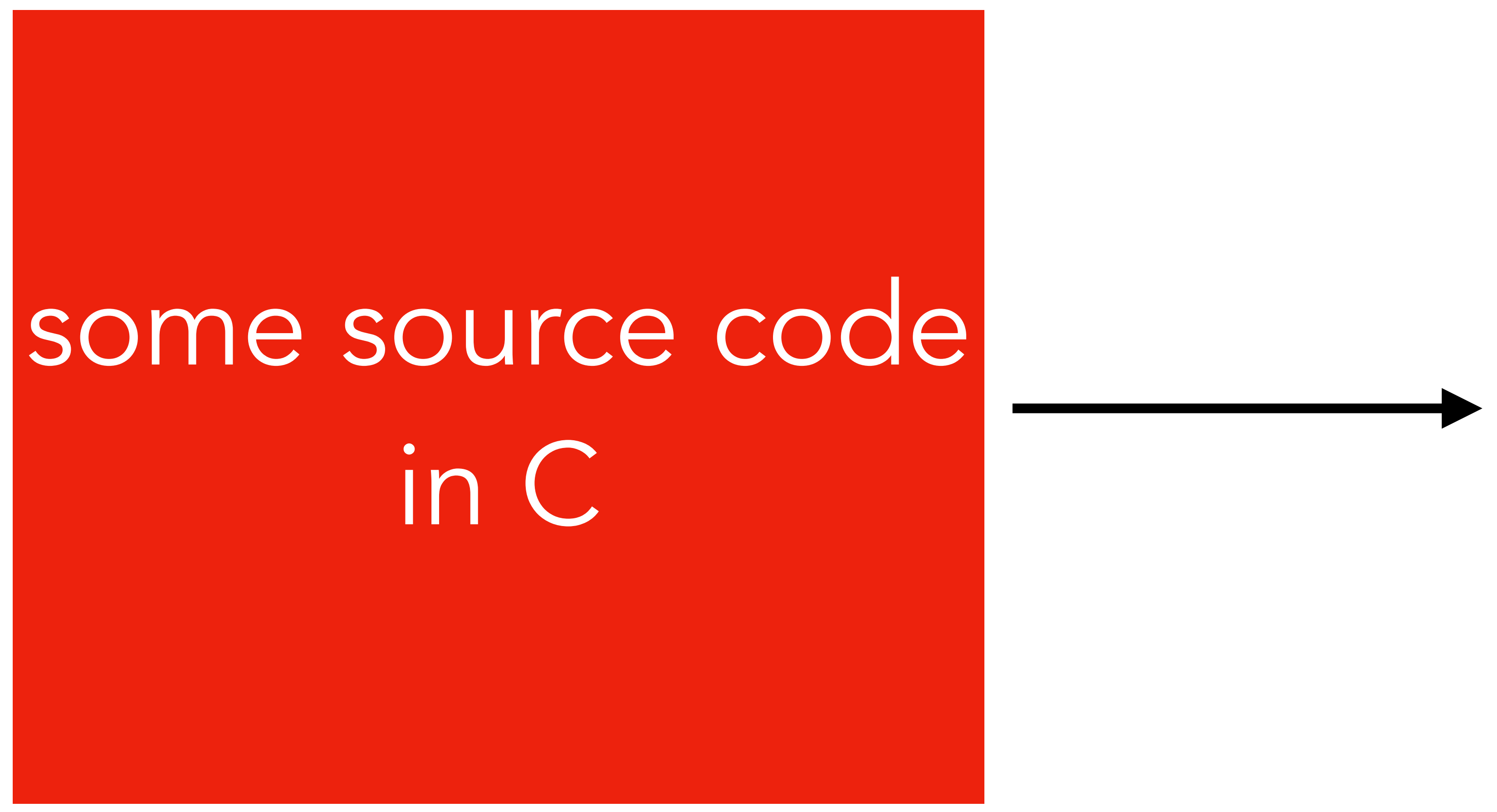
Trojan Horsing the C compiler to return malicious binaries

