CSCI 334:
Principles of Programming Languages

Lecture 5: PL Fundamentals

Instructor: Dan Barowy

Williams

Announcements

Announcements

Google summer internships

Announcements

Google summer internships

HW2 grades out

Announcements

Google summer internships

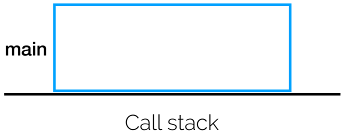
HW2 grades out

Project presentation date:
December 11; last day of reading period

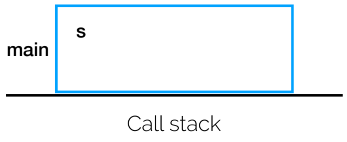
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.

Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.

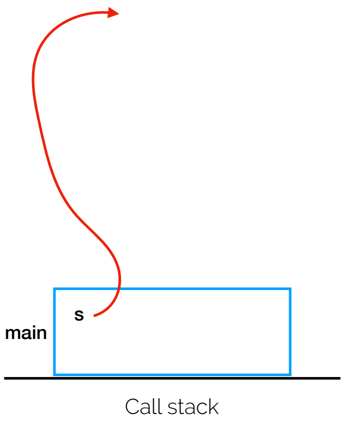
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



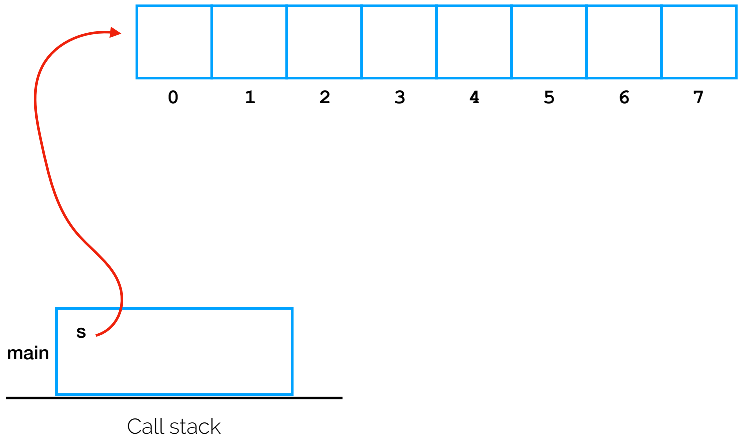
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



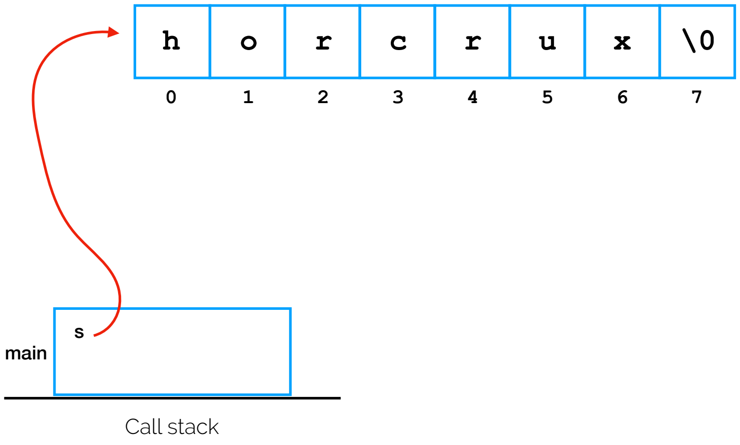
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



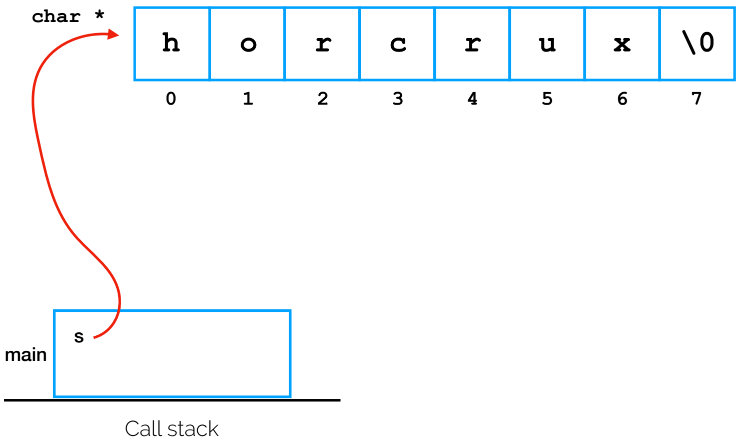
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



