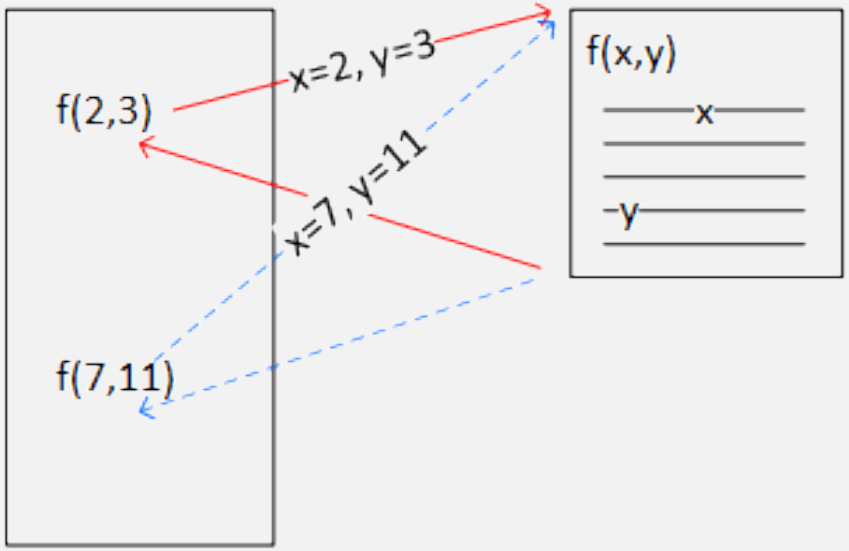
INTRODUCTION TO TEMPLATES

Variables generalize data values

Templates generalize data types

Delroy A. Brinkerhoff

FUNCTION PARAMETERS ARE PLACEHOLDERS



- Functions define reusable operations
- Parameters are placeholders for function data
  - Programs pass data to function parameters
  - Functions use parameters wherever they need the corresponding data
  - Parameters can have different data each time the program calls the function

SAME OPERATION
DIFFERENT DATA TYPES

C

- `double sqrt(double x);`
- `float sqrtf(float x);`
- `long double sqrtl(long double x);`

C++

- `double sqrt(double x);`
- `float sqrt(float x);`
- `long double sqrt(long double x);`
- `double sqrt(T x);`
  - `T is a template variable matching any integral type`

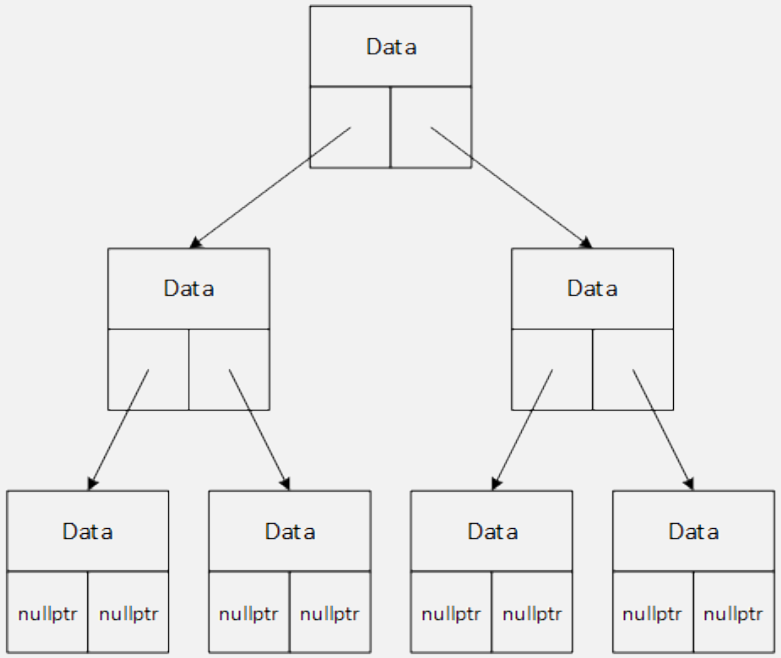
WHEN OVERLOADING DOESN'T WORK, USE TEMPLATES

- `int f(int a);`
- `double f(double a);`
- `char f(char a);`

- `Person f(Person a);`
- `Shape f(Shape a);`

```
template <typename T>
T f(T p)
{
        T t1 = p;
        T t2 = …;
        …
        return t2;
}
```

GENERALIZED TEMPLATE CLASSES




```
template <class T>
class BTree
{
    private:
        T data;
    public:
        void insert(T x);
        T search(T key);
};

template <class T>
void insert(T x) { ... }

template <class T>
T search(T key) { ... }
```