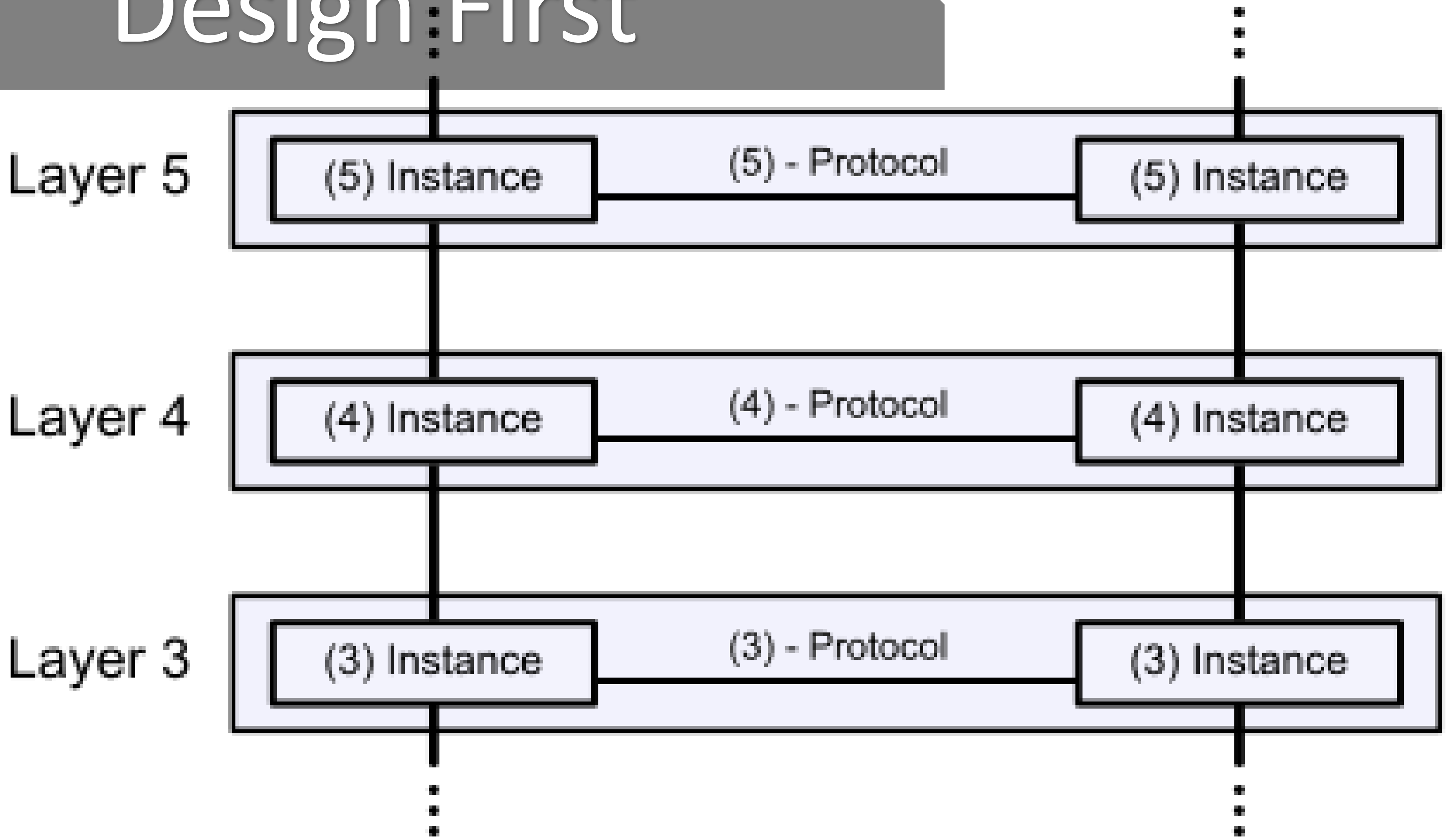
Lenna, Inspired by Liu Chang

Towards A Better Code

VG101 TA Group / Patrick Yao

Yao Yue 2016.18

Design First



Layer 5

(5) Instance

(5) - Protocol

(5) Instance

Layer 4

(4) Instance

(4) - Protocol

(4) Instance

Layer 3

(3) Instance

(3) - Protocol

(3) Instance

B Designs

1 Data

2 Tasks

B

Data

What data?

What key information must be kept?

Storing data?

How to store the data? Logic, easy to access….

B

Tasks

What are the tasks?

List what the problem requires you to do?

What type are the tasks?

For each task, determine it’s nature. IO ? Algorithm? Etc.

B

Data Tasks relation

Input Tasks

Prepares data for further analysis

Q2 read/random

15 points

Algorithm Tasks

Alter the data, compute more data

Q3 Half

15 points

Output Tasks

Use existing data, output the data

Q3 Half, Q4

20 points

B Design function prototype for each task.

Data: 1)Time, t, Length, l

2) Location, array, pos

3) Direction array , { -1, 1}, dir

=> result: matrix, one row per second

Input task

function [pos, t, l] = getInitialCondition(filename)

Output tasks

function outputToFile   (resultPos, resultDir, filename)

function plotAnimation (resultPos, resultDir, filename)

B

Design function prototype for each task.

Data: 1)Time, t, Length, l

2) Location, array, pos

3) Direction array , { -1, 1}, dir

=> result: matrix, one row per second

Algorithm:

```
[resultPos, resultDir] =  algorithm(pos, dir);
```

Algorithm:

```
[posT, dirT] =  algorithm(pos, dir, t);
resultPos (end+1, : ) = posT;
resultDir  (end+1, : ) = dirT;
```

B Do implementation Top-Down

First write main function!

Always begin with the bigger picture!

Input/Output then

Since they're easier

Algorithm should come last:

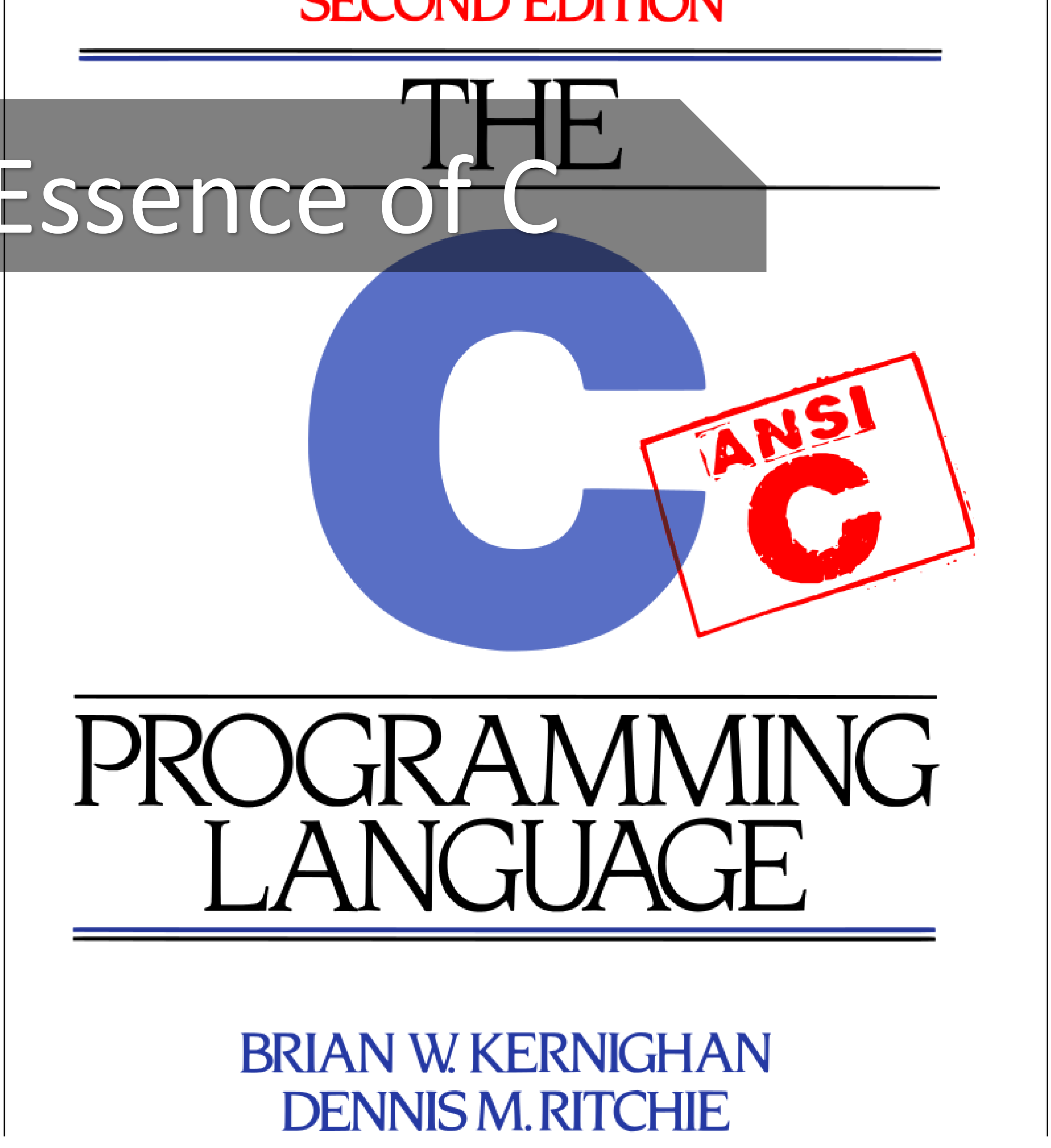
```
[posT, dirT] =  algorithm(pos, dir, t);
resultPos (end+1, : ) = posT;
resultDir  (end+1, : ) = dirT;
```

B Designs

1 Data Motivated

2 Tasks Categorize

Essence of C



SECOND EDITION

THE C PROGRAMMING LANGUAGE

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

A

Where is doc center for C/C++?