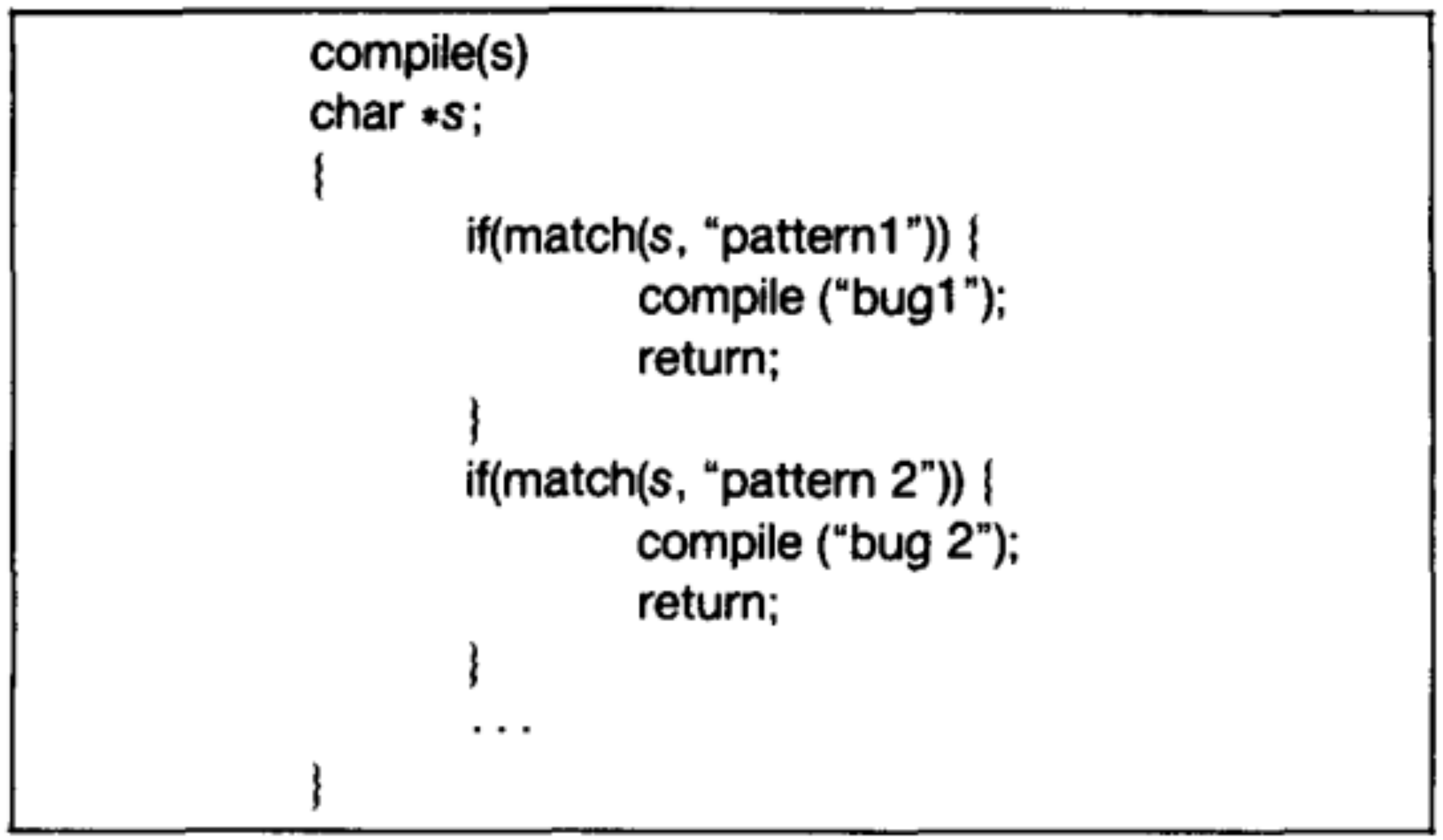

login program on UNIX
Evil GCC
Compromised login binary that allows deepak:deepak on all machines

… but this is very easy to detect if you read the C compiler code. Why?