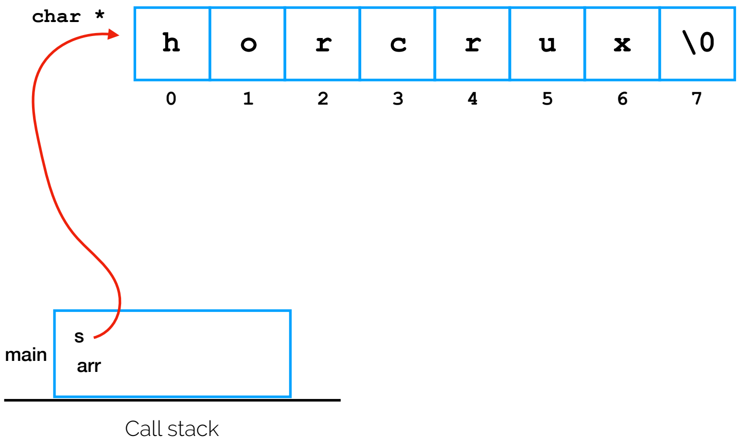
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



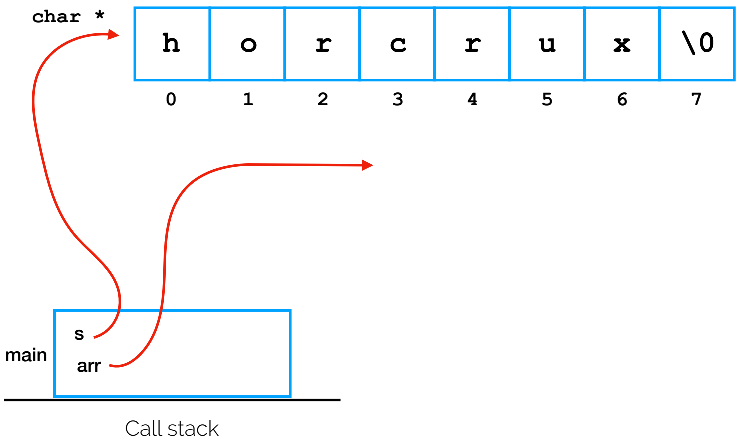
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



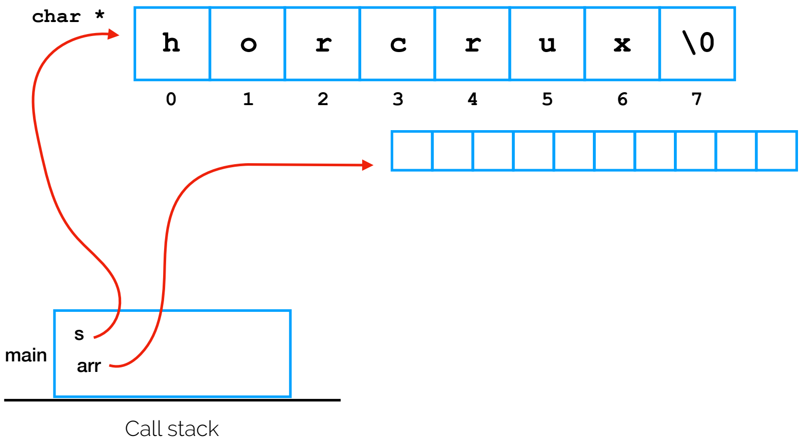
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



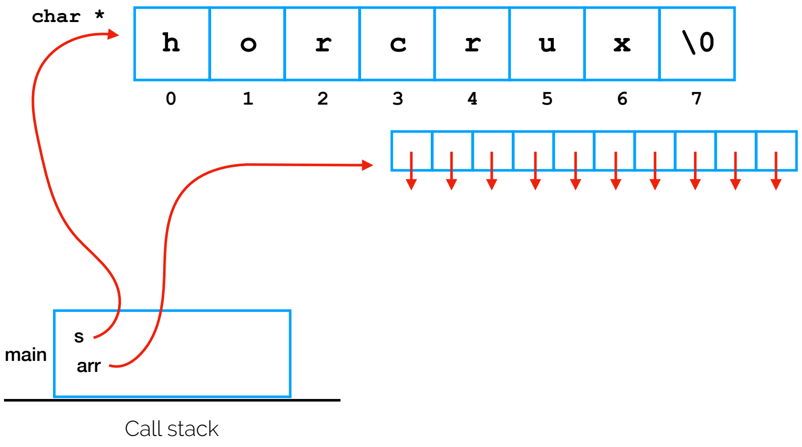
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