Cplusplus.com

www.cplusplus.com

Cppreference.com

http://en.cppreference.com/w/

A

So, “C”, What’s the difference?

Compilation

How the machine understands your code (language)

Memory and Machine Code

How machine executes your code (physical sense)

A

Table Of Contents

1 Function

- Syntax
- Definition
  vs
- Declaration

2 Preprocess

- **#include**
- **#define**
- Conditional Compilation

3 Datatype & Variables

- Variables
- Types
- Datatype in expression
- Datatype and memory.

A

Compiling a C Code:

Compile only the source files

i.e. “.c” files all and only. Each file from top to bottom

- Compile: `> gcc -o a s1.c s2.c s3.c …`

Each source file compiles independently.

Only after all files are compiled, put them together.

A The `main()` function

`int main();`

In homework `int main(int argc, char* argv[]);`

1 ALWAYS THERE

- Some context we call this fun. entry point.
- Your program begins here.
- Called by OS

2 ALWAYS `int`

- It's good practice!

3 ALWAYS `return`

- Return this function means end of programs.
- OK `return 0;`
- Error `return -1;`
- `exit(0)`

A Declaration vs Definition

Declaration:
An announcement for compiler

Definition:
Actual code, actual implementation

A Declaration

```
returnType functionName(type arg1, . . . );
```

You can omit argument name. `int add(int, int);`

- Function declaration should be put outside any function.
- We also call this function prototype. Prototype, some thing remains to improve, something just has a shape.
- In every ".c" file, a function must be DECLARED before use.
- It's there to let the compiler know how to check if you used the function with correct arguments.
- It's OK to declare a function multiple times. As far as they are the same.

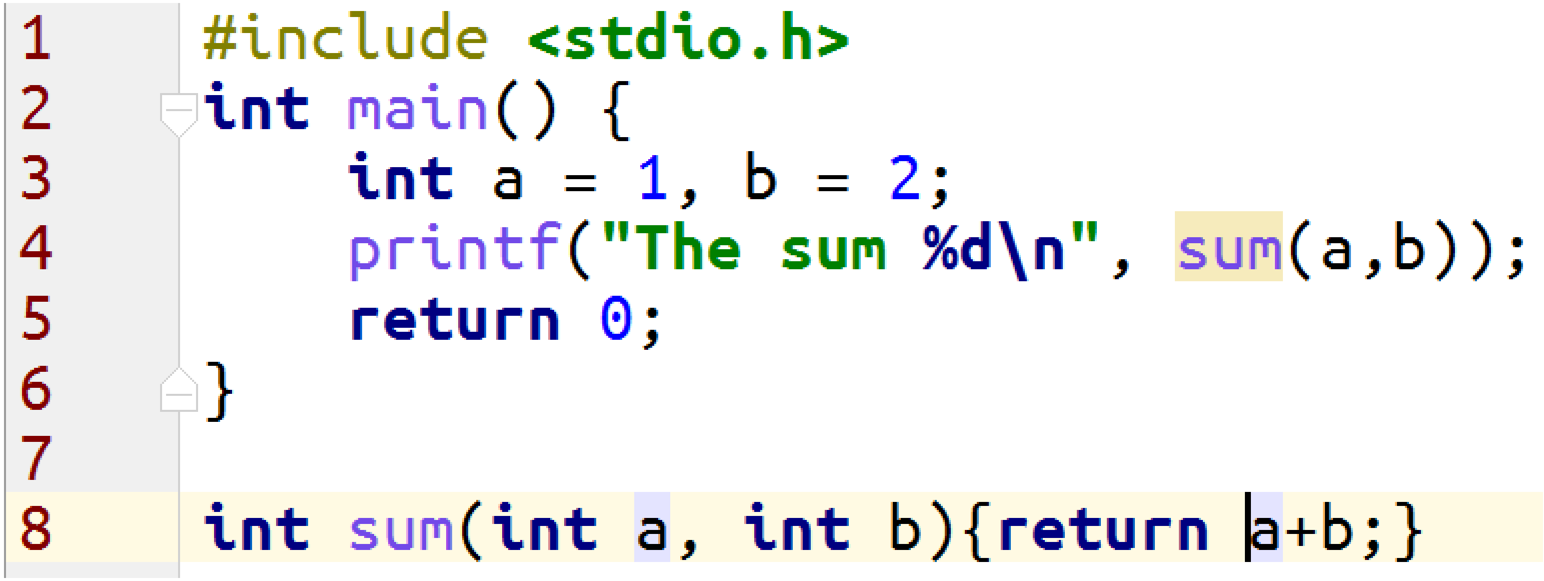
A Definition

```
returnType functionName(type arg1, . . . ){
    // HERE IS SOME CODE
    // Definition contains the actual code
    // for the function. Don’t put it locally.
    // there must ONLY exist ONE definition
    // across all files. failing to do so
    // results in compiler error
    return VALUE_OF_CORRECT_TYPE;
}
```

- A definition is automatically a declaration

A

What’ s wrong?



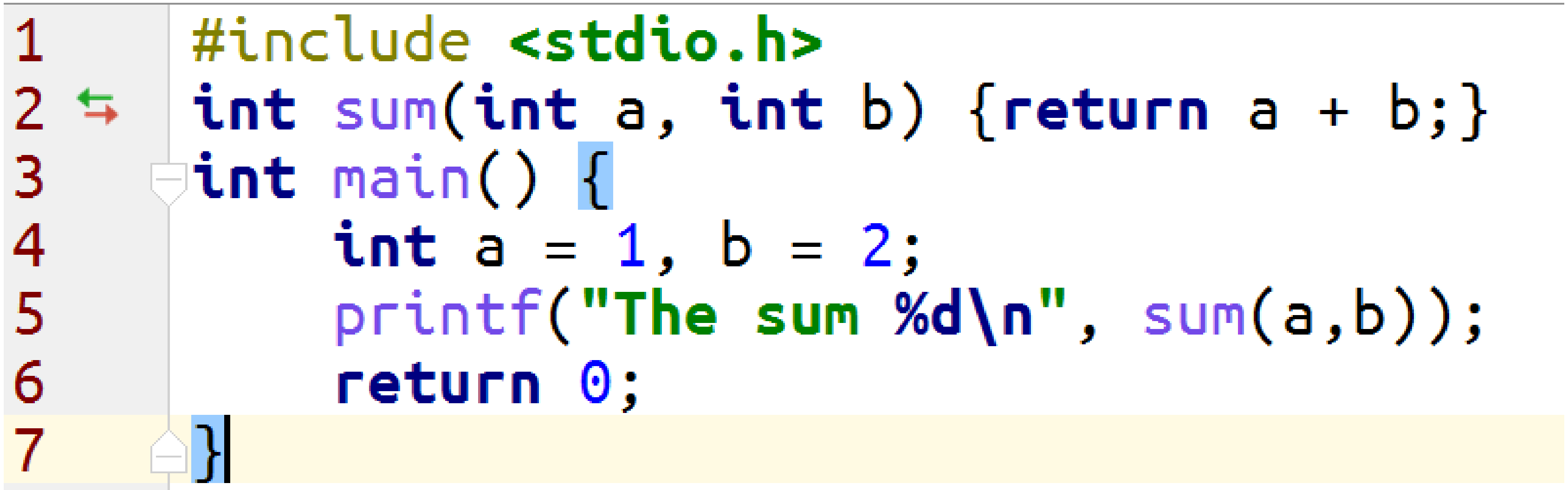
```c
#include <stdio.h>
int main() {
    int a = 1, b = 2;
    printf("The sum %d\n", sum(a,b));
    return 0;
}

int sum(int a, int b){return a+b;}
```

A

Why this works?

