CP::stack

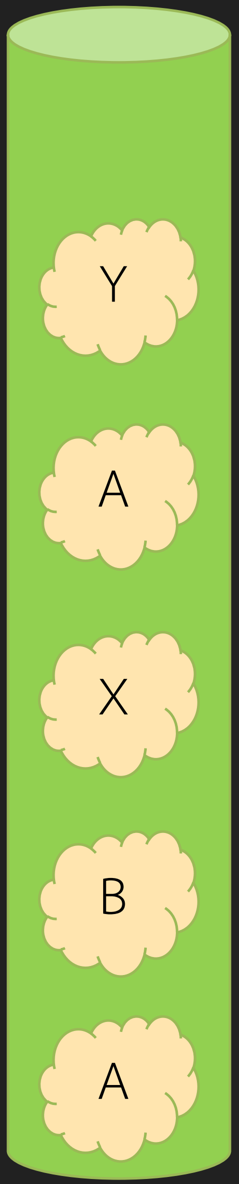
Nattee Niparnan

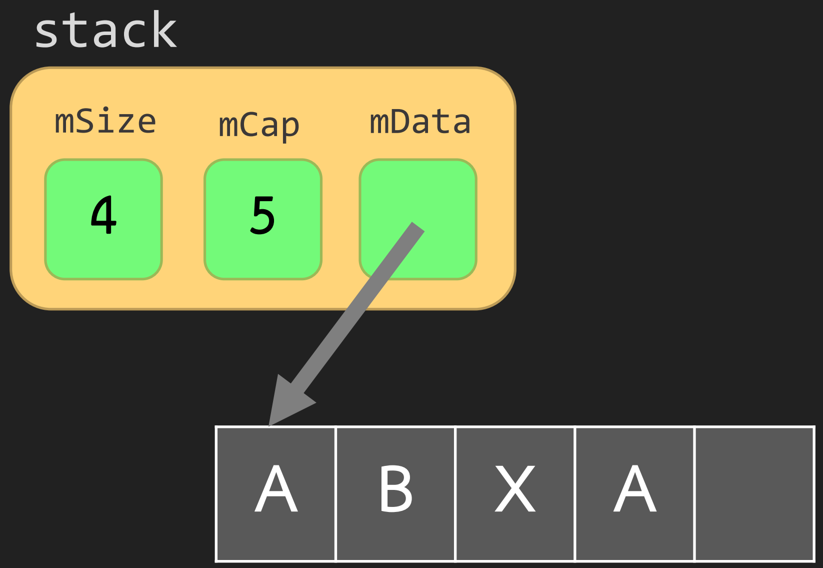
Intro

- Now we will create less complex data structure CP::stack
- Just like a vector without iterator, insert, erase, resize, at and operator[ ]
  - Add top() which is just a shorthand of looking at the last element
- That’s it, really

Key Idea

- The data is stored in the same way as a vector
  - The first element of mData is the bottom of stack while the last element is the top of stack
- We just take vector.h and remove unnecessary function





stack.h

```cpp
namespace CP {
  template <typename T>
  class stack
  {
    protected:
      T *mData;
      size_t mCap;
      size_t mSize;
      void expand(size_t capacity)  {...}
      void ensureCapacity(size_t capacity)  {...}
    public:
      //------------- constructor ----------
      stack(const stack<T>& a) {...}
      stack()  {...}
      stack<T>& operator= {...}
      ~stack()  {...}
      //-------- capacity function ----------
      bool empty() const {...}
      size_t size() const {...}
      //------------- access ---------------
      const T& top() const {...}
      //------------- modifier -------------
      void push(const T& element) {...}
      void pop() {...}
  };
}
```

Same as vector

```cpp
const T& top() const{
  return mData[mSize-1];
}
```

This is push_back

This is pop_back

Speed of each operation

- All read operation always take constant time
  - size(), top() simply return something that is directly accessible
- All modify operation also take constant time
  - push() is constant on average (same as push_back of vector)
  - pop() is always constant

Stack By Vector

- Instead of writing our own function, there is another way to write a stack
- We simply use vector as our sole data member
- Benefit: code reuse
- Drawback: almost none except that we need one more layer of function call

```
namespace CP {
  template <typename T>
  class stack
  {
    protected:
      vector<T> v;
    public:
      // default constructor
      stack() : v() { }
      //------------- capacity function -------------------
      bool empty() const           {    return v.empty(); }
      size_t size() const          {     return v.size(); }
      //----------------- access --------------------------
      const T& top() const        { return v[v.size()-1]; }
      //----------------- modifier ------------------------
      void push(const T& element) { v.push_back(element); }
      void pop()                          { v.pop_back(); }
  };
}
```