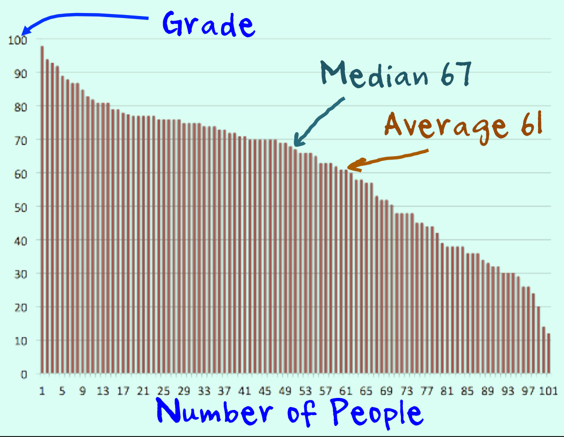
CS205

C / C++

Stéphane Faroult
faroult@sustc.edu.cn

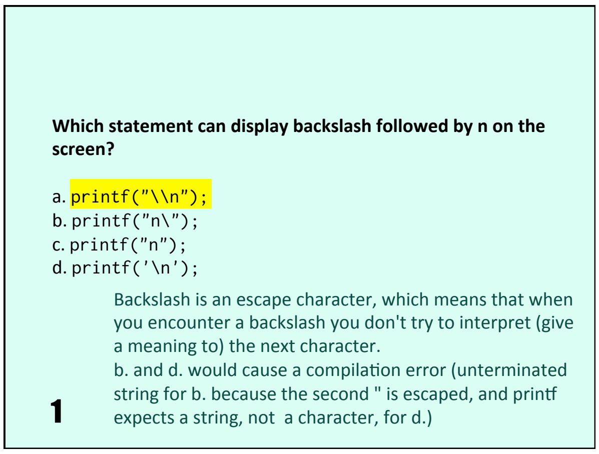
Wang Wei (Vivian) vivian2017@aliyun.com



I don't see any particular problem for people above 40%. Obviously if you are near the low end you haven't quite understood everything yet but the course is fast, C is difficult, and if you have little programming experience I'm confident that with time you'll get better. If you are between 30 and 40% and do relatively well in lab, you should also manage to do it. Below 30%, it's not impossible for you to pass, but make sure that you understand well what you do in labs, and review the exam questions in detail. If something is still unclear, both Vivian and I can try to make it clearer for you. The final exam is in the same format and covers ALL the course, and so you can expect a significant part of it to cover the same ground as the MidCourse exam.

Part 1

Average: 16.4 / 25

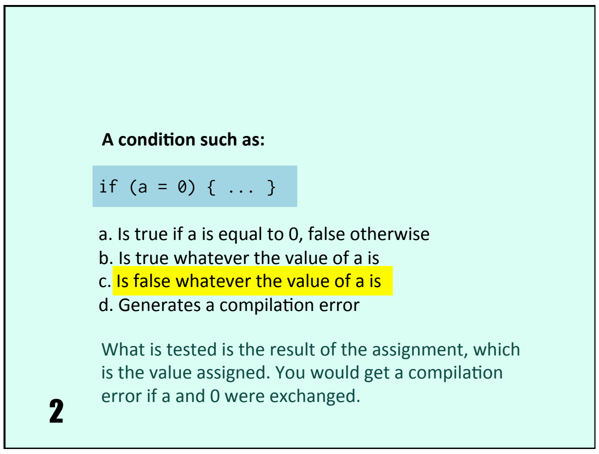


Which statement can display backslash followed by n on the screen?

a. `printf(”\\n”);`
b. `printf(”n\”);`
c. `printf(”n”);`
d. `printf('\n');`

Backslash is an escape character, which means that when you encounter a backslash you don't try to interpret (give a meaning to) the next character.
b. and d. would cause a compilation error (unterminated string for b. because the second " is escaped, and printf expects a string, not a character, for d.)

1



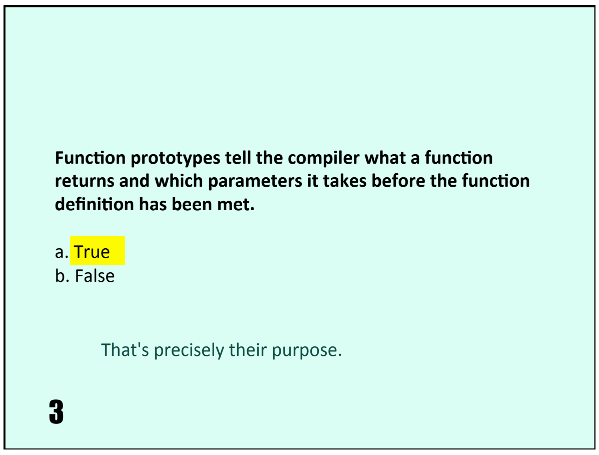
A condition such as:

```
if (a = 0) { ... }
```

a. Is true if a is equal to 0, false otherwise
b. Is true whatever the value of a is
c. Is false whatever the value of a is
d. Generates a compilation error

What is tested is the result of the assignment, which is the value assigned. You would get a compilation error if a and 0 were exchanged.

2

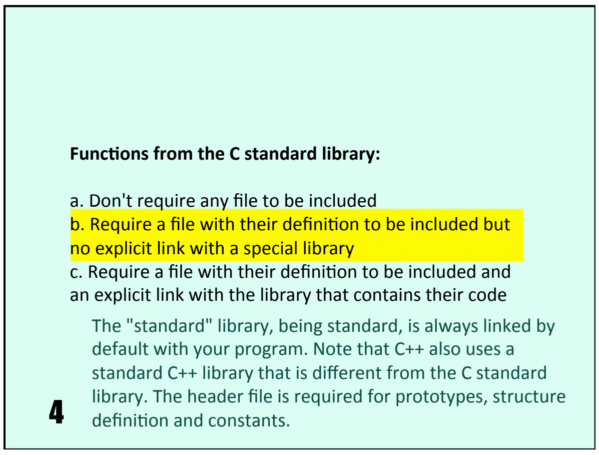


Function prototypes tell the compiler what a function returns and which parameters it takes before the function definition has been met.

a. True
b. False

That's precisely their purpose.

3



Functions from the C standard library:

a. Don't require any file to be included
b. Require a file with their definition to be included but no explicit link with a special library
c. Require a file with their definition to be included and an explicit link with the library that contains their code

The "standard" library, being standard, is always linked by default with your program. Note that C++ also uses a standard C++ library that is different from the C standard library. The header file is required for prototypes, structure definition and constants.

4

What does the following program print:

```
  #include <stdio.h>

  int main() {
    char str[50] = "SUSTech";
    char *ptr = str;
    printf("%c",*ptr);
1   printf("%c",*(++ptr));
2   printf("%c",*ptr++);
3   printf("%c",*(++ptr));
    putchar('\n');
    return 0;
  }
```

a. SUST
b. SUUT
c. SSSS
d. SUSTech

On lines 1 and 3, the pointer is incremented, then dereferenced. On line 2 it's dereferenced, then incremented, so U is displayed a second time, and the second S is skipped on line 3.