main.c

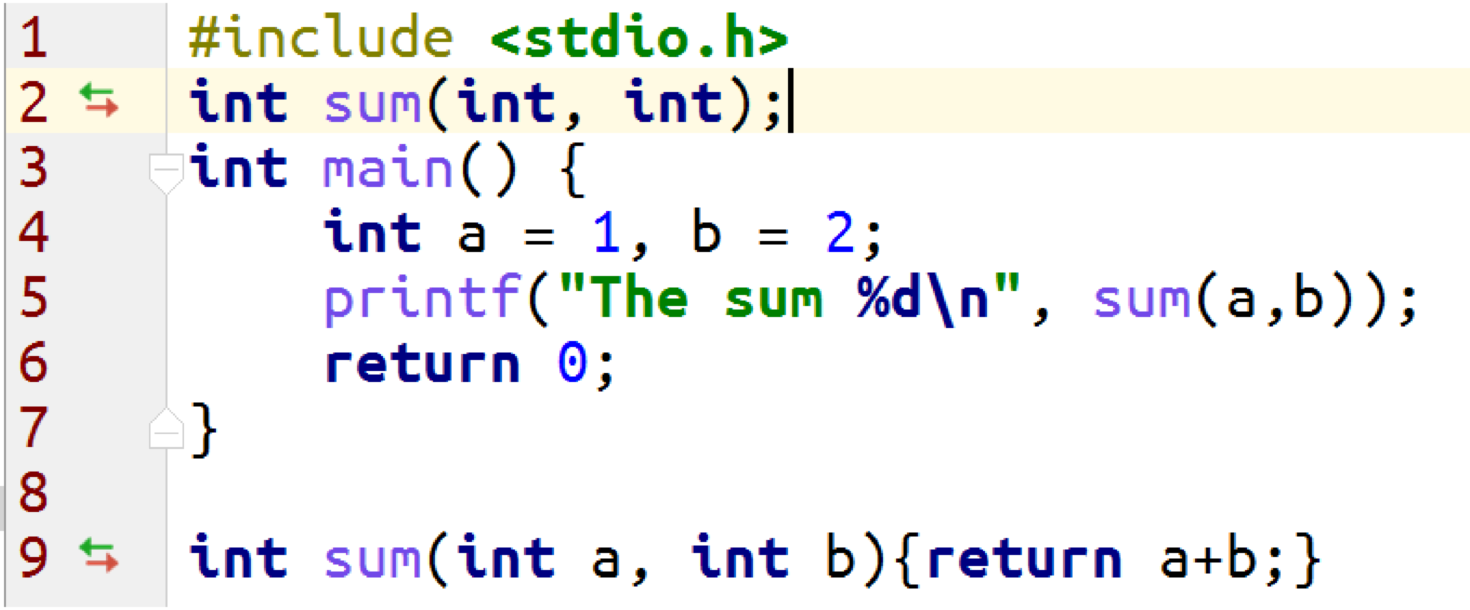


```c
#include <stdio.h>
int sum(int a, int b) {return a + b;}
int main() {
    int a = 1, b = 2;
    printf("The sum %d\n", sum(a,b));
    return 0;
}
```

A

What’ s wrong?

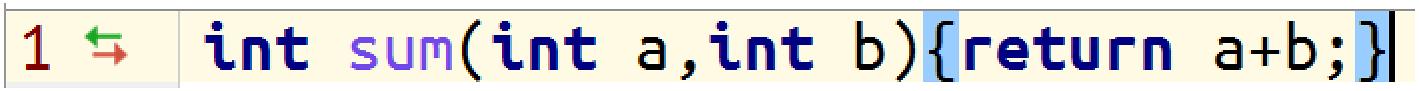
main.c



```c
#include <stdio.h>
int sum(int, int);
int main() {
    int a = 1, b = 2;
    printf("The sum %d\n", sum(a,b));
    return 0;
}

int sum(int a, int b){return a+b;}
```

sum.c



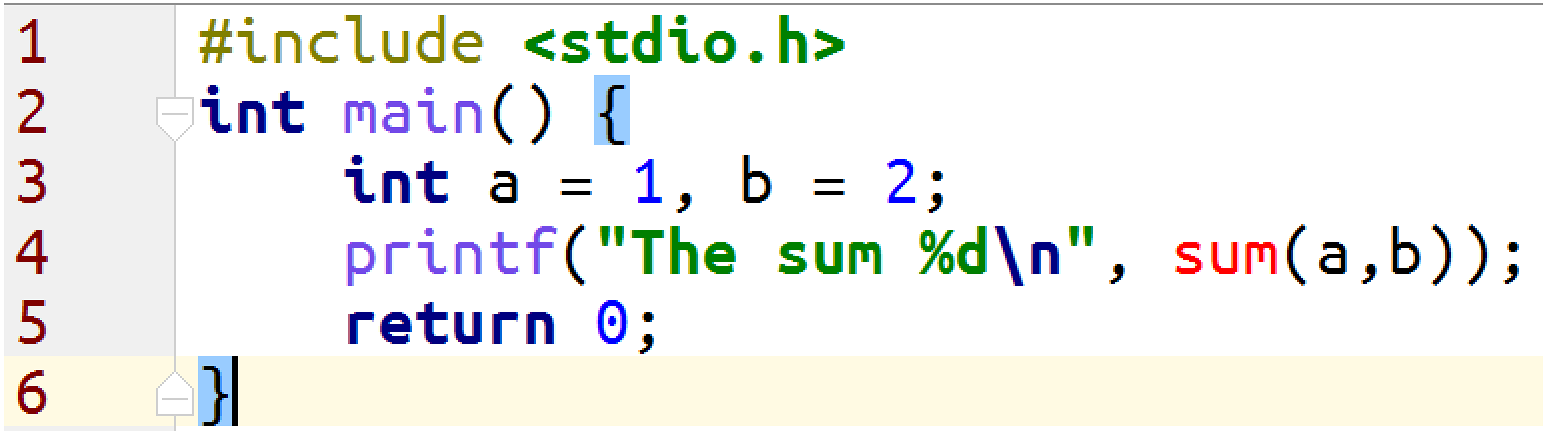
```c
int sum(int a,int b){return a+b;}
```

- Compile: `> gcc main.c sum.c -o a`

A

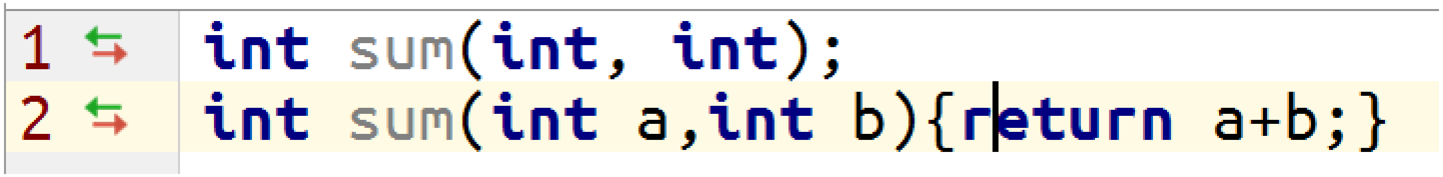
What’ s wrong?

main.c



```
#include <stdio.h>
int main() {
    int a = 1, b = 2;
    printf("The sum %d\n", sum(a,b));
    return 0;
}
```

sum.c

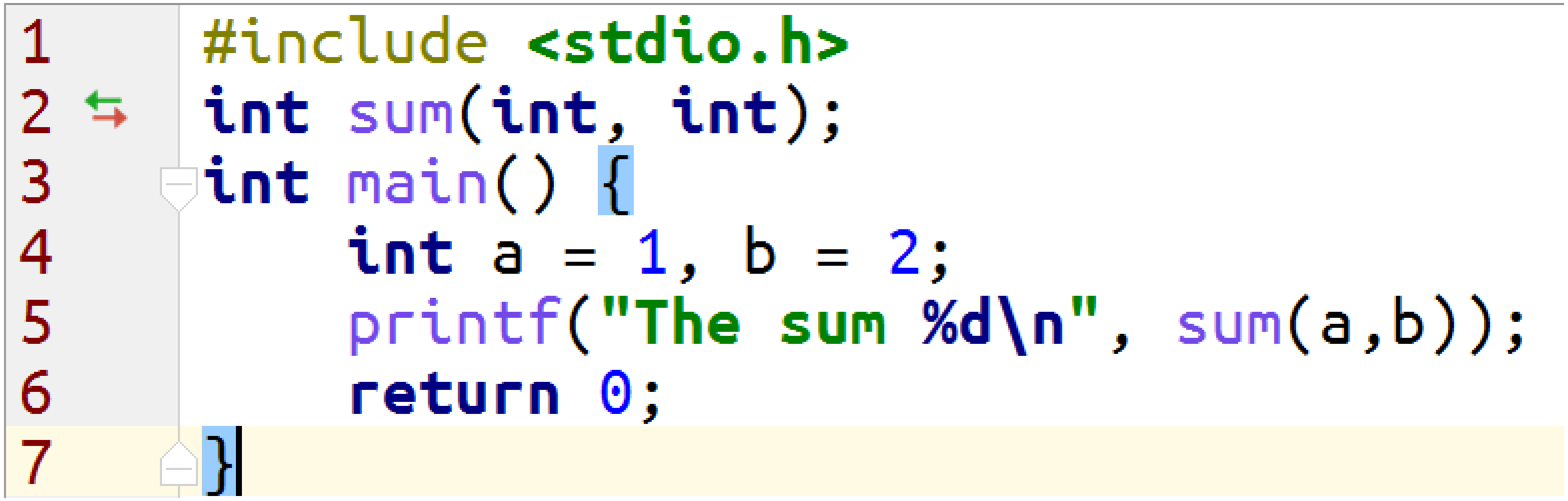


```
int sum(int, int);
int sum(int a,int b){return a+b;}
```

- Compile: `> gcc main.c sum.c -o a`

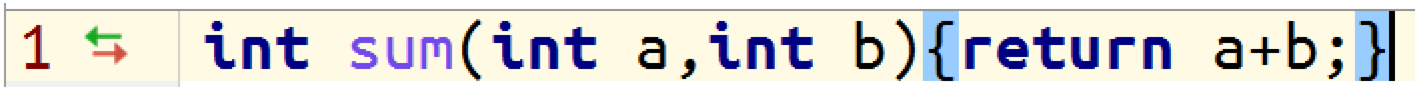
A What’ s wrong?

main.c



```
#include <stdio.h>
int sum(int, int);
int main() {
    int a = 1, b = 2;
    printf("The sum %d\n", sum(a,b));
    return 0;
}
```

sum.c



```
int sum(int a,int b){return a+b;}
```

- Compile: `> gcc main.c -o a`

A

Preprocessing

Preprocessing Is TEXT MANIPULATION

Before compilation, No understanding of code.

