

CS2002 - C Lecture 3 - Arrays

## CS2002 **Computer Systems** Lecture 3

#### Arrays

Jon Lewis (JC 0.26) School of Computer Science University of St Andrews

1

CS2002 - C Lecture 3 - Arrays

# The ? operator

- The ? operator in C can be used as a shorthand for if-else in an expression.
- The general form is

```
expression1 ? expression2 : expression3
max = (a > b)? a: b;
                                   // if (a > b) max = a;
// else max = b;
                                   // if (a > b) return(a);
// else return(b);
return ((a > b)? a: b);
                                   // float f;
// if (x < 0) f = -x;
// else f = x;
float f = x < 0? -x: x;
```



CS2002 - C Lecture 3 - Arrays

## Overview

- a little bit more on Control
- · ? Operator, switch, break, continue, goto
- booleans
- Arrays
  - Basics
  - · strings (just Arrays of char)
  - · command line arguments
- Preprocessor defines, macro functions



CS2002 - C Lecture 3 - Arrays

## Switch statements

- Switch statements take an integer or enum (to come later)
- each case must be a literal (const value)
- default: is a special label, which catches all other values.
- End each case with a break; else next case will be executed as well.

7

```
goto

Has limited use in C, where you have no exception handling.
Only works within a function.
goto bob; jumps to the label bob:

void f() {
  int i = 1;
  start:
  if (i < 10) {
    g(i++);
    goto start;
  }
}
```

CS2002 - C Lecture 3 - Arrays

## **Exiting loops**

- break and continue can be used (carefully) to exit the surrounding statement. continue jumps to the next loop iterator. A break jumps out of the (innermost) loop.
- Usually better to use loop conditions to exit

6



CS2002 - C Lecture 3 - Arrays

8

## Uses for goto

- Uses of goto should be minimised, but there are two places where it may be seen in C
- · Escaping from inside multiple loops.
- · Error handling.
  - Have a cleanup: label at the end of a function, and jump to it when you finish.

You should not **need** to use it (in this module)!



CS2002 - C Lecture 3 - Arrays

#### **Booleans**

- Until 1999, C had no Boolean type.
- Most code you will see still doesn't use the boolean type.
  - Non-zero integer values treated as true
- You can get it by doing:

#include <stdbool.h>

• Including this header introduces bool (just like boolean in Java), true and false.

9



CS2002 - C Lecture 3 - Arrays

П

### Arrays

- Arrays do not have "methods" attached to them, like in Java.
- Giving an array into a function passes a pointer to the array, not a
- You cannot return arrays from functions.

```
void change(int a[], int v) {
    a[0] = 2; // modifies original array!
    v = 2;
int main() {
    int array[1] = {1}; // array is [1]
    int val = 1;
    change(array, val);
    printf("%d, %d\n", array[0], val); // 2, 1
```

CS2002 - C Lecture 3 - Arrays

#### Arrays

- · Arrays are superficially similar in C and Java.
- · C arrays are indexed from 0.
- There is (almost) no way of getting the length of an array once it has been passed to a function.
- C arrays are not bounds checked (clang is better than gcc)

```
int main() {
    int doubles[10];
    doubles[0] = 1;
    for (int i = 1; i < 10; i++) doubles[i] = 2 * doubles[i - 1];</pre>
    for (int i = 0; i < 10; i++) printf("2^%i = %i\n", i,</pre>
doubles[i]);
```

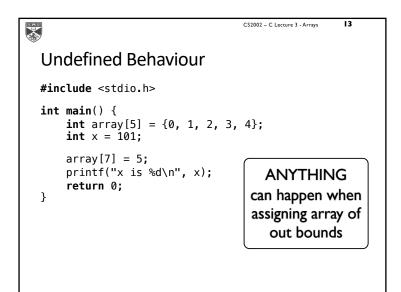
10



CS2002 - C Lecture 3 - Arrays

#### Arrays

- Passing an array to a function actually passes a pointer to the start of the array to the function.
- a == b : Always false, compares if the arrays are the same 'object' (like Java), i.e. memory location
- a = b : Won't compile.