If you own gcc, you can be even more stealthy

Trojan horsing the C compiler to bug code without you knowing it's trojan horsed





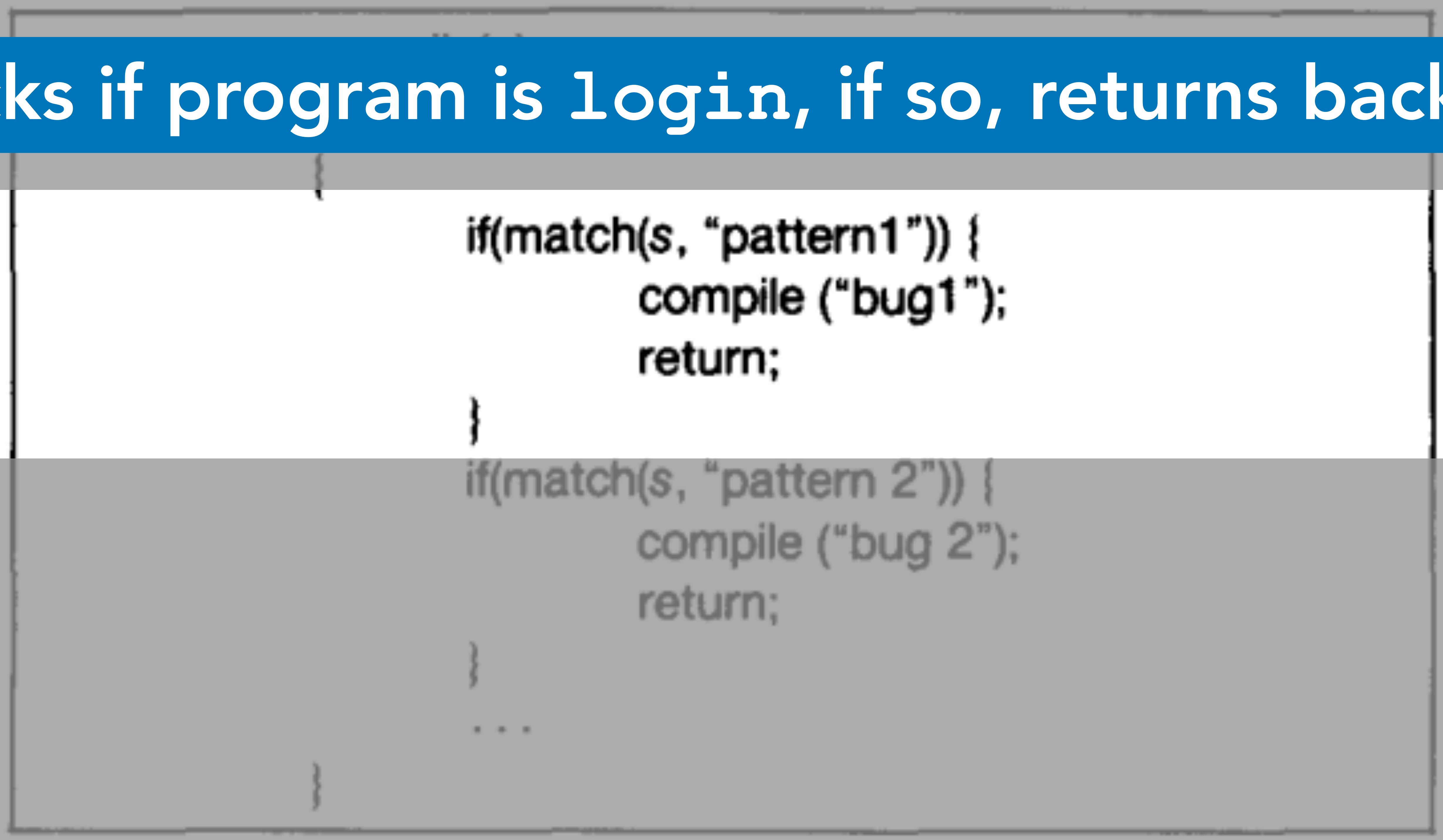
```
compile(s)
char *s;
{
        if(match(s, "pattern1")) {
                compile ("bug1");
                return;
        }
        if(match(s, "pattern 2")) {
                compile ("bug 2");
                return;
        }
        . . .
}
```

In addition to matching on *login*, I could also match the *C compiler itself*, and compile in both trojans

If you own gcc, you can be even more stealthy

Trojan horsing the C compiler to bug code without you knowing it's trojan horsed

some source code in C



```
{
        if(match(s, "pattern1")) {
                compile ("bug1");
                return;
        }
        if(match(s, "pattern 2")) {
                compile ("bug 2");
                return;
        }
        . . .
}
```