5

Consider the following code snippet:

```
typedef struct {
      short a;
      int   b;
      int   c;
      float d; } DUMMY_TYPE;
```

How many bytes does DUMMY_TYPE occupy in memory?

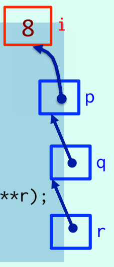
a. 10
b. 12
c. 13
d. 14

short = 2, int = 4 (most commonly), float = 4

6

Consider the code snippet below.

```
#include<stdio.h>

int main() {
     int ***r, **q, *p, i=8;
     p = &i;
     q = &p;
     r = &q;
     printf("%d, %d, %d\n", *p, **q, ***r);
     return 0;
}
```



What is the output if the integer is 4 byte long?

a. 8, 8, 8
b. 4000, 4002, 4004
c. 4000, 4004, 4008
d. 4000, 4008, 4016

Serial dereferencing.
The information about the integer size is irrelevant, just to try to fool you.

7

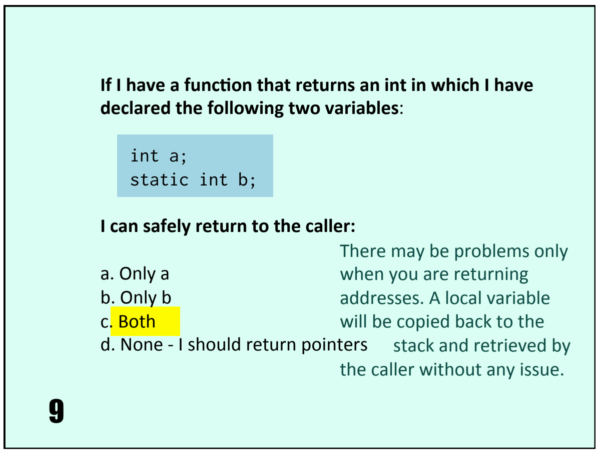
What is the output of the following code snippet?

```
int main() {
 int arr[4] = {1,2,3,4};
 printf("%d\n", arr[4]);
 return 0;
}
```

Memory that is reserved are slots 0 to 3 in the array.
Slot 4 is a step in the Great Unknown, outside the array. Normally the compiler will catch it.

a. 3
b. 4
c. 0
d. Garbage or nothing as the program may fail to compile or crash when running.

8



If I have a function that returns an int in which I have declared the following two variables:

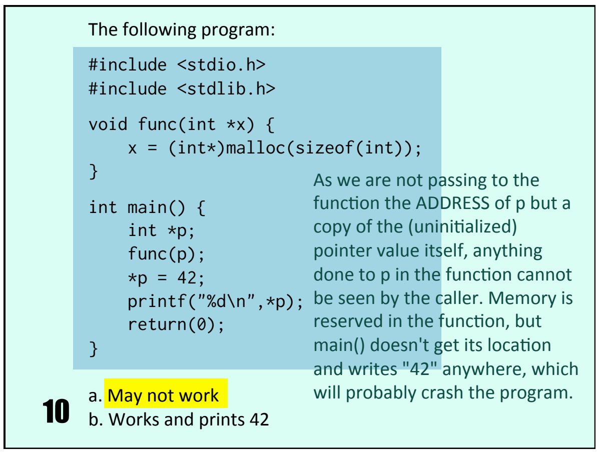
```
int a;
static int b;
```

I can safely return to the caller:

a. Only a
b. Only b
c. [✓ Both]
d. None - I should return pointers

There may be problems only when you are returning addresses. A local variable will be copied back to the stack and retrieved by the caller without any issue.

9



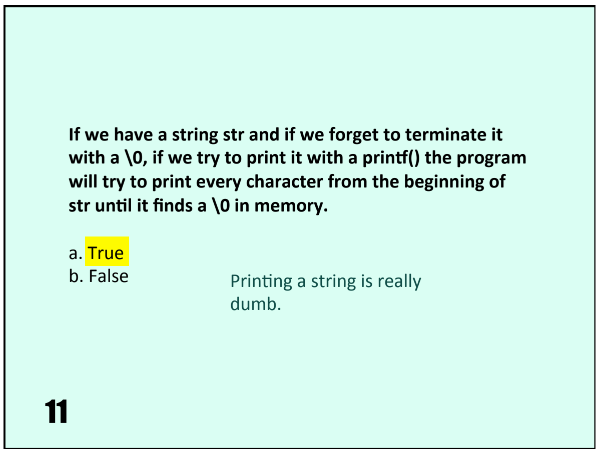
The following program:

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

void func(int *x) {
    x = (int*)malloc(sizeof(int));
}

int main() {
    int *p;
    func(p);
    *p = 42;
    printf("%d\n",*p);
    return(0);
}
```

a. [✓ May not work]
b. Works and prints 42

As we are not passing to the function the ADDRESS of p but a copy of the (uninitialized) pointer value itself, anything done to p in the function cannot be seen by the caller. Memory is reserved in the function, but main() doesn't get its location and writes "42" anywhere, which will probably crash the program.

10

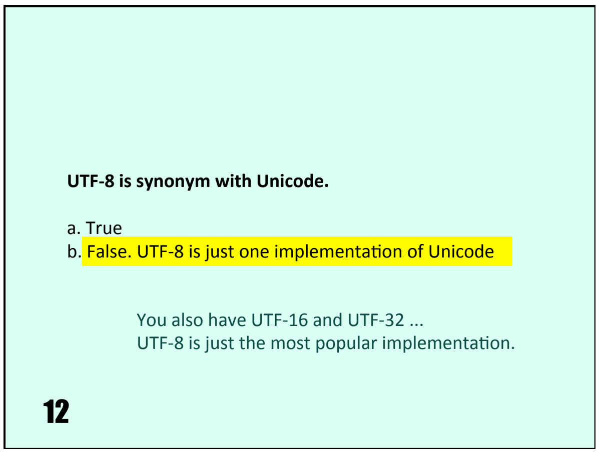


If we have a string str and if we forget to terminate it with a \0, if we try to print it with a printf() the program will try to print every character from the beginning of str until it finds a \0 in memory.

a. [✓ True]
b. False

Printing a string is really dumb.

11

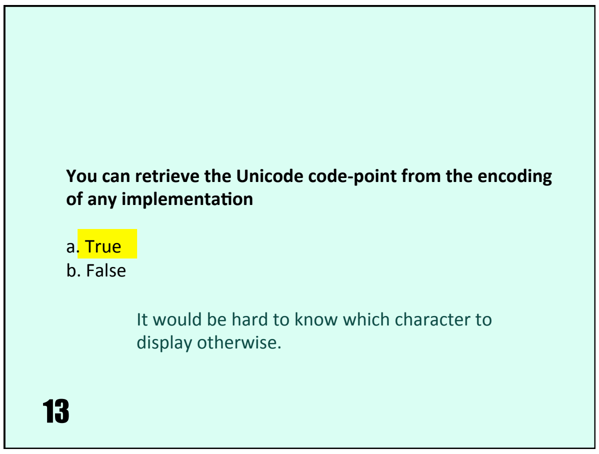


UTF-8 is synonym with Unicode.

a. True
b. [✓ False. UTF-8 is just one implementation of Unicode]

You also have UTF-16 and UTF-32 ...
UTF-8 is just the most popular implementation.

12

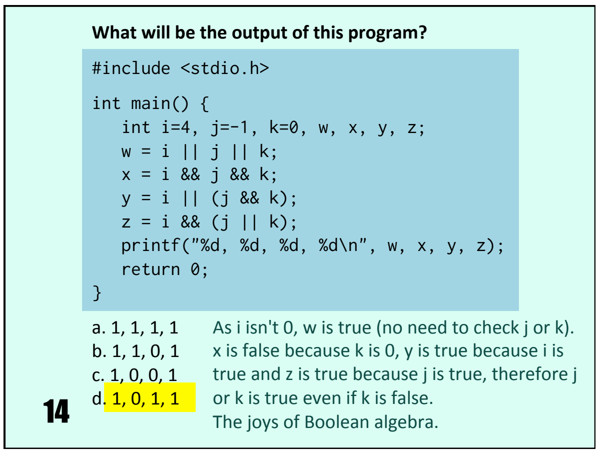


You can retrieve the Unicode code-point from the encoding of any implementation

a. [✓ True]
b. False

It would be hard to know which character to display otherwise.

13



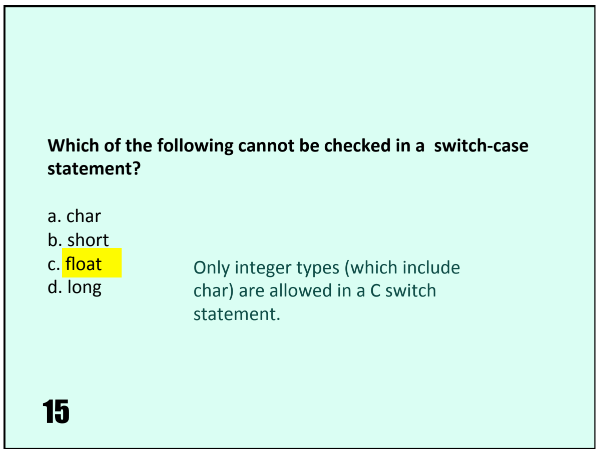
What will be the output of this program?

```
#include <stdio.h>

int main() {
   int i=4, j=-1, k=0, w, x, y, z;
   w = i || j || k;
   x = i && j && k;
   y = i || (j && k);
   z = i && (j || k);
   printf("%d, %d, %d, %d\n", w, x, y, z);
   return 0;
}
```

a. 1, 1, 1, 1
b. 1, 1, 0, 1
c. 1, 0, 0, 1
d. [✓ 1, 0, 1, 1]

As i isn't 0, w is true (no need to check j or k). x is false because k is 0, y is true because i is true and z is true because j is true, therefore j or k is true even if k is false.
The joys of Boolean algebra.

14

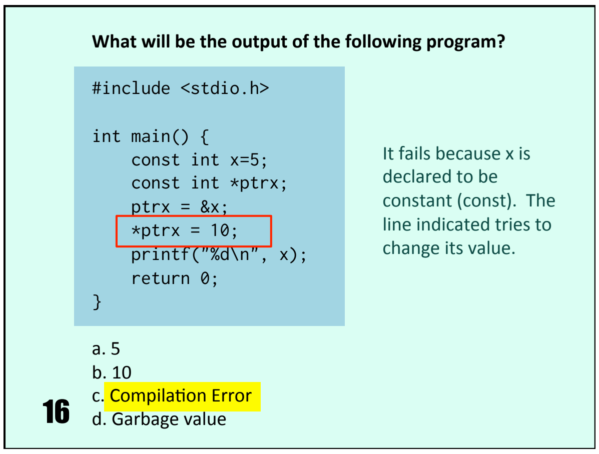


Which of the following cannot be checked in a switch-case statement?

a. char
b. short
c. [✓ float]
d. long

Only integer types (which include char) are allowed in a C switch statement.

15



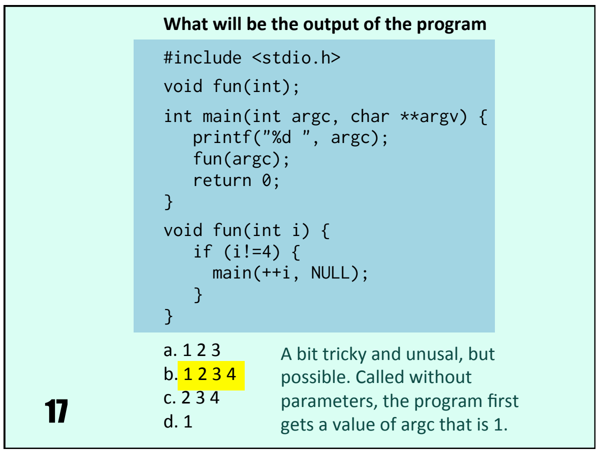
What will be the output of the following program?

```
#include <stdio.h>

int main() {
    const int x=5;
    const int *ptrx;
    ptrx = &x;
    *ptrx = 10;
    printf("%d\n", x);
    return 0;
}
```

It fails because x is declared to be constant (const). The line indicated tries to change its value.

a. 5
b. 10
c. [✓ Compilation Error]
d. Garbage value

16



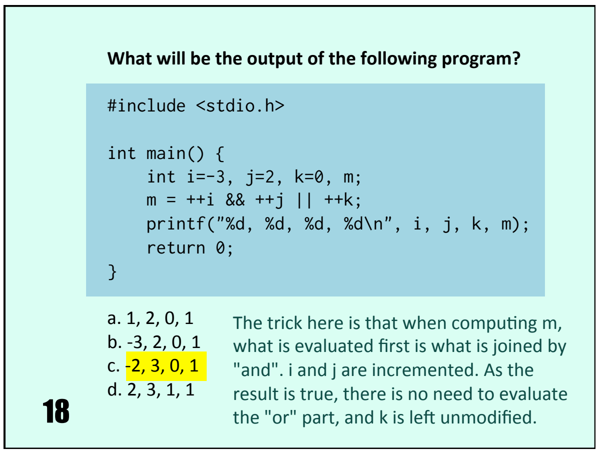
What will be the output of the program