CS2002 - C Lecture 3 - Arrays

15

- C does not have a string type
- By convention, a string is an array of chars terminated by the NULL character '\0'.
- '\0' is just the 0 of the char type, but you can't write that in a string.
- Always allocate 1 more byte than string length!

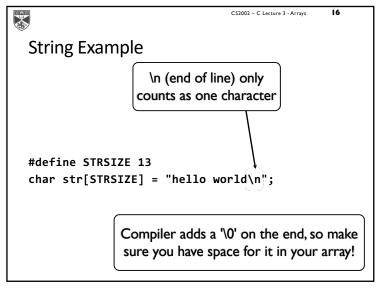
```
Undefined Behaviour

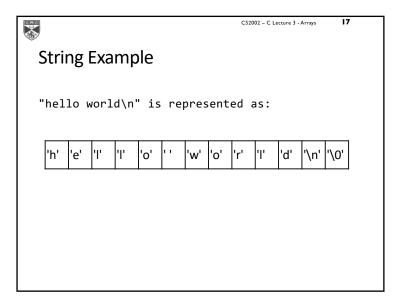
#include <stdio.h>

int main() {
    int array[5] = {0, 1, 2, 3, 4};
    int x = 101;
    array[7] = 5;
    printf("x is %d\n", x);
    return 0;
}

This prints 'x is 5' when compiled with gcc on lab
    gcc -Wall -Wextra -O0 undefined_behaviour_gcc.c
```

14





```
Command Line Arguments

Command line arguments are passed into main, similarly to java.

Note: for C, argv[0] is the program name.

#include <stdio.h>
int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; i++)
    printf("argv[%i]: %s\n", i, argv[i]);
  return 0;
}
```

```
String Example
#include <stdio.h>
// strings are just null-terminated char arrays
// compiler fills in size of 'str'
char str[] = "hello world\n";

int main() {
    // can print as chars
    for (int i = 0; str[i] != '\0'; i++) {
        printf("%c", str[i]);
    }
    // can print as string
    printf("%s", str);
}
```

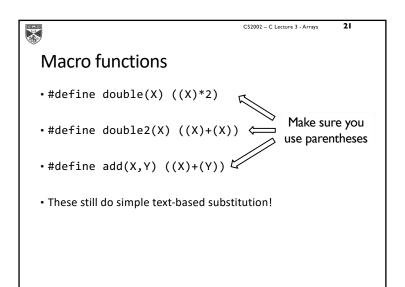
18

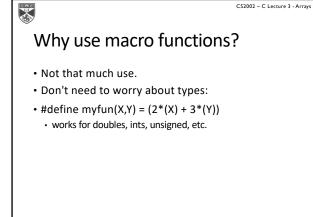
```
Preprocessor Defines

• Performs simple substitutions, can be used for a number of purposes from providing simple defaults

#define DEFAULT_X 0

int main() {
   int x = DEFAULT_X;
   printf("x = %d\n", x);
   return 0;
}
```





```
// Bad example without parentheses
#define double(X) X+X
#define mult(X,Y) X*Y

int main() {
   int i = 1, j = 2;
   printf("%d\n", 2 * double(i));
   printf("%d\n", mult(i + j, i - j));

   return 0;
}

Rarely, use macros, but if you do, include parentheses, i.e.
#define double(X) ((X)+(X))
#define mult(X,Y) ((X)*(Y))
```

```
Preprocessor

There are some other useful preprocessor commands:

#ifdef, #ifndef, #else, #endif

Define a label in the preprocessor

#define X

#define X

#define X a

Include some code only if a symbol is defined (or not defined)

#ifdef X

#ifdef X

#else

#endif
```

```
// define_debug.c
#include <stdio.h>

#define DEBUG // comment out to disable DEBUG

int main() {
   int result = 0;
   // code here to compute/alter value of result
#ifdef DEBUG
   printf("DEBUG main.c: result = %i\n", result);
#endif
   return result;
}
```