- The command begin with # tags are processed one line after another.
- These commands are called preprocess commands.
- No, # tags in comments will not get processed.

A

#include

#include Is copying code from another file

Best for include library.

- By convention, all **#include** commands comes in the top most lines of code.
- Use **#include <stdio.h>** to include
  STanDard Input Output library
- Use **#include <math.h>** to include math library
- Use **#include <stdlib.h>** to include C standard main library. Future functions like **malloc()** might need it.
- Use **#include <string.h>** to include string library

A

#include

#include Is copying code from another file

Used for organizing your own code

- #include "header.h" looks for the file in the same directory, then replace this command with all contents of header.h
- Which means a header file can include another header. Be careful this can potentially become an "loop".
- Best for cases where you have a function that needs to be used by multiple files. You put its declaration in the header file, then include it in every source file (.c) that need it.
- Sometimes used with #define if you need a constant everywhere.

A

`#define`

`#define` performs TEXTUAL replacement

Call these things MACROs. (NOT Macross…)

- `#define A B` replaces all occurrences of A with B
- B can by empty, this case replace A with “nothing”, i.e. deleting all occurrences of A.
- It only replace complete words. `#define this that` will replace things like `this();` or `this = 1;` or `fun(this);` but not `this_var` or `thisFunction();`
- Use them to define constants! Mind brackets!
- MACROs are always in case letters!!

A

#define

`#define` performs TEXTUAL replacement

Call these things MACROs. (NOT Macross…)

- `#define SQR(x) (x * x)` acts like a function. It replace all x in the right hand side with left `x`.
- Use to define simple functions
- For instance `SQR(a * b)` expands into `(a * b * a * b)`
- For instance `SQR(x + 1)` expands into `(x + 1 * x + 1)`
- See any thing wrong?
- Mind the brackets!!

Experienced programmers still make this mistake.

A

#ifdef , #ifndef

#ifdef keep part of the code by condition

Yes, you do have #if #elseif #endif

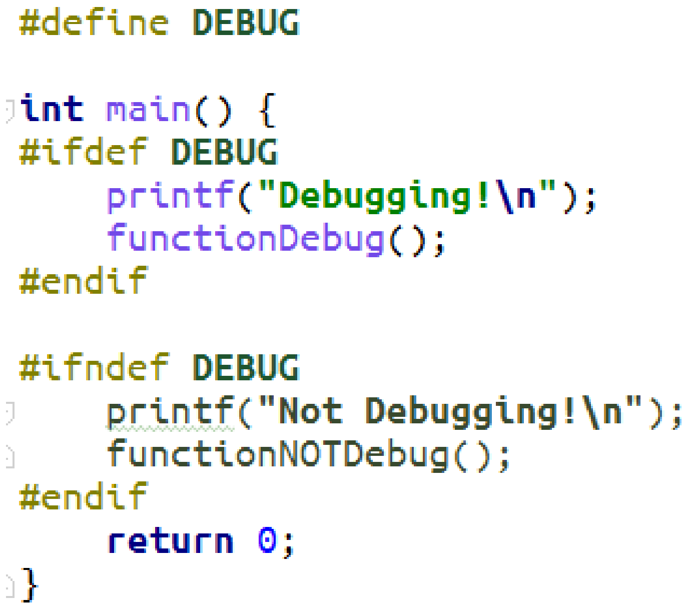
- We call the behavior of these commands conditional compilation.
- The condition is whether a MACRO has been defined.
- Remember, preprocessing goes from top to bottom.
- You best friend for debugging code!

A

#ifdef , #ifndef

Compare the following code

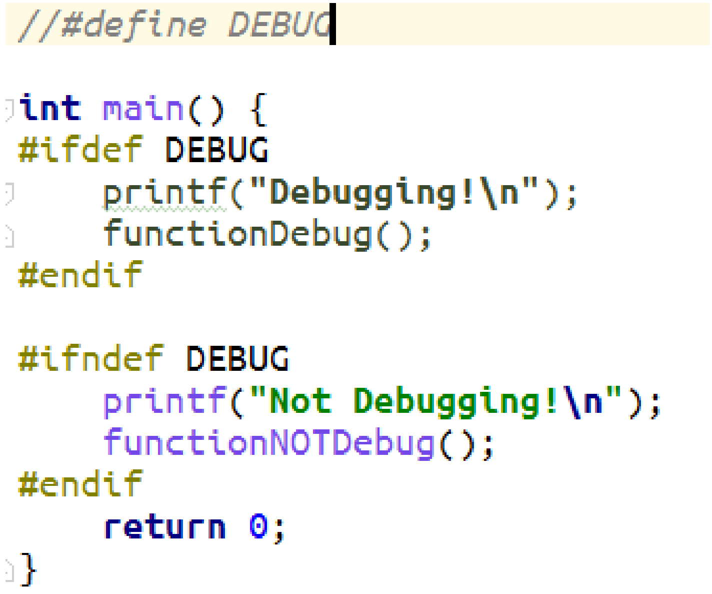
The colorful code are code gets compile,
Dark ones gets deleted before compiling.



```
#define DEBUG

int main() {
#ifdef DEBUG
    printf("Debugging!\n");
    functionDebug();
#endif

#ifndef DEBUG
    printf("Not Debugging!\n");
    functionNOTDebug();
#endif
    return 0;
}
```



```
//#define DEBUG

int main() {
#ifdef DEBUG
    printf("Debugging!\n");
    functionDebug();
#endif

#ifndef DEBUG
    printf("Not Debugging!\n");
    functionNOTDebug();
#endif
    return 0;
}
```

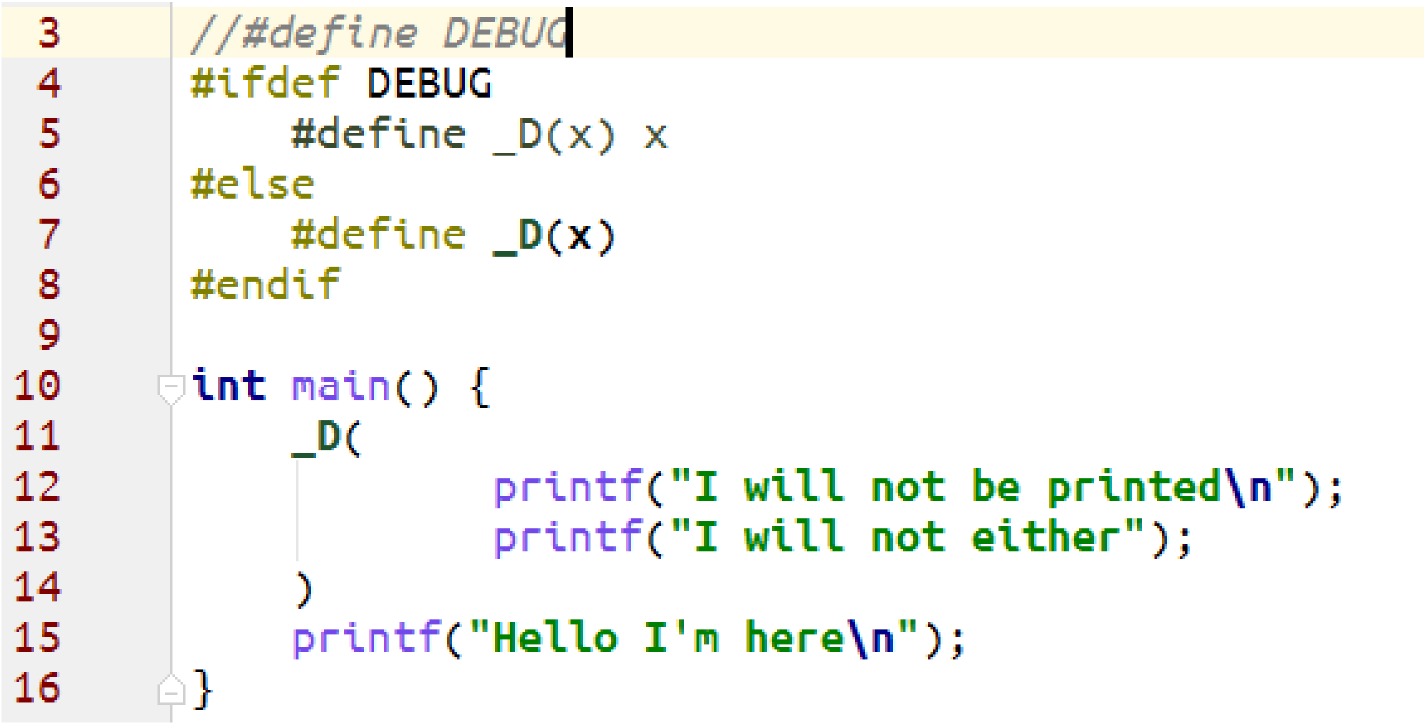
Yep another reason I love Clion.