```
#include <stdio.h>
void fun(int);
int main(int argc, char **argv) {
   printf("%d ", argc);
   fun(argc);
   return 0;
}
void fun(int i) {
   if (i!=4) {
     main(++i, NULL);
   }
}
```

a. 1 2 3
b. [✓ 1 2 3 4]
c. 2 3 4
d. 1

A bit tricky and unusual, but possible. Called without parameters, the program first gets a value of argc that is 1.

17



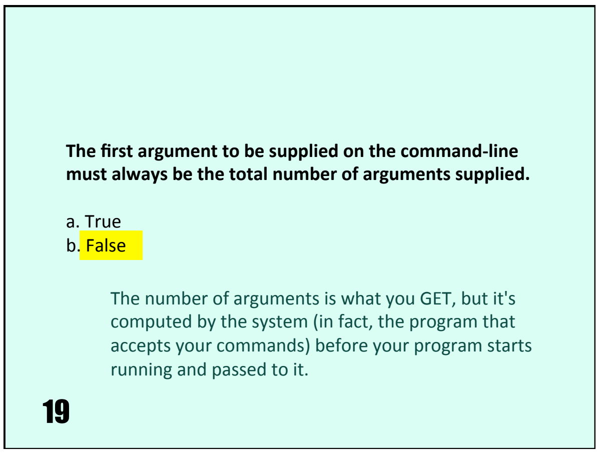
What will be the output of the following program?

```
#include <stdio.h>

int main() {
    int i=-3, j=2, k=0, m;
    m = ++i && ++j || ++k;
    printf("%d, %d, %d, %d\n", i, j, k, m);
    return 0;
}
```

a. 1, 2, 0, 1
b. -3, 2, 0, 1
c. [✓ -2, 3, 0, 1]
d. 2, 3, 1, 1

The trick here is that when computing m, what is evaluated first is what is joined by "and". i and j are incremented. As the result is true, there is no need to evaluate the "or" part, and k is left unmodified.

18

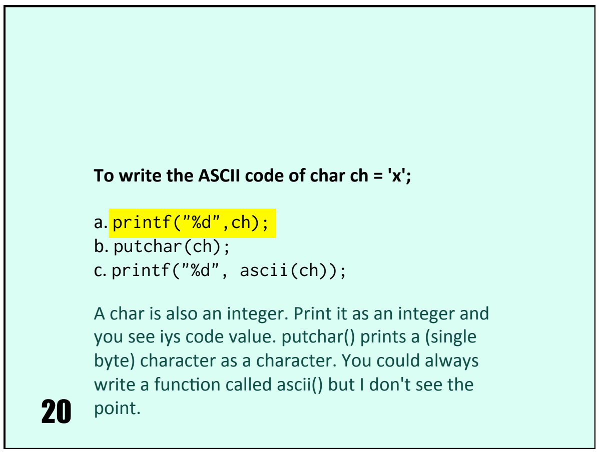


The first argument to be supplied on the command-line must always be the total number of arguments supplied.

a. True
b. [✓ False]

The number of arguments is what you GET, but it's computed by the system (in fact, the program that accepts your commands) before your program starts running and passed to it.

19



To write the ASCII code of char ch = 'x';

a. [✓ `printf("%d",ch);`]
b. `putchar(ch);`
c. `printf("%d", ascii(ch));`

A char is also an integer. Print it as an integer and you see iys code value. putchar() prints a (single byte) character as a character. You could always write a function called ascii() but I don't see the point.

20

What is the output of the following snippet:

```
int  change(int  x) {
 x = 7;
}

int main(){
  int x = 5;
  change(x);
  printf("%d\n",x);
  return 0;
}
```

Somebody rightly made the remark that the int functions returns nothing. Some compilers let it pass. gcc issues a warning but generates the program.

a. 5
b. 7
c. 0
d. unknown

A copy of x is passed, not it's address, and therefore it remains unchanged in main().

21

In the following snippet:

```
typedef struct my_node {
       char* info;
       struct my_node *next;
} NODE_T;
int main() {
    NODE_T *p = NULL;
    p = (NODE_T*)malloc(sizeof(NODE_T));
    return 0;
}
```

What is p?

p is definitely pointer to a NODE_T, reserved by malloc().

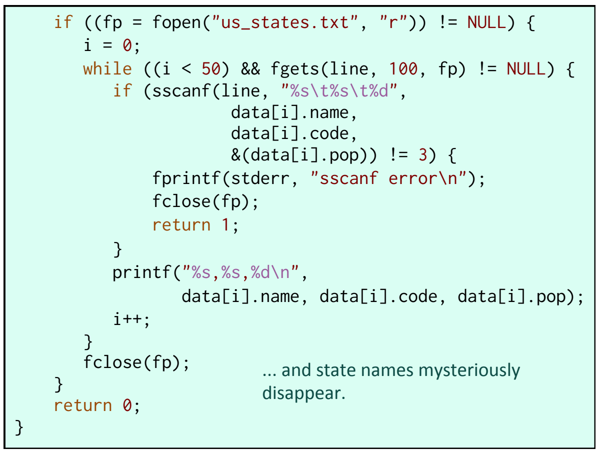
a. A variable of type NODE_T
b. A string of type NODE_T
c. A pointer of type NODE_T
d. A pointer to a block of memory of size, sizeof(NODE_T)
e. Both the two previous answers

22

Part 2

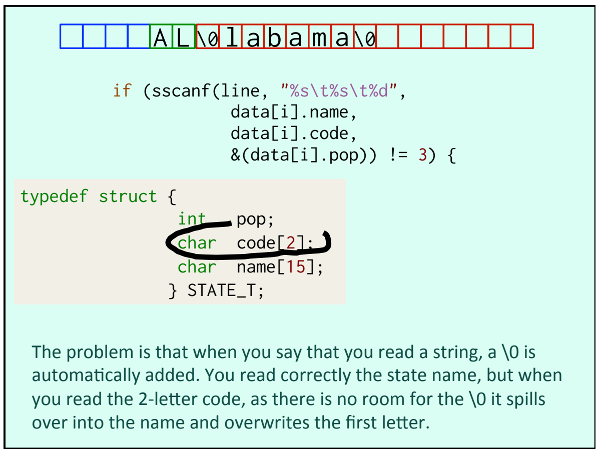
Average: 3 / 15

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

typedef struct {
                int   pop;
                char  code[2];
                char  name[15];
               } STATE_T;

static STATE_T data[50];

int main() {
    FILE *fp;
    int   i;
    char  line[100];
    char *p;
```



```
    if ((fp = fopen("us_states.txt", "r")) != NULL) {
       i = 0;
       while ((i < 50) && fgets(line, 100, fp) != NULL) {
          if (sscanf(line, "%s\t%s\t%d",
                      data[i].name,
                      data[i].code,
                      &(data[i].pop)) != 3) {
              fprintf(stderr, "sscanf error\n");
              fclose(fp);
              return 1;
          }
          printf("%s,%s,%d\n",
                 data[i].name, data[i].code, data[i].pop);
          i++;
       }
       fclose(fp);
    }
    return 0;
}
```

... and state names mysteriously disappear.