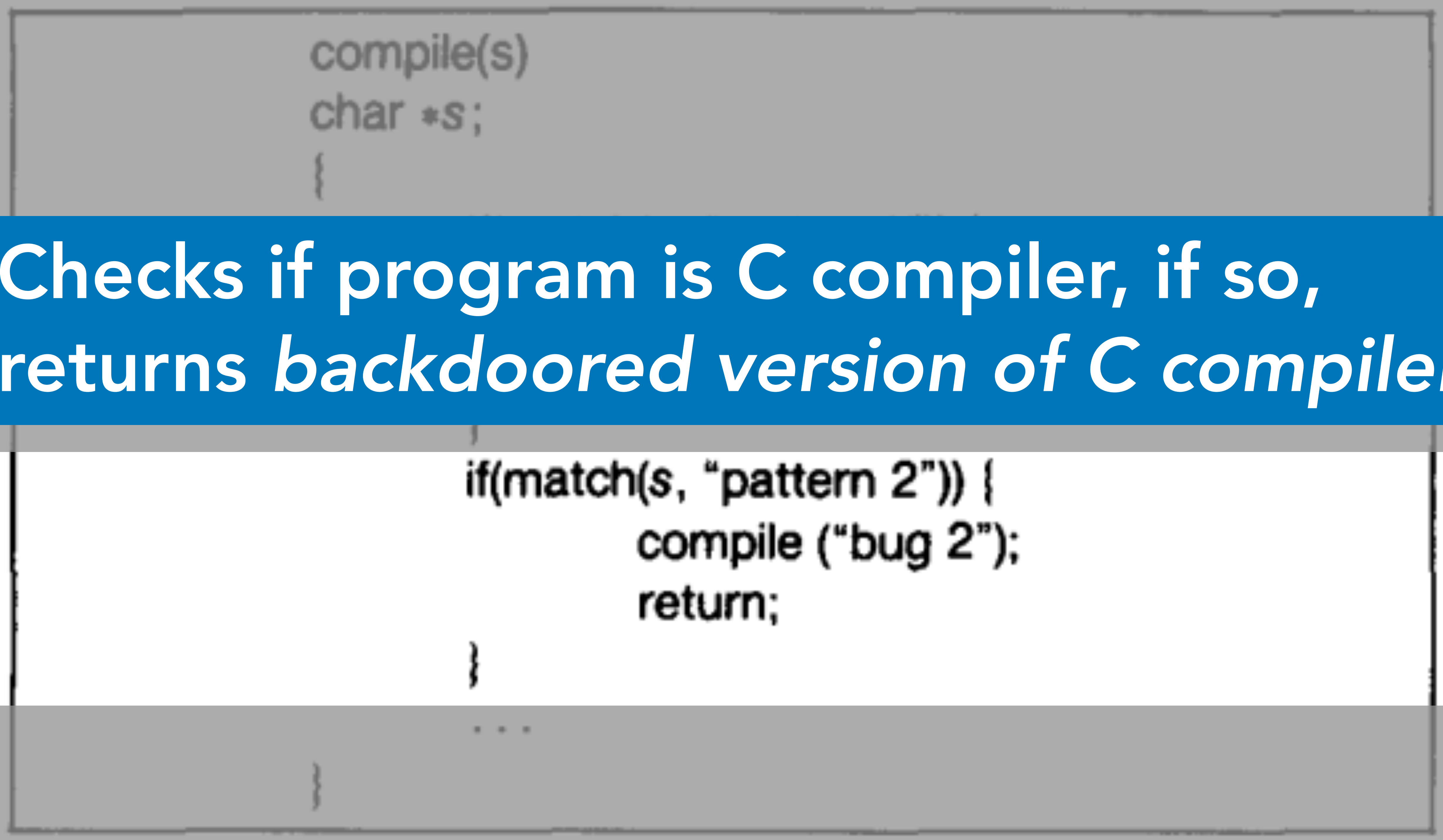
Checks if program is `login`, if so, returns backdoor

In addition to matching *login*, I could also match the *C compiler itself*, and compile in both trojans