A

Sample code

Simple Debug helper

Define **DEBUG** Macro to enable/disable debug outputs



```c
//#define DEBUG
#ifdef DEBUG
    #define _D(x) x
#else
    #define _D(x)
#endif

int main() {
    _D(
            printf("I will not be printed\n");
            printf("I will not either");
    )
    printf("Hello I'm here\n");
}
```

A Sample code

Header Guards

Define `NAME_H` Macro make sure file included only once

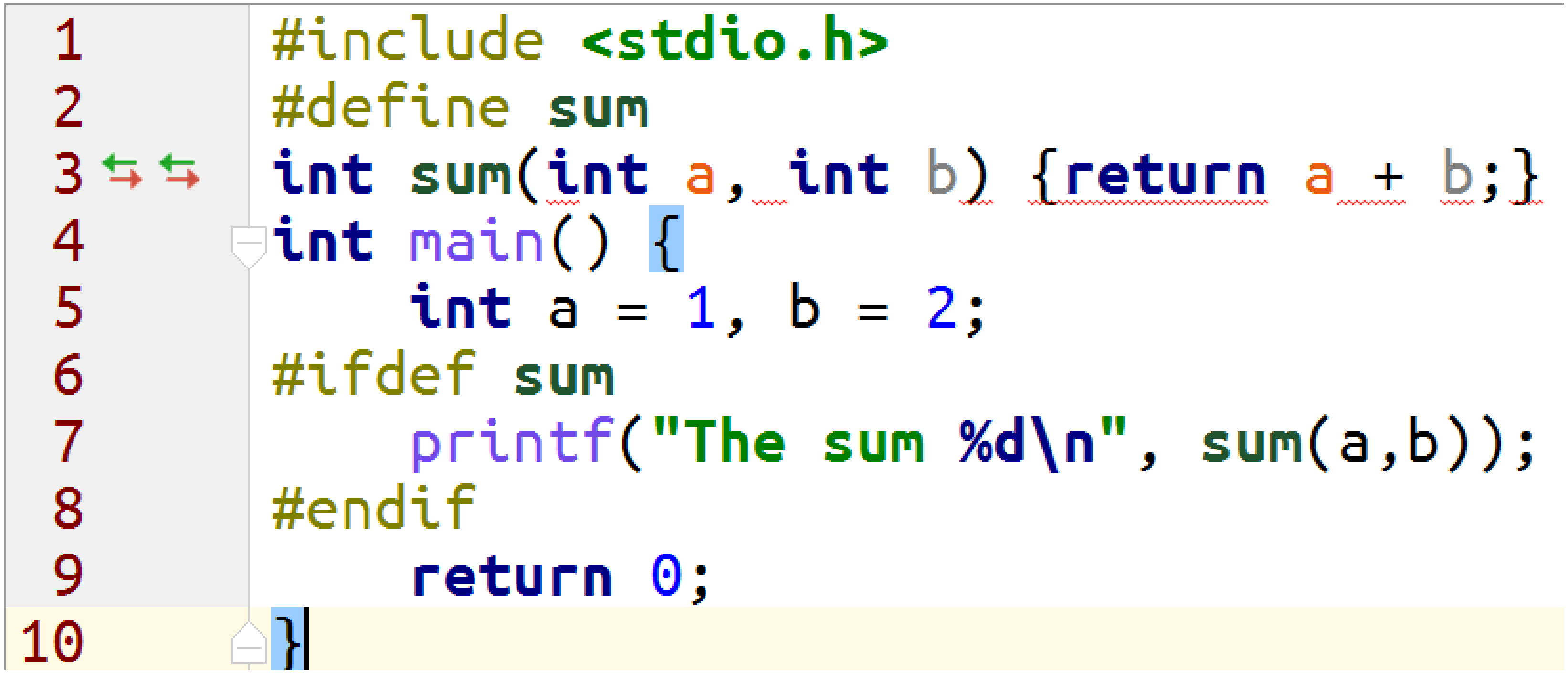
Example is the template files!

How you separate different functions in to multiple files
And make them available to the `main` function

A

What’ s wrong?

main.c

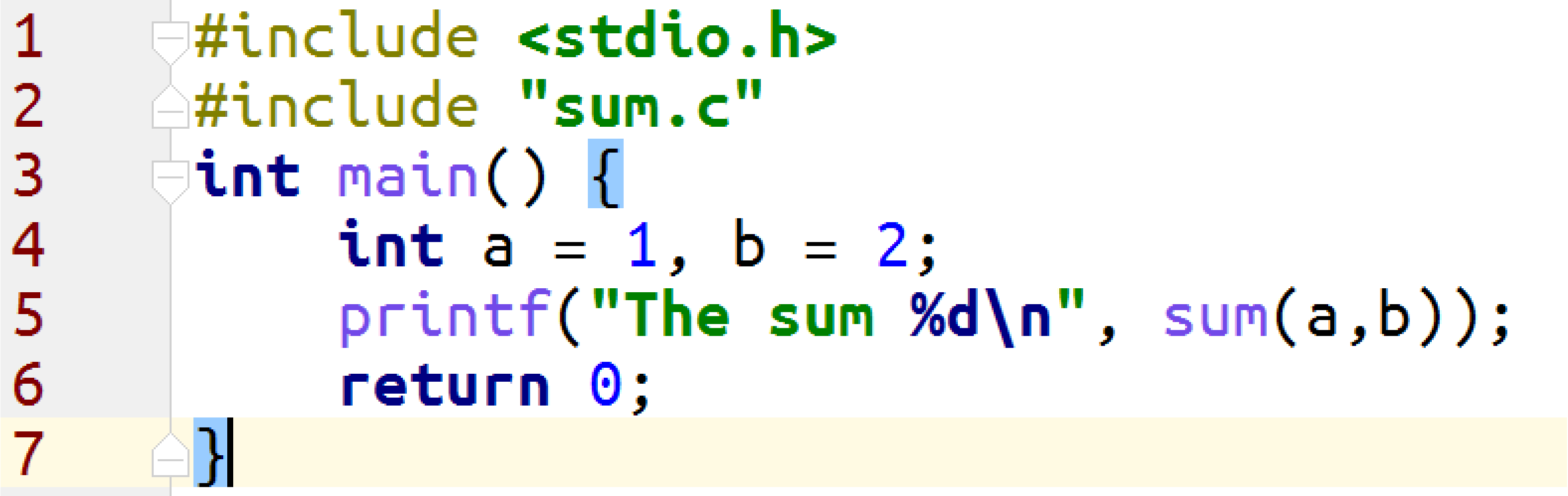


```c
#include <stdio.h>
#define sum
int sum(int a, int b) {return a + b;}
int main() {
    int a = 1, b = 2;
#ifdef sum
    printf("The sum %d\n", sum(a,b));
#endif
    return 0;
}
```

A

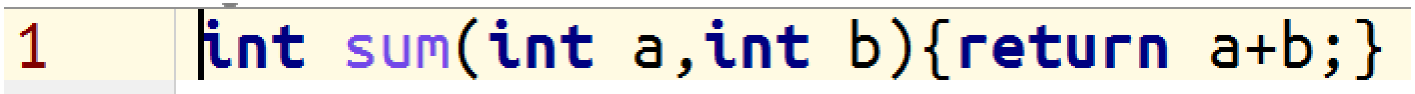
What’ s wrong?

main.c



```c
#include <stdio.h>
#include "sum.c"
int main() {
	int a = 1, b = 2;
	printf("The sum %d\n", sum(a,b));
	return 0;
}
```

sum.c



```c
int sum(int a,int b){return a+b;}
```

- Compile: `> gcc main.c sum.c -o a`

A

Datatype and Variables

Variable: Named place in memory

Type, size, range and scope (and where)

A

Variable Syntax

```
type varName;
```

Declaration: state the type, value is Undefined

```
type varName = initValue;
```

Definition: give an initial value. USE THIS ONE

- Naming rules. 1) Numbers can't lead 2) No keyword 3) Suggested: smallCaseCamel
- Declare before use. One definition only.

A

Size of variable

Variables occupies some memory

Different type occupies different amount of memory

```
sizeof([var|type|arrayName]);
```

Checks how much memory a variable (a variable of specified type will take up).

- sizeof() is not a function. Value determined **during compiling**

A

Scope of variable

Global Var, accessible to all functions

DISALLOWED, defined outside function

Local Var, accessible within a function

Define in function. Vanish after the function exits

Static Var, accessible within a function

It keeps it’s value after the function exits

A Variable

Type: What does it store?

Range: How large can it store?

Size: How much memory it takes?

Scope: Who can use it?

A

YOU SHOULD KNOW:

EVERY VALUE IN C HAS A TYPE

No mater it’s a variable or “value of an expression”

A Type, that stores **DIFFERENT** data

`int, char`

Integers (And their variants):

`float, double`

Decimal numbers, (float points):

`char*` (to be continued….)

A string.

(In the future you see it falls into a bigger category, i.e. “pointers”)

A

WARNING

STRING BEHAVE COMPLETELY DIFFERENT

Although it does have the 4 properties of variables

A Type, stores same kind of data

`unsigned int, unsigned char`

Sign variants. Whether it could represent negative numbers,
Changes range, doesn’t changes size.

`long int, long long int, short int.`

Size Variants. Changes size and range.
No size variant for **`char`**

A Variable

Type: What does it store?

Range: How large can it store?

Size: How much memory it takes?

Scope: Who can use it?

A

Answer the questions

- Is it a double of float ?
  - YES: `sizeof(double)==8` , `sizeof(float)==4`. Range DC
- No, Then it is of integer type. Is it a `char`?
  - Yes: `sizeof(char)==1`, 8 bits, thus
    - Unsigned: 0 to 255. (0 to 2^8-1)
    - Signed : One bit used for sign. -128 to 127. [-2^7,2^7-1]
    - `unsigned char` is essentially a `byte`
- No, then it is `int` or its variant.
  - Use `sizeof(type)` to find its size. Let's say 4 bytes, 32 bits.
  - Unsigned, 0 to 2^32 – 1
  - Signed, -2^31 to 2^31 -1
  - Think why in HW3 you have to use `uint64`

A

char and encoding

How do you store a “character”?