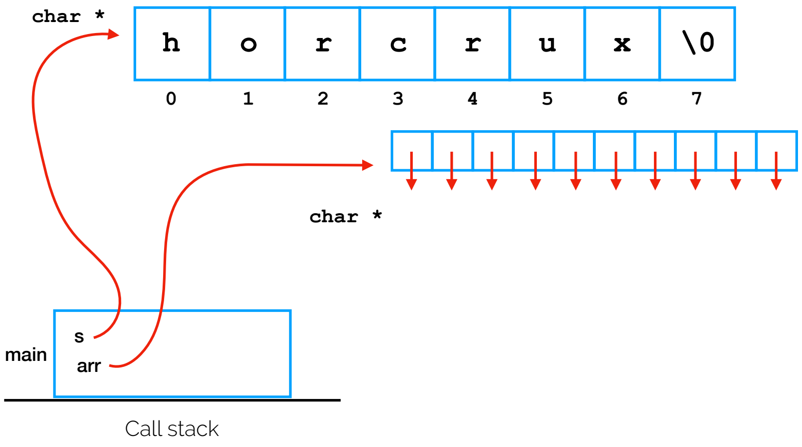
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



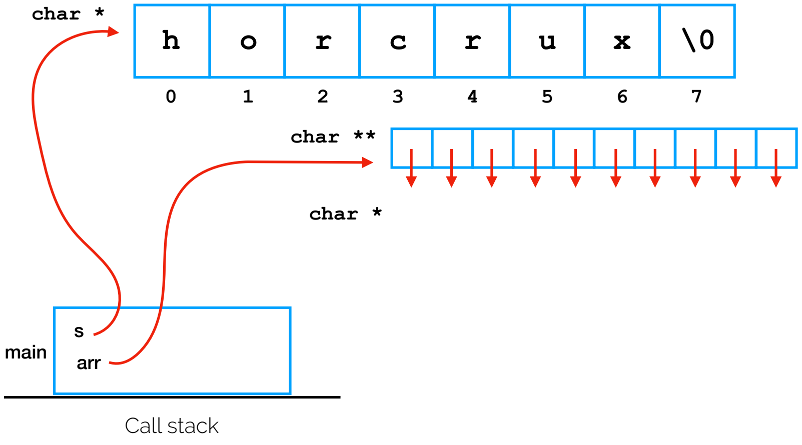
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.



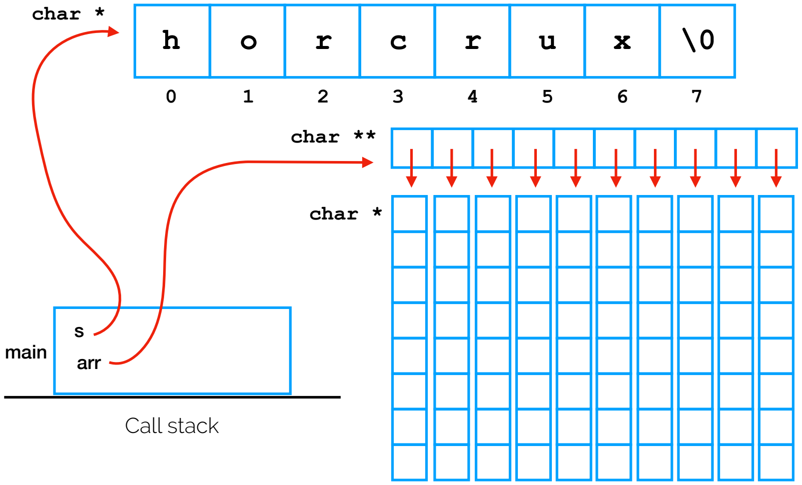
Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.