If you own gcc, you can be even more stealthy

Trojan horsing the C compiler to bug code without you knowing it's trojan horsed

some source code in C →



```
compile(s)
char *s;
{
        if(match(s, "pattern 2")) {
                compile ("bug 2");
                return;
        }
        . . .
}
```

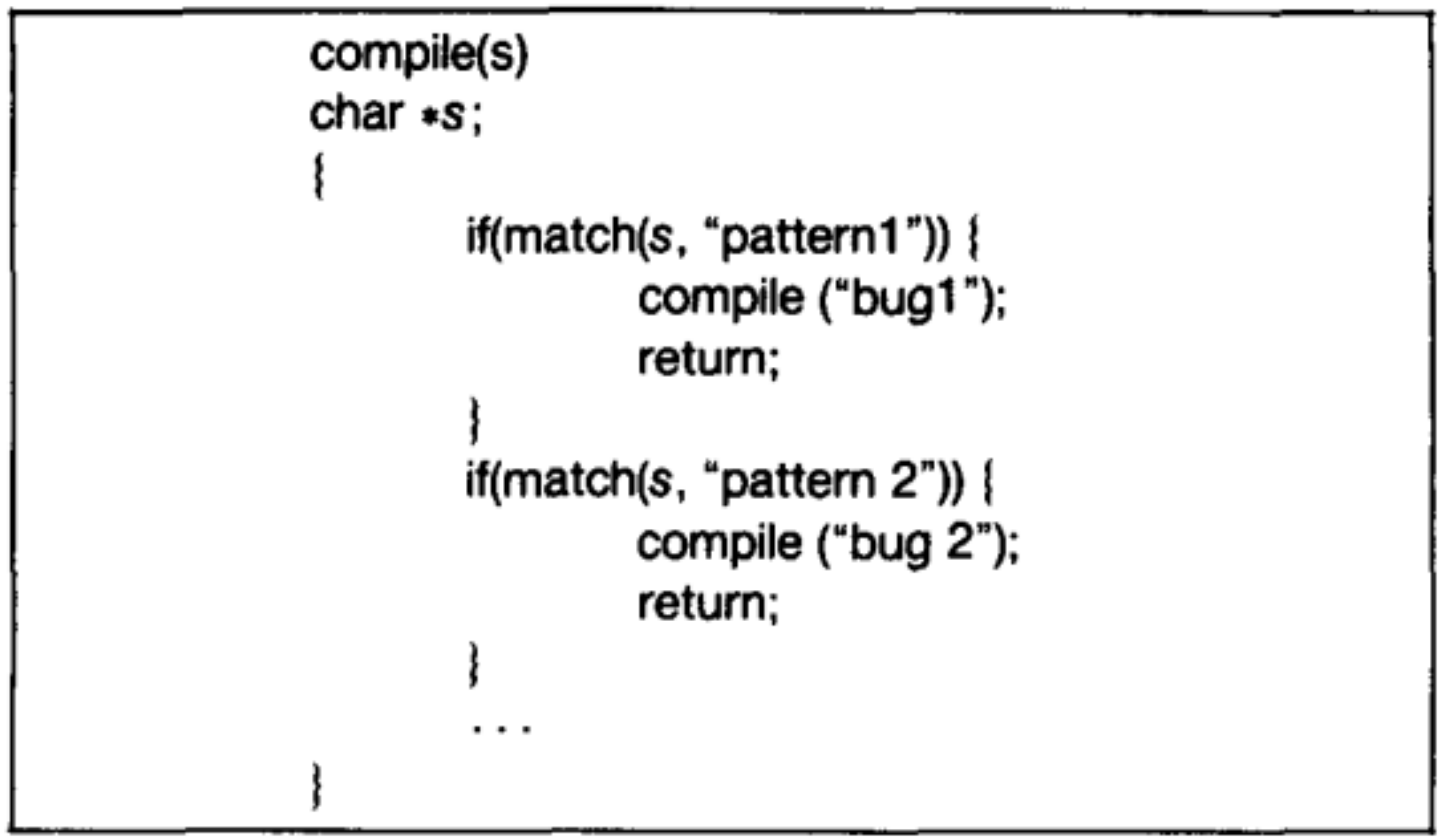
Checks if program is C compiler, if so, returns *backdoored version of C compiler*

