C++ function overloading

```
#include<iostream>
const float PI = 3.14;
double area(double length)      // Area of square
{
        return length * length;
}
double area(double length, double width)    // Area of rectangle
{
        return length * width;
}
double area(int radius)      //Area of circle
{
        return PI * radius * radius;
}
int main()
{
        std::cout << area(2) << std::endl;      // 12.56
        std::cout << area(2.0) << std::endl;    // 4
        std::cout << area(1.1, 2) << std::endl; // 2.2
        return 0;
}
```

C++ function overloading

- Function overloading allows us to use same function name for multiple function definitions
  - Only related functions should be overloaed
- All the overloads of a function must have unique signature
  - Signature of the function includes **number, type** and **order** of parameters
    - Functions with same name but different number of arguments are valid overloads
    - Functions with same name and same number of arguments, but different types are valid overloads
      - int area(int x, float y);
      - Int area(int x, double y);
    - Functions with same name and same number and type of arguments, but different order of arguments are valid overloads
      - int area(float y, int x);
      - Int area(int x, float y);
  - Signature of the function does not include return type
    - Followig two are not valid overloads, it will result in error
      - int area(int x, float y);
      - float area(int x, float y);

C++ function overloading

- Function selection (a.k.a overload resolution) is done by compiler during compilation
  - For each function call compiler decides which overload of the function to call based on actual arguments passed during function call
- It follows following rules for each function call:
  1. Prepare list of **candidate functions** (All overloads with same function name)
  2. Select **viable functions** from candidate functions (based on # of arguments)
  3. One function from viable functions is called based on following rules
     I. Exact match
     II. Promotions
        - e.g. integral conversions - char, bool, enum to int,
        - e.g. float promotions - float to double
     III. Standard type conversions
        - e.g. float to double, int to long, int to float
     IV. User-defined conversions (related to classes, ignore for now)

C++ function overloading

I. Exact match
II. Promotions
    - e.g. integral conversions - char, bool, enum to int,
    - e.g. float promotions - float to double
III. Standard type conversions
    - e.g. float to double, int to long, int to float
IV. User-defined conversions (related to classes, ignore for now)

- During any of the above four steps, if
    - Only one function could be selected – then that function is called – overload resolution stops
    - More than one function could be selected – ambigous call – compilation error
    - No function can be selected – overload resolution continues to next step

- If no function could be called after step IV, then it results in compilation error

C++ function overloading

A) double area(int r);
B) double area(int l, double w = 10.0);

Sr. No.	Function call	Viable functions	Exact match	Prom-otions	Std. type conversions	Remark
1	area(10, 10.0);	B	B			Calls B (returns 100.0)
2	area(10);	A, B	A, B			Ambiguous call
3	area(true, 10.0);	B	-	B		Calls B (returns 10.0)
4	area('A');	A, B	-	A, B		Ambiguous call
5	area(true, 10);	B	-	-	B	Calls B (returns 10.0)
6	area(10.0);	A, B	-	-	A, B	Ambiguous call
7	int i; area(&i);	A, B	-	-	-	No matching f^n

C++ function overloading

```
#include<iostream>
#define PI 3.14

double area(int r)
{
	std::cout << "A" << std::endl;
	return PI * r * r;
}

double area(int l, double w = 10.0)
{
	std::cout << "B" << std::endl;
	return l * w;
}
```

```
int main()
{
	std::cout << area(10, 10.0) << std::endl;
	//std::cout << area(10) << std::endl;

	std::cout << area(true, 10.0) << std::endl;
	//std::cout << area('A') << std::endl;

	std::cout << area(true, 10) << std::endl;
	//std::cout << area(10.0) << std::endl;
	//int i; std::cout << area(&i) << std::endl;

	return 0;
}
```

Interesting reads

- Default arguments in function definition Vs function prototype
  - https://stackoverflow.com/questions/4989483/where-to-put-default-parameter-value-in-c
- C++ overload resolution
  - https://en.cppreference.com/w/cpp/language/overload_resolution