After all data in a computer is numbers.

A

char and encoding

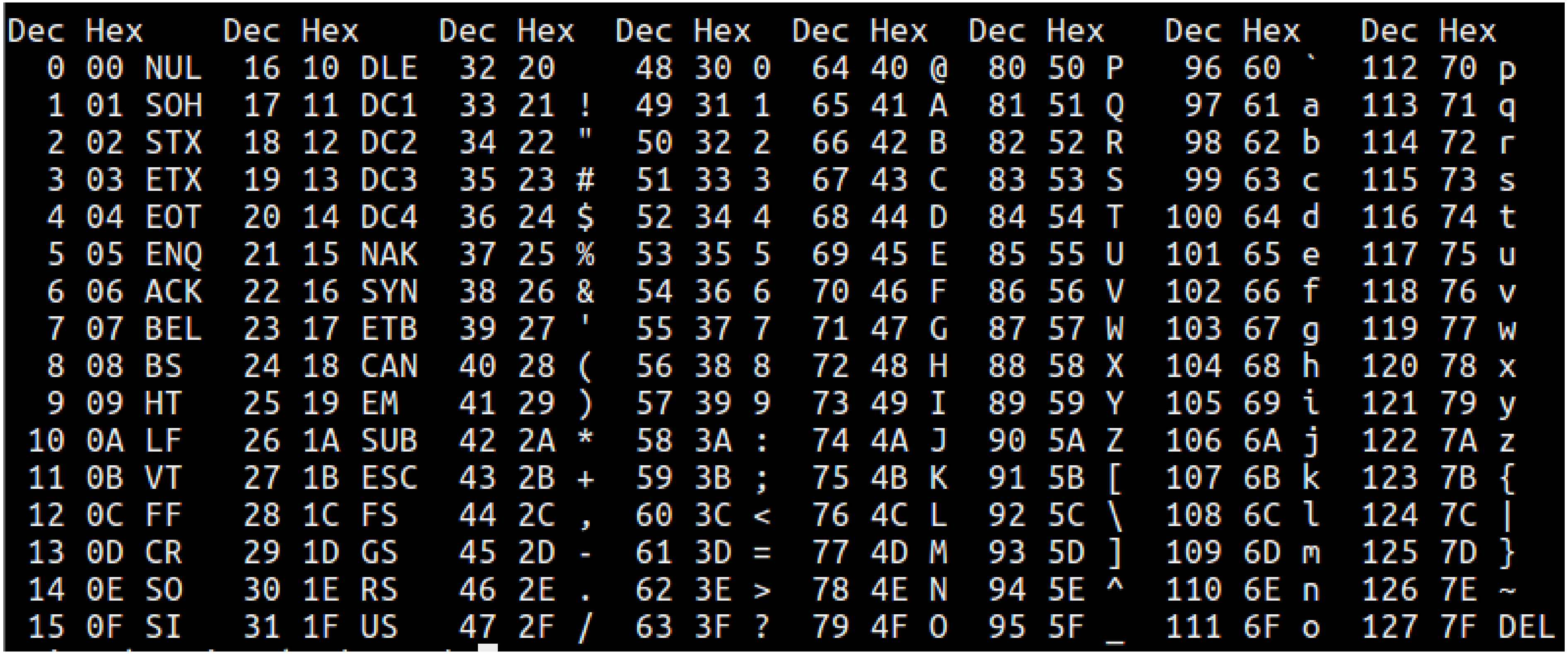
How do you store a “character”?

After all data in a computer is numbers.

Encoding: the one to one relation Between numbers and characters

A

the ASCII code



```
Dec Hex    Dec Hex    Dec Hex  Dec Hex  Dec Hex  Dec Hex   Dec Hex   Dec Hex
  0 00 NUL  16 10 DLE  32 20    48 30 0  64 40 @  80 50 P   96 60 `  112 70 p
  1 01 SOH  17 11 DC1  33 21 !  49 31 1  65 41 A  81 51 Q   97 61 a  113 71 q
  2 02 STX  18 12 DC2  34 22 "  50 32 2  66 42 B  82 52 R   98 62 b  114 72 r
  3 03 ETX  19 13 DC3  35 23 #  51 33 3  67 43 C  83 53 S   99 63 c  115 73 s
  4 04 EOT  20 14 DC4  36 24 $  52 34 4  68 44 D  84 54 T  100 64 d  116 74 t
  5 05 ENQ  21 15 NAK  37 25 %  53 35 5  69 45 E  85 55 U  101 65 e  117 75 u
  6 06 ACK  22 16 SYN  38 26 &  54 36 6  70 46 F  86 56 V  102 66 f  118 76 v
  7 07 BEL  23 17 ETB  39 27 '  55 37 7  71 47 G  87 57 W  103 67 g  119 77 w
  8 08 BS   24 18 CAN  40 28 (  56 38 8  72 48 H  88 58 X  104 68 h  120 78 x
  9 09 HT   25 19 EM   41 29 )  57 39 9  73 49 I  89 59 Y  105 69 i  121 79 y
 10 0A LF   26 1A SUB  42 2A *  58 3A :  74 4A J  90 5A Z  106 6A j  122 7A z
 11 0B VT   27 1B ESC  43 2B +  59 3B ;  75 4B K  91 5B [  107 6B k  123 7B {
 12 0C FF   28 1C FS   44 2C ,  60 3C <  76 4C L  92 5C \  108 6C l  124 7C |
 13 0D CR   29 1D GS   45 2D -  61 3D =  77 4D M  93 5D ]  109 6D m  125 7D }
 14 0E SO   30 1E RS   46 2E .  62 3E >  78 4E N  94 5E ^  110 6E n  126 7E ~
 15 0F SI   31 1F US   47 2F /  63 3F ?  79 4F O  95 5F _  111 6F o  127 7F DEL
```

A ASCII code

All value between 0 and 127

Just enough to be put in to a **`char`**

Escape Characters : `‘\n’, ‘\t’`

Represents a new line / a Tab

A Storing a character

```
char c = ‘a’;
```

1 Before compiled :
Find ‘a’ get replaced by the ASCII code

2 Equivalently: `char c = 97;`

A

A few notes about **char** and string

Everything in a single quote become an integer.

And strings are NOT integers

String is stored as an array of char

Yes, strings in memory essentially a series integers.

DO NOT MISSUSE QUOTATIONS

A

Data type and IO

```
printf((char*)format,var,…);
```

Similar to Matlab (“fprintf”)

```
scanf((char*)format,&var,…);
```

There is a “&” in front of the var.
NO “&” if “var” is a string.

A

Data type and IO

%d, %u ,(%ld, %lld) integers

If you want to output an integer

%c, characters; %s, strings

If you want to output an integer as a character.
Or in input a character (store its ASCII code in an integer)

%f, %lf, decimal float points

%f for **float**, and **%lf** for **double**

A

Data type and IO

Output formats determine how to “interpret” your input to the user

Input formats determine how to “interpret” user input to your program

A

Type Casting: 2 understanding

Force computer change type of value

The language perspective, you need Memorization

Re-interpreting same data in memory

This is a memory perspective.

A

Type Casting: Syntax

```
(newType)valueOfOldType
```

Casting has very high priority. Still use BRACKETS!

A

Type Casting: Safe and Unsafe

Casting across basic types are usually unsafe

In memory, these data are fundamentally different.

Casting between unsigned and signed may be unsafe

Take care of negative numbers! Then you are good.

Casting from longer to shorter integers may be unsafe

Think about range!

A

Type Casting: `float,double` to `int`

This is special case. If reverse, its always safe.

C Standard leave the former undefined.

In most cases, casting from `float` to `int` works

It simply drops all decimal part.

Try do this : `int n = (int)fix(varFloat);`

It's best practice, look up `fix()` in cplusplus.com !

A

YOU SHOULD KNOW:

EVERY VALUE IN C HAS A TYPE

No mater it’s a variable or “value of an expression”

A Datatype of Constants.

By default, “`1`” as considered “`(int)1`”

You can change it by saying “`1ul`” as “`(unsigned long int)1`”

By default, “`1.0`” is considered “`(double)1.0`”