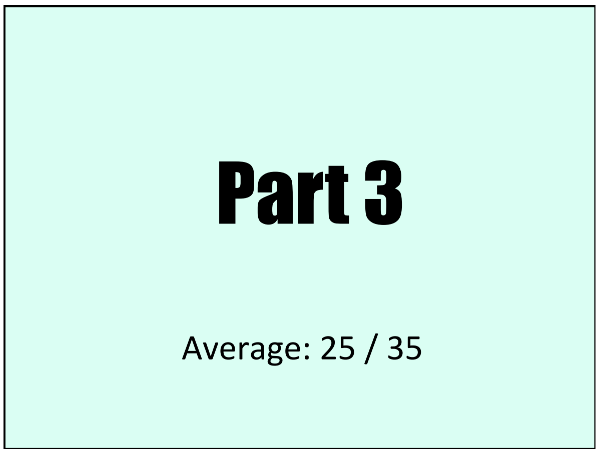


A L \0 l a b a m a \0

```
if (sscanf(line, "%s\t%s\t%d",
            data[i].name,
            data[i].code,
            &(data[i].pop)) != 3) {
```

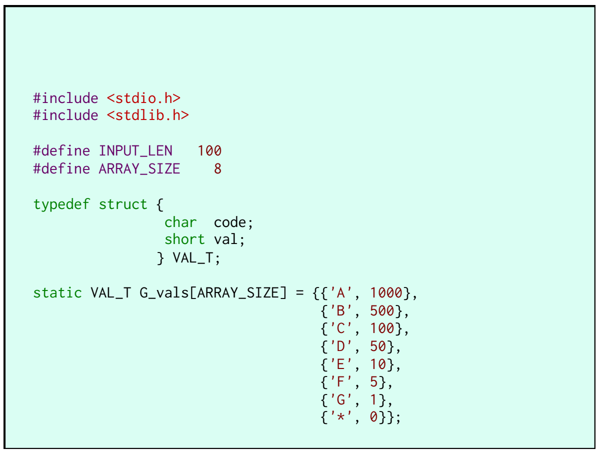
```
typedef struct {
               int    pop;
               char   code[2];
               char   name[15];
              } STATE_T;
```

The problem is that when you say that you read a string, a \0 is automatically added. You read correctly the state name, but when you read the 2-letter code, as there is no room for the \0 it spills over into the name and overwrites the first letter.



Part 3

Average: 25 / 35



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

#define INPUT_LEN   100
#define ARRAY_SIZE    8

typedef struct {
                char  code;
                short val;
               } VAL_T;

static VAL_T G_vals[ARRAY_SIZE] = {{'A', 1000},
                                   {'B', 500},
                                   {'C', 100},
                                   {'D', 50},
                                   {'E', 10},
                                   {'F', 5},
                                   {'G', 1},
                                   {'*', 0}};
```

```c
short get_val(char code) {
    short i = 0;

    G_vals[ARRAY_SIZE - 1].code = code;
    while (G_vals[i].code != code) {
      i++;
    }
    if (i == ARRAY_SIZE - 1) {
      return 0;
    } else {
      return G_vals[i].val;
    }
}
```

Plain search. Notice a clever trick (I found it in a classic book, "The Art of Computer Programming" by Donald Knuth), the searched value is added at the end of the array that is searched. It makes checking that i doesn't go too far unnecessary.

```c
int compute_value(char *str) {
    char *p;
    int   val = 0;
    int   next_val;
    int   remainder;

    if (str && *str) {
      p = str;
      p++;
      if (*p == '\0') {
        val = get_val(*str);
      } else {
        remainder = compute_value(p);
        val = get_val(*str);
        next_val = get_val(*p);
        if (val < next_val) {
          val = remainder - val;
        } else {
          val = remainder + val;
        }
      }
      printf("Evaluating %s = %d\n", str, val);
    }
    return val;
}
```

The recursive function. The thing to notice is that it calls itself as long as the string that is passed contains more than one character. In other words, it goes to the end, then walks back towards the beginning of the string.

```c
int main() {
    printf("%d\n", compute_value("CCEDFGG"));
    return 0;
}
```

Before printing anything, it must be computed. So we go to the end, then come back. Simply put, when the value of the current letter is bigger than the value of the next one, this value is added to what has already been computed, otherwise it's subtracted from it.

```
Evaluating G = 1
Evaluating GG = 2
Evaluating FGG = 7
Evaluating DFGG = 57
Evaluating EDFGG = 47
Evaluating CEDFGG = 147
Evaluating CCEDFGG = 247
247
```

Printed by the various calls to compute_value()

Printed by main()

That's how you compute roman numerals (with other letters – 247 would be CCXLVII in roman numerals)

Part 4

Average: 17.8 / 25

We want to process a string of characters that only contains unaccented uppercase characters (A to Z) and replace by a space every character that appears alone, that is every character that is different from the preceding character and from the following character.
For instance:
`"AASRFFZRRRZZDTT" => "AA  FF RRRZZ TT"`
Write the corresponding pseudocode.

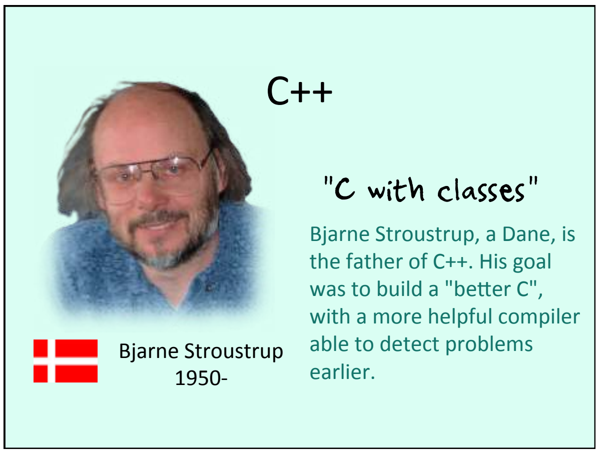
`"AASRFFZRRRZZDTT" => "AA  FF RRRZZ TT"`

We cannot only look at the next character when advancing in the string, we must simultaneously look at the previous one and the next one. There are always difficult spots in an algorithm: extreme cases. No idea about what precedes the first character (however, we know what follows the last one: \0).
There are different ways to handle the problem, more or less complicated. Some of the points were given for the elegance of the solution. The purpose of pseudo-code is basically that a competent programmer can write from it a program that works without even knowing precisely what is the purpose of the program. If it's too complicated, it increases the risks of errors. And it makes the program harder to modify in the future, if requirements change.

`"AASRFFZRRRZZDTT" => "AA  FF RRRZZ TT"`