In addition to matching *login*, I could also match the *C compiler itself*, and compile in both trojans

If you own gcc, you can be even more stealthy

Trojan horsing the C compiler to bug code without you knowing it's trojan horsed

some source code in C



```
compile(s)
char *s;
{
        if(match(s, "pattern1")) {
                compile ("bug1");
                return;
        }
        if(match(s, "pattern 2")) {
                compile ("bug 2");
                return;
        }
        . . .
}
```

GCC will return an evil version of `login`, *and* an evil version of the C compiler itself

So here's what happens…

- You install the double-trojaned version as the *compiler* on a machine
- You can change the source code of `gcc` back to the non-malicious version
- Anytime someone compiles `login` or even *`gcc`*, they'll still get the backdoor even when it's nowhere to be found in the source code.
  - It's the *exact same* as **11** only being needed one time, after that, you can use '\v'
- **Moral: *You can't trust code that you did not totally create yourself (aka, everything)***
  - Compilers, assemblers, loaders, even hardware microcode
  - Deepak's extension: *You can't have security without* ***trust***

This totally happens in real life

XZ Utils Backdoor – CVE-2024-3094

- In February 2024, a Microsoft employee found a backdoor in XZ, a popular compressor for Unix-like operating systems
  - The backdoor was found in compressed test files that enabled **full remote-code execution on the machine using the compromised xz** via sshd
  - The backdoor was introduced *in the supply chain* – a developer (Jia Tan) who contributed to XZ for *years* introduced the patch in **obfuscated test files**
    - Primary guess right now is nation-state, but we're not sure
- Provocation: Do you trust that every piece of software on your machine has not been tampered with? Why?

My take on computer security

- Security is not a *“thing you do”*, it is a *way of life*
  - We call this the “adversarial mindset:” how do you think like an attacker so you can be ready to 1) find problems or 2) fix them before they happen?
  - Some paranoia is good, maximal paranoia is worthless; we want a sweet spot, called **rational paranoia**
- Security is always in relation to to the threat
  - What does it mean to “be secure?” Against what? With what assumptions?

Course Ethos + Logistics

Course info

- Website: https://cseweb.ucsd.edu/classes/wi26/cse127-a/
    - Slides posted right before class so you can follow along
- Canvas: https://canvas.ucsd.edu/courses/71475
    - Gradescope for grading
    - Piazza for communication (avoid email if you can)
- Office Hours
    - Tuesday 11 – 12pm (or by appointment), CSE 3248

Goals and non-goals of this course

This is a hands-on course

- **Goals**
  - Provide a solid foundation of security concepts, applied to concrete topic areas and hands-on PAs
  - Learn the security mindset
    - How to think like an attacker / security engineer
    - Looking at systems *beyond* intended functionality
  - Understanding how things work, how they break, and how to fix them
- **Non-goals**
  - A deep dive into any one subarea (this is a breadth course)
  - Review of all security mechanisms (we cannot cover everything)

This course, broadly

Topic areas

- We'll cover the following topic areas as best we can in ten weeks:
  - Application security
  - Systems security
  - Web security
  - Network security
  - Cryptography
  - Sociotechnical security

Course materials

- **No textbook** for the class, but here's some I recommend anyway if you're curious…
  - Security Engineering, by Ross Anderson
  - Cryptography Engineering, by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno
- Other readings I'll provide as we go on the course website for additional context

Structure + Grading of this course

- Programming Assignments (40% of your grade)
- Midterm (25% of your grade)
- Final (35% of your grade)

Programming Assignments

- You’ll get hands-on experience deploying *real* attacks and defenses in the following domains:
  - **Application Security** (PA1, PA2)
  - **Web Security** (PA3)
  - **Network Security** (PA4)
  - **Cryptography** (PA5)
- Schedule of release + due dates are already available on website
  - *This class moves fast. There is always an active PA. PA1 is already out!*

Programming Assignments

- Programming assignments MAY be done in teams (if you wish); teams do not need to stay the same between PAs
- PA1 is 4% of your grade (more of a PA0…), the remainder are 9% of your grade each
- Students MUST NOT collaborate with anyone outside of their partner; any unsanctioned group work is academic integrity violation and will be reported