“`1.0f`” as “`(float)1.0`”. Mind precision in assignment

For special needs, do not rely on default behavior

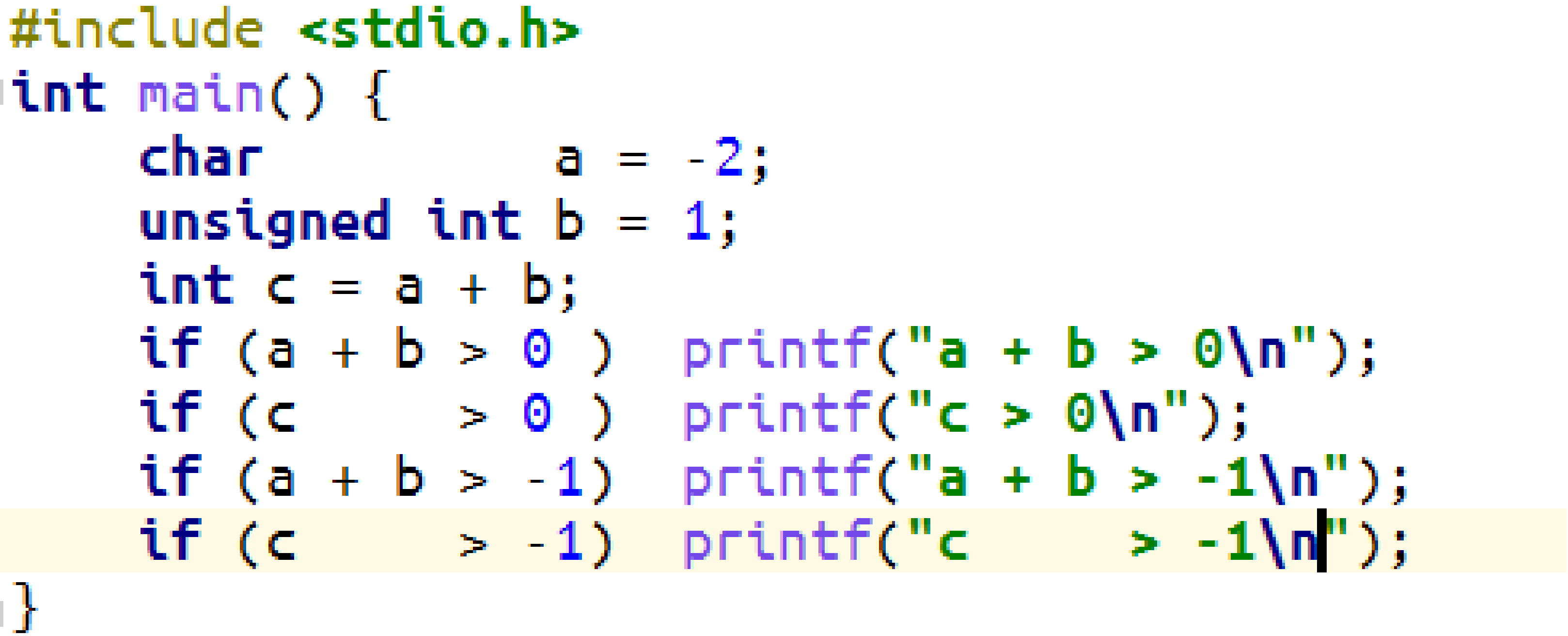
Look it up in online reference. We won’t test it.

A

Type Casting in expressions

Consider the following code

Types don’t match! Will the code compile?



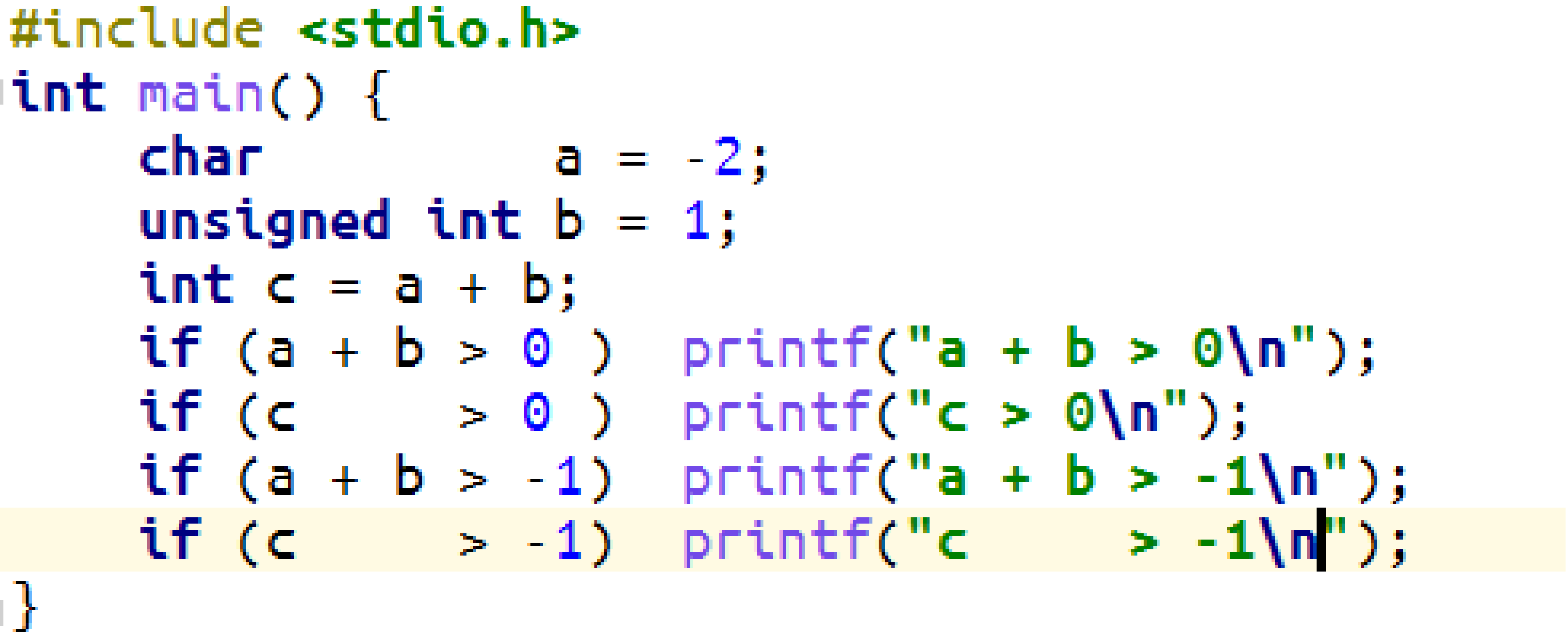
```
#include <stdio.h>
int main() {
    char         a = -2;
    unsigned int b = 1;
    int c = a + b;
    if (a + b > 0 )  printf("a + b > 0\n");
    if (c     > 0 )  printf("c > 0\n");
    if (a + b > -1)  printf("a + b > -1\n");
    if (c     > -1)  printf("c     > -1\n");
}
```

A

Implicit Casting

The code will compile and run.

The result is “a + b > 0” only. Not making sense.



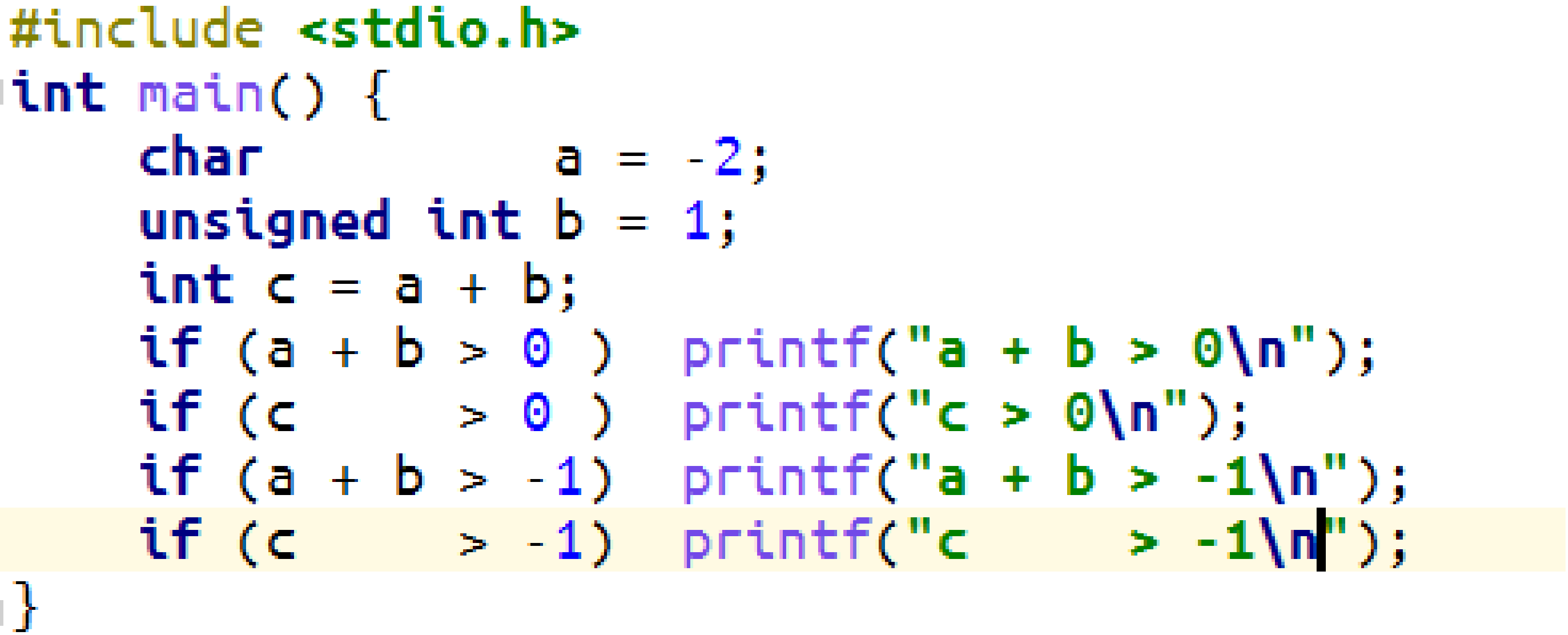
```
#include <stdio.h>
int main() {
    char         a = -2;
    unsigned int b = 1;
    int c = a + b;
    if (a + b > 0 )  printf("a + b > 0\n");
    if (c     > 0 )  printf("c > 0\n");
    if (a + b > -1)  printf("a + b > -1\n");
    if (c     > -1)  printf("c     > -1\n");
}
```

A

Implicit Casting

Reason: Implicit casting happens

Casting is performed without you telling it.



```
#include <stdio.h>
int main() {
    char         a = -2;
    unsigned int b = 1;
    int c = a + b;
    if (a + b > 0 )  printf("a + b > 0\n");
    if (c     > 0 )  printf("c > 0\n");
    if (a + b > -1)  printf("a + b > -1\n");
    if (c     > -1)  printf("c     > -1\n");
}
```

A

Implicit Casting:

Implicit Casting

Casting performed by the compiler to automatically deal with not matching data type in expressions

Usually happens in operations (+, -, * , /) , assignments (=) and logical operations (<,<=, > …)

Note we won’t test you for this. But really many bugs exist because of it. You may pose your self with such bug without even knowing it!