```
set a char called prev to something that isn t a letter, eg *
set an int i to 0
loop on string as long as letter @ i isn t \0 (end of string)
    if letter @ i is different from prev
       and letter @ i is different from letter @ i + 1
        replace letter @ i by space
    end if
    set prev to letter @ i (can be space now but it changes nothing)
    increment i
end loop
```

C++

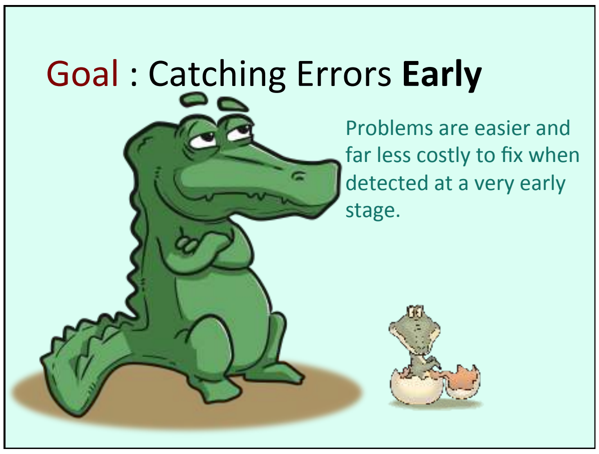


C++

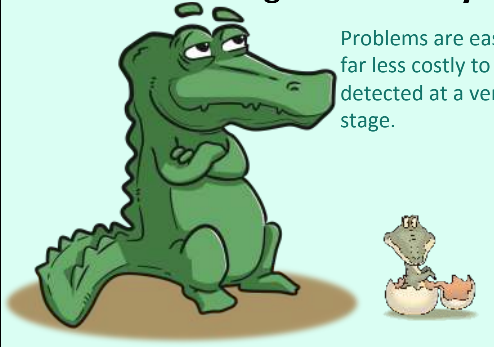
"C with classes"

Bjarne Stroustrup, a Dane, is the father of C++. His goal was to build a "better C", with a more helpful compiler able to detect problems earlier.

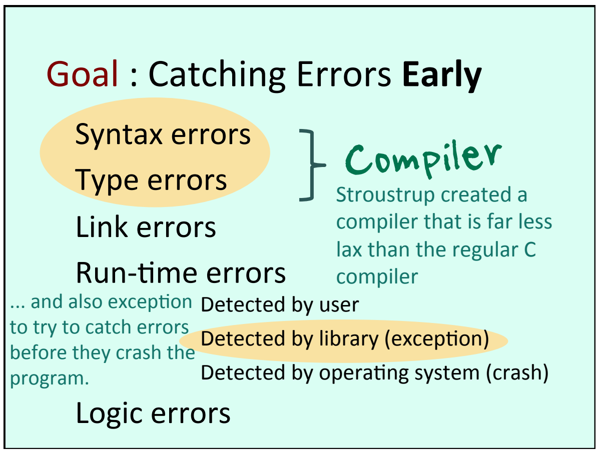
Bjarne Stroustrup
1950-



Goal : Catching Errors **Early**



Problems are easier and far less costly to fix when detected at a very early stage.



Goal : Catching Errors **Early**

Syntax errors
Type errors } Compiler

Stroustrup created a compiler that is far less lax than the regular C compiler

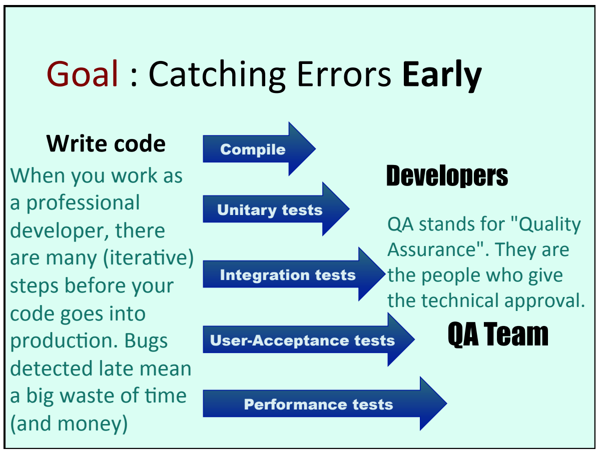
Link errors

Run-time errors

... and also exception to try to catch errors before they crash the program.

Detected by user
Detected by library (exception)
Detected by operating system (crash)

Logic errors



Goal : Catching Errors **Early**

Write code

When you work as a professional developer, there are many (iterative) steps before your code goes into production. Bugs detected late mean a big waste of time (and money)

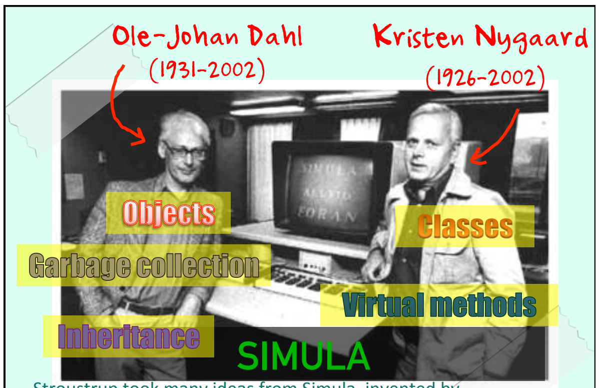
QA stands for "Quality Assurance". They are the people who give the technical approval.

Goal : Catching Errors **Early**

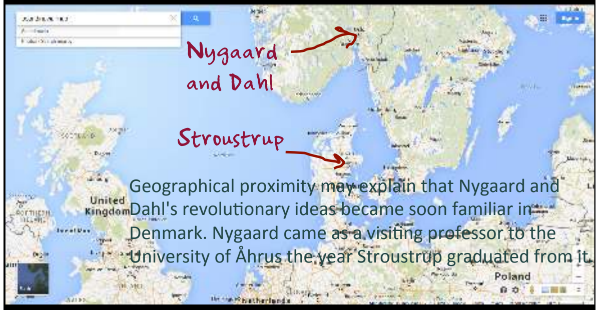
Problem in production

- Identify
- Fix & Test
- Non-regression tests
- Schedule new version release

Bugs found when the programs are in production are even worse. First they must be identified (very hard with some seemingly random bugs) and fixed. Then everybody fears that the fix introduces a new bug or breaks something that worked. Then it must be scheduled for release.



Stroustrup took many ideas from Simula, invented by two Norwegians who created all object-related concepts in the 1960s and became enormously influential.



Geographical proximity may explain that Nygaard and Dahl's revolutionary ideas became soon familiar in Denmark. Nygaard came as a visiting professor to the University of Århus the year Stroustrup graduated from it.

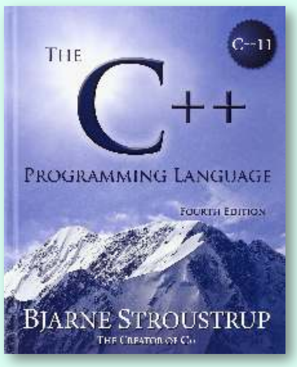
C++

Started in 1978

AT&T Bell Laboratories

Bjarne Stroustrup
1950-

Stroustrup began work on C++ as a doctoral student at the university of Cambridge (UK). He then spent most of his career at the Bell labs.



A tougher read than K&R. But timing was perfect, because C++ matured just when everybody was creating window interfaces, for which object-oriented programming helps a lot.

First published end 1985

PHILOSOPHY

Anchored in reality
- Solve actual problems
- Reasonably easy to implement
- Work alongside prior languages (eg C)

Performance
- No overhead due to unused features
- No language beneath C++ other than assembly

Developer freedom
- Programmers free to pick their style
- Manual override
- Allowing a useful feature is more important than preventing possible misuse

Stroustrup thought a lot about his language, and made a number of decisions which are different with other popular languages.

C++ not easier than C

... but makes code organization easier

"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off."

Bjarne Stroustrup

Extensions

	.cpp .cc	Many extensions are allowed, .cpp and .cc are the most popular ones.
header files	.h .hpp	.h files are "C compatible", .hpp files usually contain class definitions.

A C program is a valid C++ program ...

Even so, Stroustrup redesigned a number of features in what he was seeing as a cleaner way of programming. Many of these features were later adopted by other languages, some of them remain specific to C++. Note that some people happily mix C and C++ - which has always been considered valid by Stroustrup.
Let's take a quick look at the C++ innovations.

Obvious ~~differences with C~~ extensions to **C**

