

Today's plan

- Class interactions
 - Ex 2-2
 - Keys points
 - Recap discussion
- Class exercises
 - Ex 2-3

Ex 2-2

Volunteers? (We need three)

- Practicing conditionals: `switch`, `if`
- Using ASCII codes for comparison or as array indices
- Array initialization
- A little bit about the “null terminator”

Q & A

<https://pigeonhole.at/3VE467/q/1580034>

Key points -file I/O

- Difference between `printf` and `fprintf`, `scanf` and `fscanf`.
- `stdin`, `stdout` is a “kind” of file.
- Open a file for read/write/append:
`fopen("filename", "flags")`, which return a file handler `FILE*` - a pointer to a structure of file.
- Check if opening the file successfully: if the handler is not `NULL`.
- Other file I/O functions: `fEOF`, `ferror`, `rewind`, etc.

Key points - assertion and math functions

- Assertions: `assert`. Use it to check for something should **NEVER** happen in your program.
- Assertions vs conditionals: why not just use `if` to catch the error, print the error message, and exit the program with an error code? What is the difference?
- Essentially, the behaviors are the same, but we use conditionals for program logical flows (e.g. handling different scenarios based on user input), and use assertions to make sure our assumptions of the logic (something we must have for the logic).
- Furthermore, we can disable assertions by adding `-DNDEBUG` in our gcc compilation command.
- Checkout the C math functions. Pay attention to the return type and parameter types. Remember to link: `-lm`
- Type promotion: `int` \rightarrow `float` \rightarrow `double`

Key points - functions

- How to write a function in C?

```
1  int foo(char c, int i) {  
2      return i ? c : c + i;  
3  }
```

- What is a valid function name in C?
 - Start with alphabet or _
 - The rest can be alphabet, digits, and _
 - “implementation-defined” characters (e.g. \$, @, etc.), which is not always guaranteed.
- `void foo();` vs `void foo(void);` in C.
 - `void foo();` accepts any number of arguments
 - `void foo(void);` accepts no arguments
- Only passed by value in C: i.e. parameters are local copies.
Caution: when passing an array, the address is copied. That's why we can change the values in a function.
- Command-line arguments: `argc` and `argv`

When should we use assertions instead of an if statement?

To validate, not to use in logic

for something that should never happen in the program, whereas if for user input

When we want to check for errors that would go against the whole logic of the code

If you're checking for something that implies that should never happen, use assert. If checking for something like bad user input or another strange but not impossible situation, use if.

Assertions are for when something should not happen in your code. It's an assumption that should be true

We should use assertions when we check for major flaws in our coding logic

when we are checking our assumptions in order for the program to work

To check for something that should never happen in the program

for checking to make sure we make the correct assumptions

When should we use assertions instead of an if statement?

int will be turned to double; an error occurs

What will happen if you pass an int variable to a function that takes a double as its parameter? What will happen if a double is passed to an int?

With argument type promotion we can have int -> float -> double. So, passing an int variable to function that takes a double as its parameter will work. But, a double passed to a function that takes an int parameter will fail.

It will turn the int into a double, but doing it the other way around will return an error