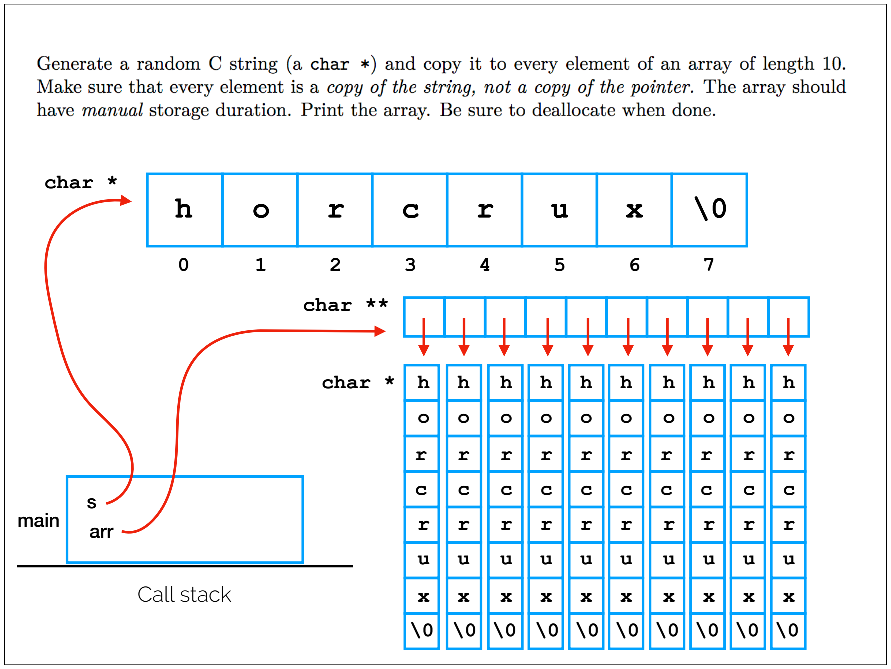


Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.

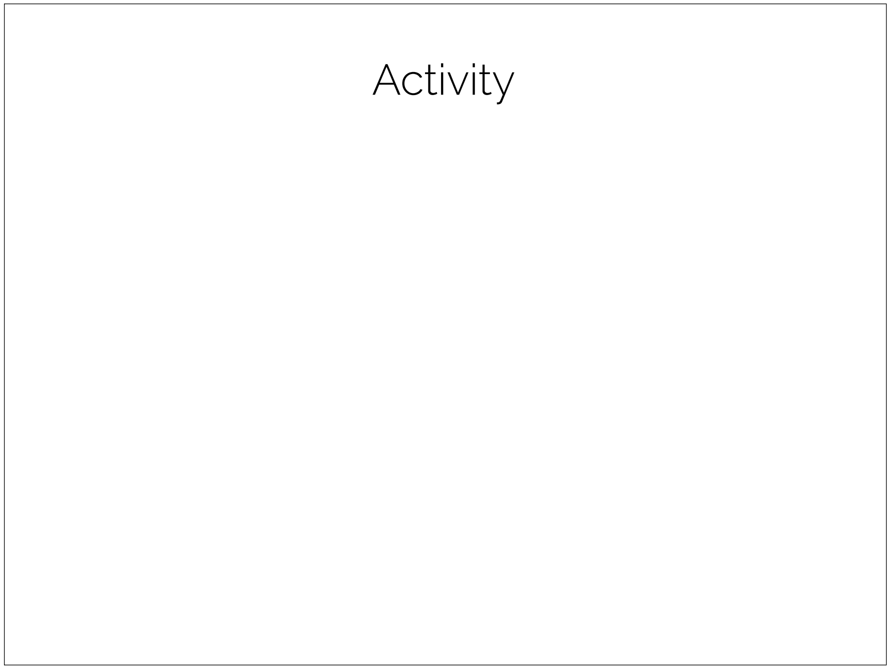


Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.