```
#include <stdio.h>



int main() {
    int val = 3;

    printf("val = %d\n", val);
    return 0;
}
```

Obvious ~~differences with C~~ extensions to **C++**

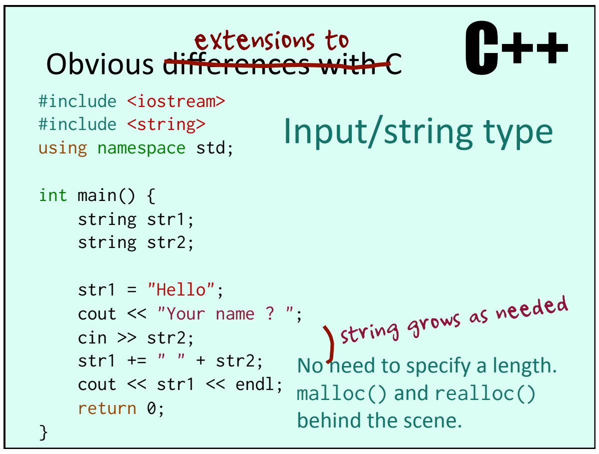
```
#include <iostream>

using namespace std;

int main() {
    int val = 3;

    cout << "val = " << val << endl;
    return 0;
}
```

iostream can replace stdio.h. Note that there is no extension specified.

without this you should refer to std::cout

'\n' for dummies

(in fact more portable than '\n')



Obvious ~~differences with~~ extensions to C

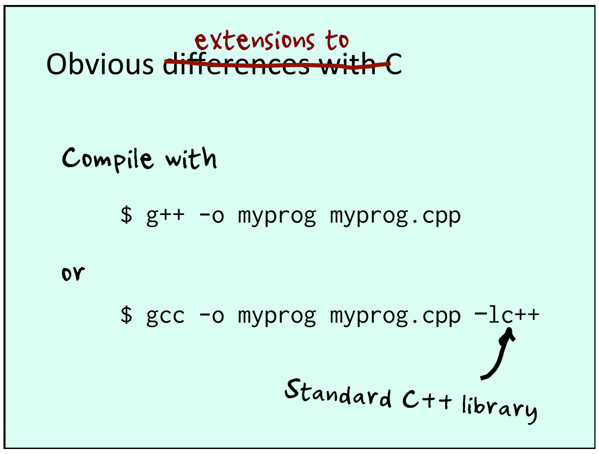
C++

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string str1;
    string str2;

    str1 = "Hello";
    cout << "Your name ? ";
    cin >> str2;
    str1 += " " + str2;
    cout << str1 << endl;
    return 0;
}
```

Input/string type

string grows as needed

No need to specify a length. malloc() and realloc() behind the scene.



Obvious ~~differences with~~ extensions to C

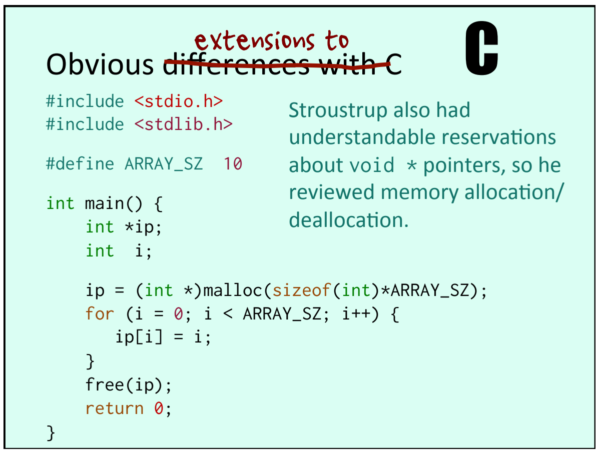
Compile with

```
$ g++ -o myprog myprog.cpp
```

or

```
$ gcc -o myprog myprog.cpp -lc++
```

Standard C++ library

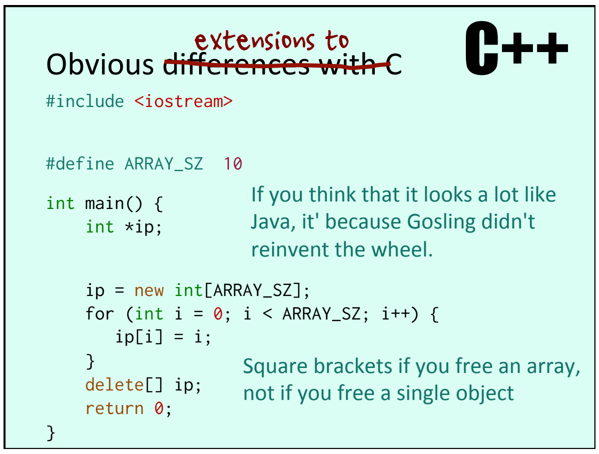


Obvious ~~differences with~~ extensions to C

C

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

#define ARRAY_SZ   10

int main() {
    int *ip;
    int  i;

    ip = (int *)malloc(sizeof(int)*ARRAY_SZ);
    for (i = 0; i < ARRAY_SZ; i++) {
       ip[i] = i;
    }
    free(ip);
    return 0;
}
```

Stroustrup also had understandable reservations about void * pointers, so he reviewed memory allocation/ deallocation.



Obvious ~~differences with~~ extensions to C

C++

```
#include <iostream>


#define ARRAY_SZ   10

int main() {
    int *ip;

    ip = new int[ARRAY_SZ];
    for (int i = 0; i < ARRAY_SZ; i++) {
       ip[i] = i;
    }
    delete[] ip;
    return 0;
}
```

If you think that it looks a lot like Java, it' because Gosling didn't reinvent the wheel.

Square brackets if you free an array, not if you free a single object

Obvious ~~differences with~~ extensions to C

Exceptions

```
  try {                        throw exception
     ...
  } catch (exception1) {
  ...
  } catch (exception2) {
  ...
  }
```

Tired with checking what every function returns?Use exceptions.

Obvious ~~differences with~~ extensions to C

Functions

Overloading

same name OK if parameters different

Optional parameters with default values

```
   int my_func(string p1, int p2=0)
```

Obvious ~~differences with~~ extensions to C

More questionable IMHO.

Functions

Automatic reference

still seen as an int

```
   int my_func(string p1, int& p2)

    int a;
    int b;

    b = my_func("Hello", a);
```

&a automatically passed

C

```
#include <stdio.h>

int main() {
    int val;

    printf("Enter a value : ");
    scanf("%d", val);
    printf("Value: %d\n", val);
    return 0;
}
```

This program crashes because scanf expects (after the format) addresses, not values.