A

Rules of implicit casting

If size don’t match, expand the smaller one

Expanding size comes first

If both signed and unsgn exists, cast to unsigned.

Take care of negative numbers! Then you are good.

If both integer and float exists, cast to float.

Be careful, integers may lose precision.

Do casting pair by pair, according to order of operations.

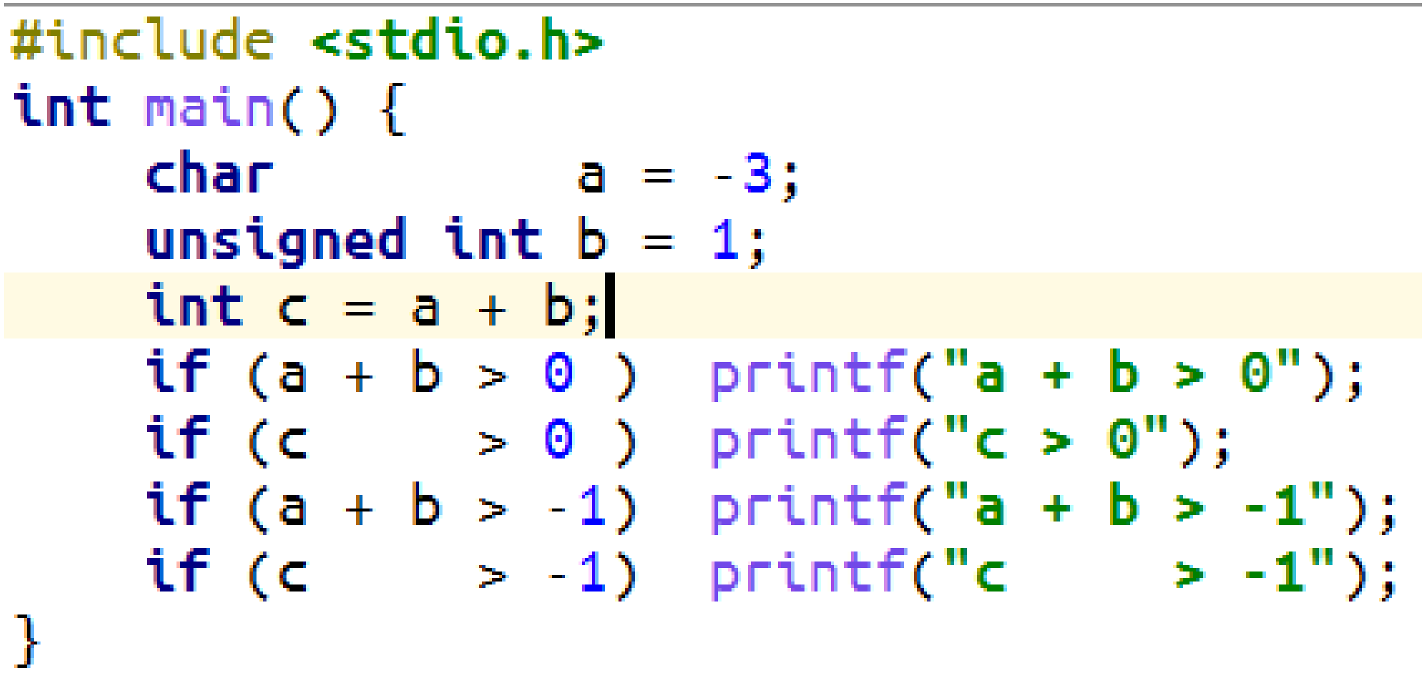
Deal with unknown cases by explicitly casting!

A

Implicit Casting

How many unsafe casting happen in this code?

Using memory point of view to understand the result.



```c
#include <stdio.h>
int main() {
    char         a = -3;
    unsigned int b = 1;
    int c = a + b;
    if (a + b > 0 )  printf("a + b > 0");
    if (c     > 0 )  printf("c > 0");
    if (a + b > -1)  printf("a + b > -1");
    if (c     > -1)  printf("c     > -1");
}
```

A

Two types of divisions:

Float point division: works on `float` and `double`

`a / b` if both sides are float points, returns float point results

Integer division: works integer data types

`a / b` if both sides are integers, returns quo. No rounding.

`7.0 / 3` and `7 / 3` returns different results?

Consider implicit casting! What is type of each literal?

A

`typedef` Keyword

`typedef` defines new “types”

By creating aliases with (combination of) existing types.

- `typedef unsigned int byte;`
  - Defines type `byte` as `unsigned char`
  - `byte a=0;` a is `byte` type, a.k.a. `unsigned char`
- Now best use with structures.
- Future to work with “pointers” (no worry)

A

Structure

STRUCTURES

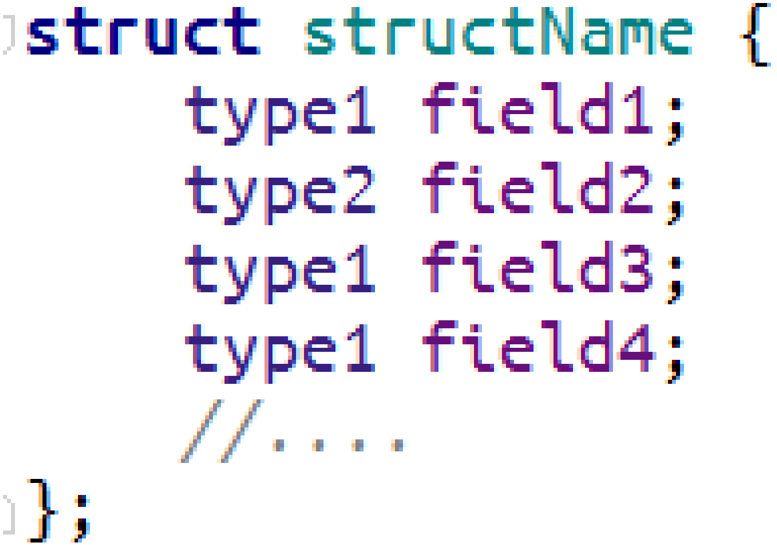
Consider structures a user defined type.

- Never cast structure variables

A

Structure Syntax

STRUCTURES



```
struct structName {
    type1 field1;
    type2 field2;
    type1 field3;
    type1 field4;
    //....
};
```

Defines a structure type “**stuct structName**”

A

Structure Syntax

STRUCTURES

Defines a structural type “`struct structName`”

Define variables by same syntax as normal types.

```
struct structName var1, var2;
```

You can choose to initialize `var1` or not.
Keep in mind data in `var1` is undefined before initializing it.

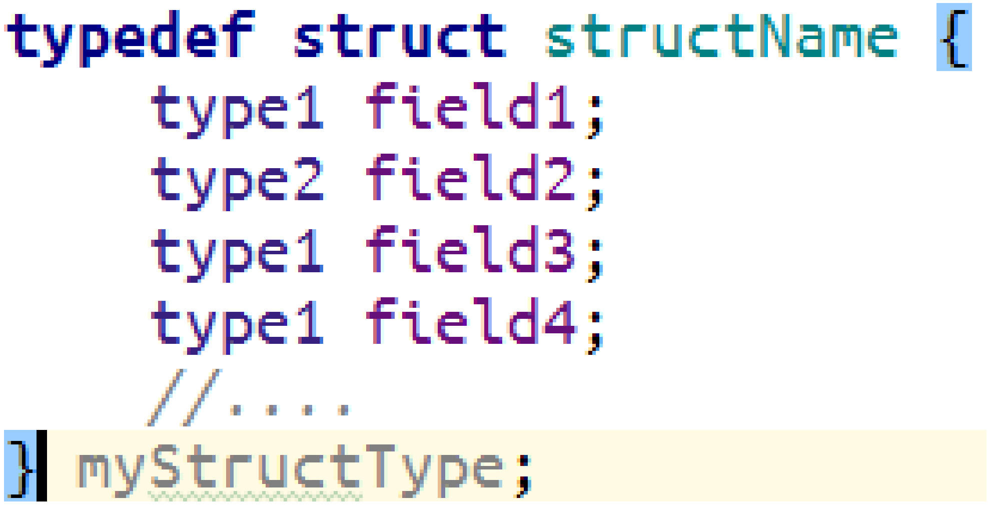
A Use typedef to cut down syntax

STRUCTURES

Do “struct structName myStructType;

In previous case.

Or simply do



```
typedef struct structName {
        type1 field1;
        type2 field2;
        type1 field3;
        type1 field4;
        //....
} myStructType;
```

A

Structure

Pay attention to initialization

Others are basically the same as in Matlab

Use structures to pack related information!

Results in much clearer logic

HTTP 404

Next slide not found (Thanks for reading)

Why would this happen?

It was such a happy thing

It’s the first time that I read through this file

It’s also the first time that I started to like this course

Combining these, it should be even better

I should have got a dream like experience.

But why would this happen?

Adapted from “White Album”