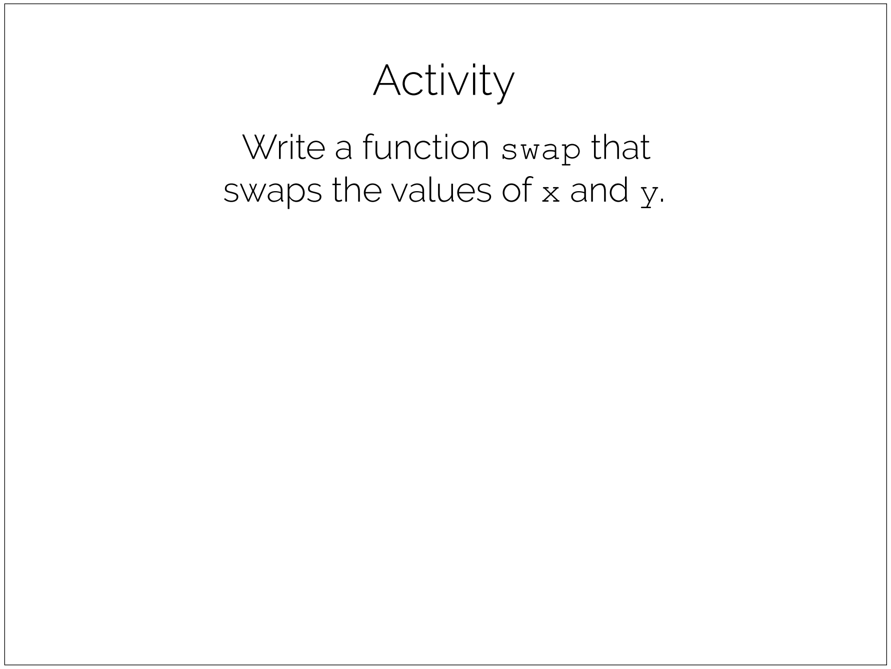




Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer*. The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.

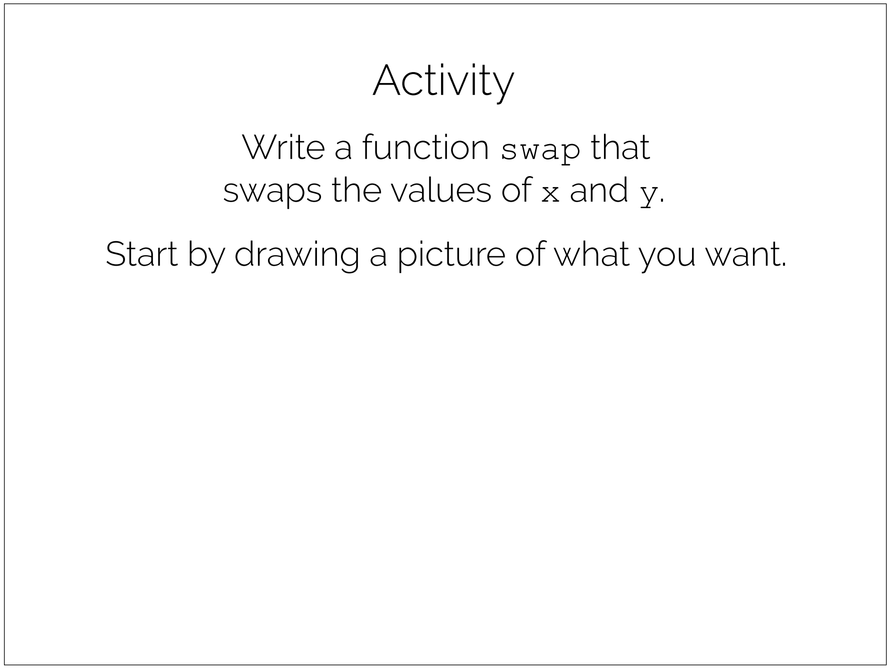


Activity



Activity

Write a function `swap` that swaps the values of `x` and `y`.



Activity

Write a function `swap` that swaps the values of `x` and `y`.

Start by drawing a picture of what you want.

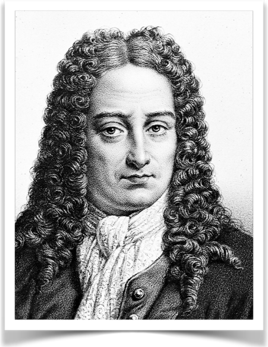
Activity

Write a function `swap` that
swaps the values of `x` and `y`.