```
$ ./scan_test
Enter a value : 3
Segmentation fault: 11
```

C++

```
#include <stdio.h>

int safe_scanf(int& value) {
    return scanf("%d", &value);
}

int main() {
    int val;

    printf("Enter a value : ");
    safe_scanf(val);
    printf("Value: %d\n", val);
    return 0;
}
$ ./scan_test2
Enter a value : 3
Value: 3
$
```

Here the address of `val` (in the main) is passed to `safe_scanf()` that passes it again to `scanf()`. Note that `&value` is still needed.

Obvious ~~differences with C~~ extensions to C

CLASSES

Classes are of course THE main addition to C.

C

```
#ifndef MATRICES_H
#define MATRICES_H // No value required

typedef struct matrix {
           short rows;
           short cols;
           double *cells;
        } MATRIX_T;
MATRIX_T *new_matrix(int rows, int cols);
void      free_matrix(MATRIX_T *m);
MATRIX_T *matrix_add(MATRIX_T *m1, MATRIX_T *m2);
MATRIX_T *matrix_scalar(MATRIX_T *m, double lambda);
MATRIX_T *matrix_mult(MATRIX_T *m1, MATRIX_T *m2);
MATRIX_T *matrix_inv(MATRIX_T *m);
double    matrix_det(MATRIX_T *m);
#endif // ifndef MATRICES_H
```

In C you usually write something like this.

C++

```
#ifndef MATRICES_HPP
#define MATRICES_HPP

struct matrix {
     short rows;
     short cols;
     double *cells;
     matrix *new_matrix(int r, int c);
     void    free_matrix();
     matrix *matrix_add(matrix *m);
     matrix *matrix_scalar(double lambda);
     matrix *matrix_mult(matrix *m);
     matrix *matrix_inv();
     double  matrix_det();
   };
#endif // ifndef MATRICES_HPP
```

In C++ you can have in a structure "member functions" (close to function pointers that are automatically initialized), also called "methods"

Also note that there is no more `typedef`; `matrix` = `struct matrix`

```
matrix *matrix::new_matrix(int r, int c) {


}
```

Required

When actually writing the function (in another file) you must prefix the function name by the struct name (spoiler: same thing with classes) because you can imagine different structure containing functions with identical names (eg "`length()`")

```
#ifndef MATRICES_HPP
#define MATRICES_HPP

struct matrix {
      short rows;
      short cols;
      double *cells;

      matrix *new_matrix(int r, int c);
      void    free_matrix();
      matrix *matrix_add(matrix *m);
      matrix *matrix_scalar(double lambda);
      matrix *matrix_mult(matrix *m);
      matrix *matrix_inv();
      double  matrix_det();
   };
#endif // ifndef MATRICES_HPP
```

But a structure doesn't offer all the privacy you may want.

Also accessible by other functions

We want ENCAPSULATION: only the methods can see the attributes.

```
#ifndef MATRICES_HPP
#define MATRICES_HPP

class  matrix {
      short rows;
      short cols;
      double *cells;
  public:
      matrix(int r, int c);
      ~matrix();
      matrix *matrix_add(matrix *m);
      matrix *matrix_scalar(double lambda);
      matrix *matrix_mult(matrix *m);
      matrix *matrix_inv();
      double  matrix_det();
   };
#endif // ifndef MATRICES_HPP
```

Turn a `struct` into a `class`, and everything that isn't declared as `public` is private.
If they exist, a constructor and a destructor (names derived from the class name) are automatically called when needed.

constructor

destructor

Contrary to Java, the code of functions isn't usually supplied in the class definition (it can be, but this is usually only done for very simple functions that need only a few lines of code).
Code is given in other files, and each function name is given as `<class name>::<function name>` as we have seen with the `struct` example.
The class really is an interface, the specification of how you interact with the object. The source code may not be supplied in a legible form but only as a .o file.

Obvious ~~differences with~~ extensions to C

Vectors

`#include <vector>`

```
vector<int> my_list;
```

any type

Finally, C++ also introduces some collections, such as a `vector` that is a built-in collection similar to a Java ArrayList.

```
my_list.size()
my_list.push_back(e)
my_list.insert(pos, e)
my_list[i]
```

Major differences with Java

Garbage Collector

If Java reused many good ideas from C++, it also introduced a new component that is missing from C++: the garbage collector. In Java, you can behave like a spoilt kid who knows that his or her mum will clean everything behind.

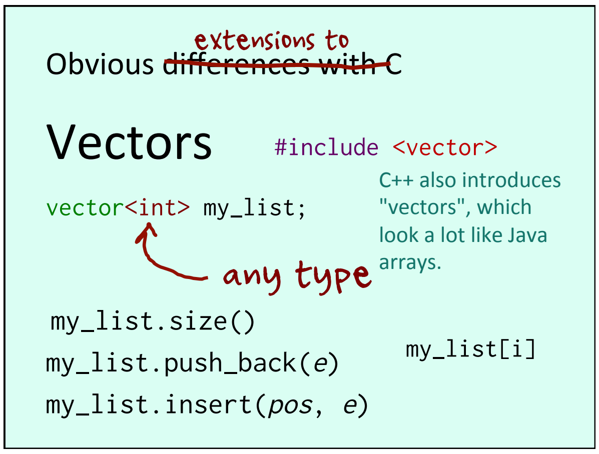
Major differences with Java

In Java, an object is always a reference (always created with new). Not in C++, where you can either declare an object or declare a pointer and allocate the object dynamically. There are two notations for accessing attributes/methods.

C++

```
ClassType  object;
ClassType *object;
```

.	With regular objects
->	With object pointers



Obvious ~~differences with C~~ extensions to

Vectors

```
#include <vector>
```

C++ also introduces "vectors", which look a lot like Java arrays.

```
vector<int> my_list;
```

any type

```
my_list.size()
my_list.push_back(e)
my_list.insert(pos, e)
```

```
my_list[i]
```