it would switch to double

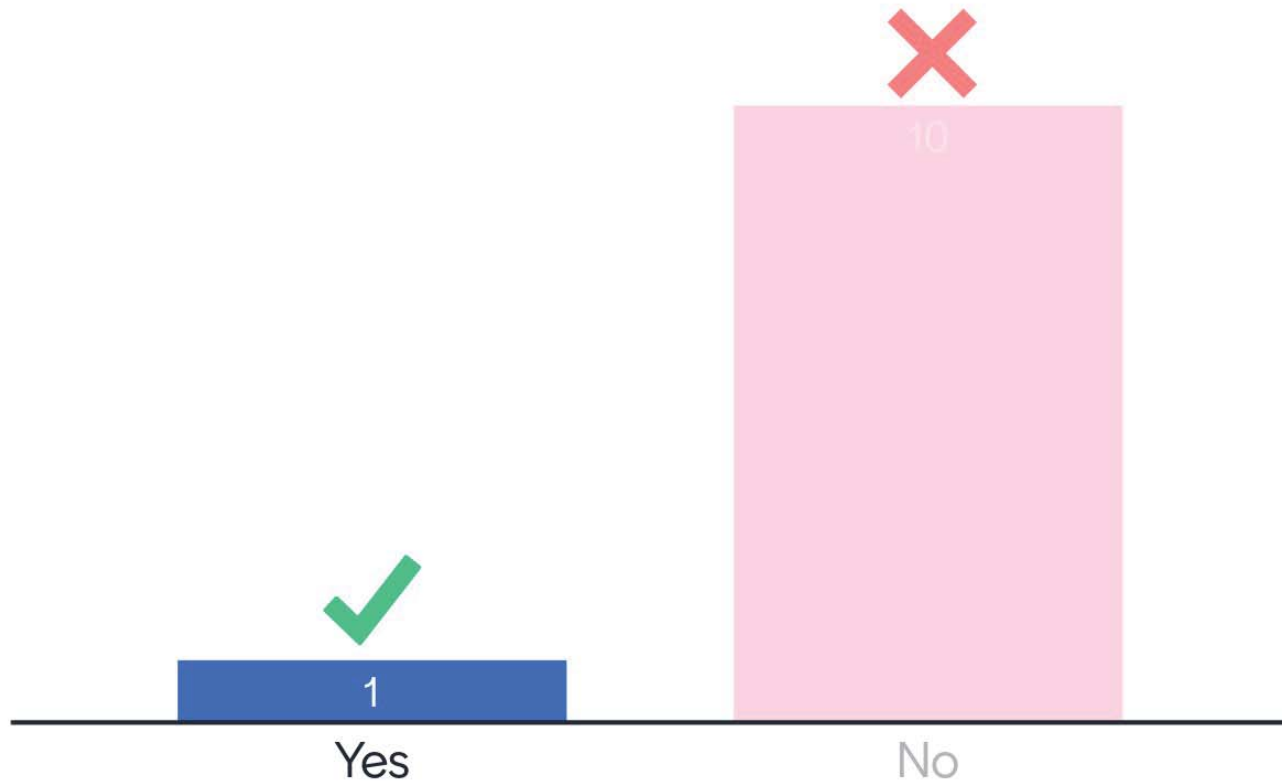
An int could be passed to a function that takes a double, but not the other way around

int into a double will work but not the other way around


converted to double, if a double to an int, fail

This should
be stdout


Is fprintf(stdin, "xxx") the same as printf("xxx")?



What is "passed by value"?


by value not by reference imply that the original variable value would not be changed 

temp variables passed into functions 

you are not passing the original variable to a function (you are passing a temporary variable that is a copy) 

Variables are local 

copy of parameter value in memory 

value of variable is passed as an argument 

a copy of the data is made and passed 

A temporary value 

function is given argument values in temporary variables (not originals) 

The correct answer is: Parameters are copies of the passing values.

How do you change the main function so that it can accept command-line arguments?

argv



2x

int main(int argc, char* argv[])



2x

argc argv



main(int argc and char *argv[])



int main(int args, char* args[])



argv[]



Add "char* argv[]" to the input



The correct answer is: `int main(int argc, char* argv[])`

Recap questions

Given the function definition below, which function call is invalid in C?

```
1  int abs(int a) {  
2      if (a < 0)  
3          return -a;  
4      return a;  
5  }
```