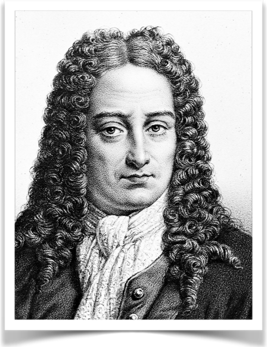
Start by drawing a picture of what you want.

```
#include <stdio.h>

void swap(int *a, int *b) {
  ???
}

int main() {
  int x = 1;
  int y = 2;
  swap(&x, &y);
  printf("x = %d, y = %d\n", x, y);
  return 0;
}
```

The Dream



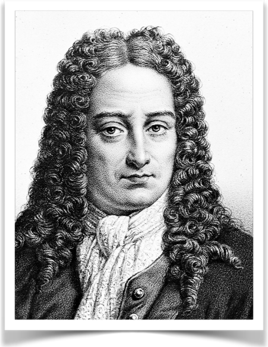
The Dream



Wilhelm Gottfried Leibniz

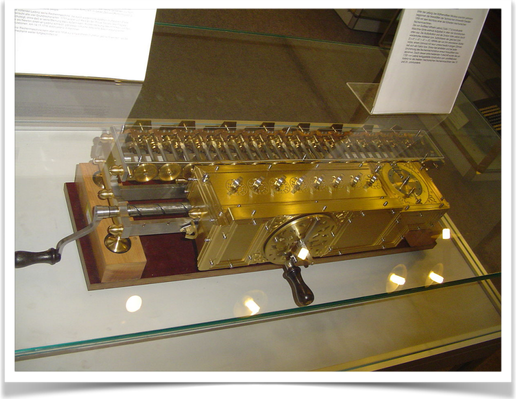
The Dream

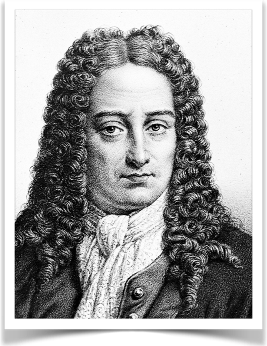
“I thought again about my early plan of a new language or writing-system of reason, which could serve as a communication tool for all different nations… If we had such an universal tool, we could discuss the problems of the metaphysical or the questions of ethics in the same way as the problems and questions of mathematics or geometry. That was my aim: Every misunderstanding should be nothing more than a miscalculation (…), easily corrected by the grammatical laws of that new language. Thus, in the case of a controversial discussion, two philosophers could sit down at a table and just calculating, like two mathematicians, they could say, 'Let us check it up …'”



Wilhelm Gottfried Leibniz

The Dream





Wilhelm Gottfried Leibniz

"What is the answer to the ultimate question of life, the universe, and everything?

What is computable?

What is computable?

- Hilbert: Is there an algorithm that can decide whether a logical statement is valid?

What is computable?

- Hilbert: Is there an algorithm that can decide whether a logical statement is valid?
- “Entscheidungsproblem” (literally “decision problem”)

What is computable?

- Hilbert: Is there an algorithm that can decide whether a logical statement is valid?
- “Entscheidungsproblem” (literally “decision problem”)
- Leibniz thought so!

What is computable?

What is computable?

- Why do we care?

What is computable?

- Why do we care?
- f(x) = x + 1

What is computable?

- Why do we care?
- f(x) = x + 1
- We can clearly do this with pencil and paper.

What is computable?

- Why do we care?
- f(x) = x + 1
- We can clearly do this with pencil and paper.
- $\int 6x\, dx$

What is computable?

- Why do we care?
- f(x) = x + 1
- We can clearly do this with pencil and paper.
- *∫ 6x dx*
- Also computable, in a different manner.

What is computable?

- Why do we care?
- f(x) = x + 1
- We can clearly do this with pencil and paper.
- *∫ 6x dx*
- Also computable, in a different manner.

- We care because the computable functions can be done on a computer.

Lambda calculus

Lambda calculus

- Invented by Alonzo Church in order to solve the Entscheidungsproblem.

Lambda calculus

- Invented by Alonzo Church in order to solve the Entscheidungsproblem.
- Short answer to Hilbert's question: no.

Lambda calculus

- Invented by Alonzo Church in order to solve the Entscheidungsproblem.
- Short answer to Hilbert's question: no.
- Proof: No algorithm can decide equivalence of two arbitrary **λ**-calculus expressions.

Lambda calculus

- Invented by Alonzo Church in order to solve the Entscheidungsproblem.
- Short answer to Hilbert's question: no.
- Proof: No algorithm can decide equivalence of two arbitrary **λ**-calculus expressions.
- By implication: no algorithm can determine whether an arbitrary logical statement is valid.