What you will need to know for PAs

- We cover a wide spectrum of tools, languages, and concepts — C, assembly, Python, JavaScript, Networking…
- I'll cover the high levels of these in class, but you need to be prepared to learn stuff on your own
    - E.g., I'm not teaching you x86 32-bit assembly or C or JavaScript

Rules for PAs

- PAs are due at 11:59:59 PT on the due date listed
- You get two late days on PAs (both teammates need to have a late day to use otherwise you don't get one) — these are applied **automatically**
- Regrades are very much the exception, not the norm
  - We reserve the right to completely regrade your assignments :)
- **NO CHEATING**
  - Read and understand the UCSD policy on academic integrity
    - https://academicintegrity.ucsd.edu
  - Not OK to copy, paraphrase, translate, etc. somebody else's work
  - If you not sure if something is cheating, **assume it is until you ask about it**

AI / LLM Policy

- Feel free to use AI / LLMs to help you with your PAs.
    - I have found these tools to be marginally useful in these PAs, outright wrong at other times… use at your own peril
    - **You must attest to your usage of AI in Gradescope. Otherwise, it's an academic integrity violation (yes, we are checking).**
- Note: Exams (which remember are 60% of your grade) will test both lecture material **and** PA material.
    - If you find a way to use AI to do no metacognition, congrats, but you'll probably do quite poorly on exams (and therefore, do poorly in the course)
    - Plus, you're here at UCSD to learn how to do things on your own…
    - My recommendation? Don't use AI at all, but you're all adults, so up to you

A word on ethics…

- In this class, you will learn how to attack the security of various computer programs and systems
- We learn attacks because understanding them is crucial to building **defenses** against those attacks
- You are obligated to use this knowledge *ethically*
  - You **MUST NOT** attack others
    - In addition to being unethical, it's *likely a felony* per the Computer Fraud and Abuse Act (CFAA)
  - If you feel like testing your chops, there are lots of sanctioned ways (see capture the flag competitions)

Exams

- Two exams, a midterm (week 6, in class) and a final (during finals week)
    - Midterm 25%, Final 35%
- Structure of both is very similar
    - Litany of multiple choice questions primarily focused on lecture
    - Broader “PA”-style questions that build on what you did for the PA but with different inputs / outputs
- Final is comprehensive but PA questions will trend towards PA4 and PA5

Attendance, Podcasting, Participation

- Attendance is not mandatory for this course, but I think you'll get the most out of it if you attend in person
- Podcast will be made available, **but only for a week following the lecture.** No exceptions will be made to this rule.
  - Why? I want to support flexibility in learning (I know it's 8am) while also preventing cramming
- I ask a lot of questions in class as a mechanism for learning, so be prepared for that
- If you want to use a laptop during lecture, that's fine, but please sit in the back (it's distracting to other students and often to me)

Course Vibes

Community-centric learning

- The classroom is *community*
  - Get to know one another! The course is only made better when you know each other
  - Come prepared with questions, comments, concerns, thoughts, etc.
  - Discussions should be *respectful*, understanding everyone is here to contribute and to learn
- *This is my first time teaching CSE 127*
  - There will be some bugs. Thanks for your patience as we figure them out.

Immediate Logistics

- **PA1 is released!** It is a basic primer on `gdb` and `x86` — both of which you will need to succeed in PA2.
  - Due on 1/15 at 11:59:59
  - Find a teammate, get to work! Piazza has the teammate search feature open should you need to coordinate.
  - I view this more of a PA0… not too challenging (imo) and should get you used to the way we submit and turn in assignments for the quarter
  - Submission on Gradescope via Canvas
- Complete the #FinAid survey on Canvas (necessary to support those w/ financial aid needs)
  - https://canvas.ucsd.edu/courses/71475/quizzes/238979

Next time…

- We'll talk about risk and threat modeling, and a general approach to analyzing and investigating computer security
  - Worth reading Ken Thompson's reflections on trusting trust (it's just 3-pages) to solidify the trojan we talked about in class today
- Next week, we'll start with Application + Systems Security (buffer overflow attacks, etc.)
- Discussions begin **next Monday** and will be more hands on with the PAs.