A `abs(2/5);`

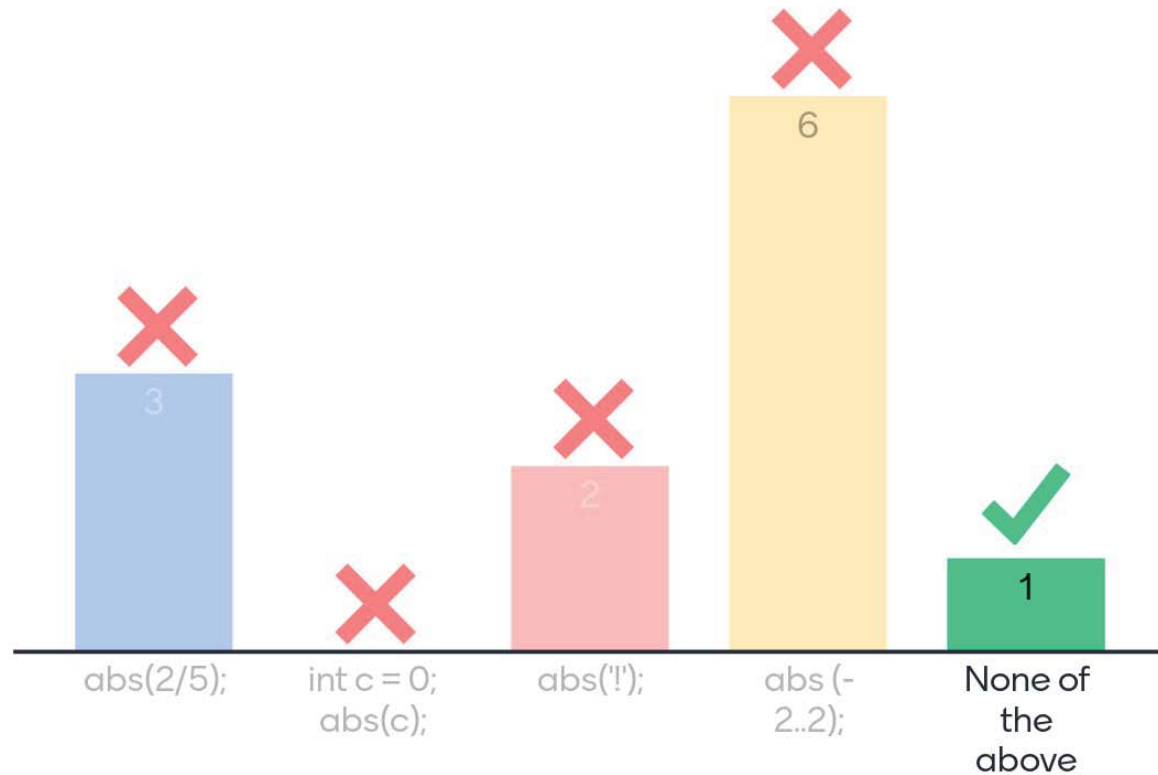
B `int c = 0; abs(c);`

C `abs('!');`

D `abs (-2.2);`

E None of the above

Given the function on the slide, which function call is invalid in C?



Recap questions

Which function name(s) below is/are guaranteed to be valid in C?

A `function_092`

B `fun$12`

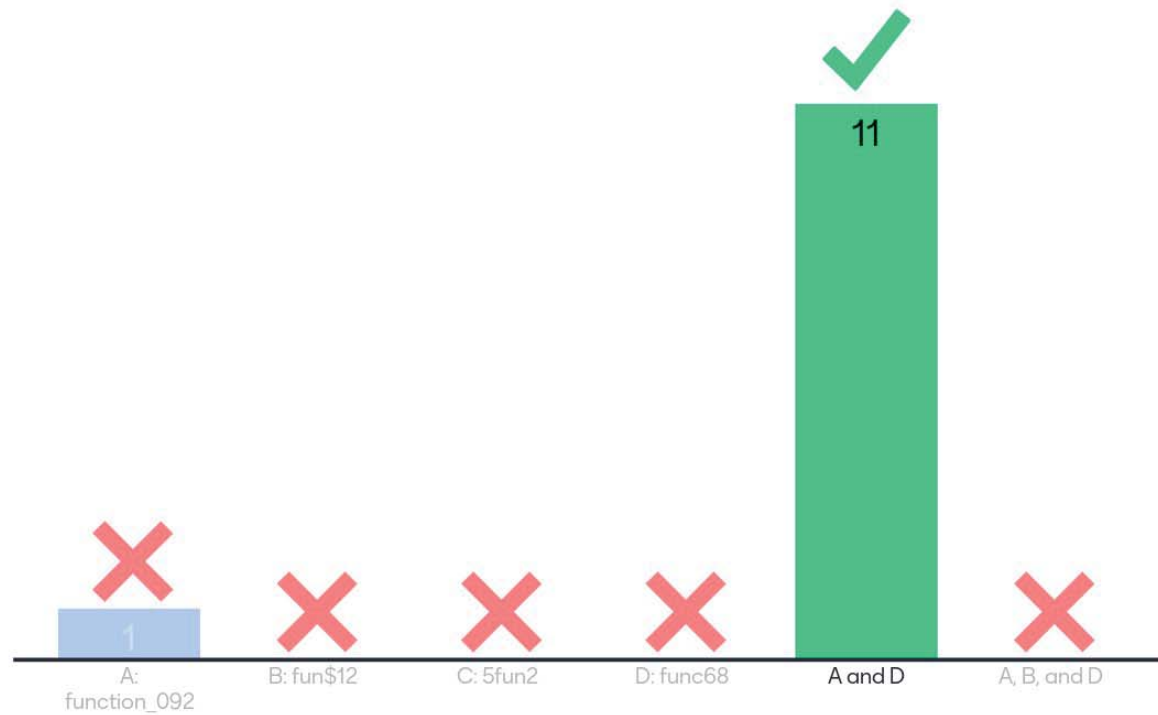
C `5fun2`

D `func68`

E A and D

F A, B, and D

Which function name(s) below is/are guaranteed to be valid in C?



Class exercises

Ex 2-3