CS 211 : Thurs 02/08 (lecture 11)

- **Topics**: vectors, parameter passing, memory management, RAII

Prof. Hummel (he/him)

February 2024

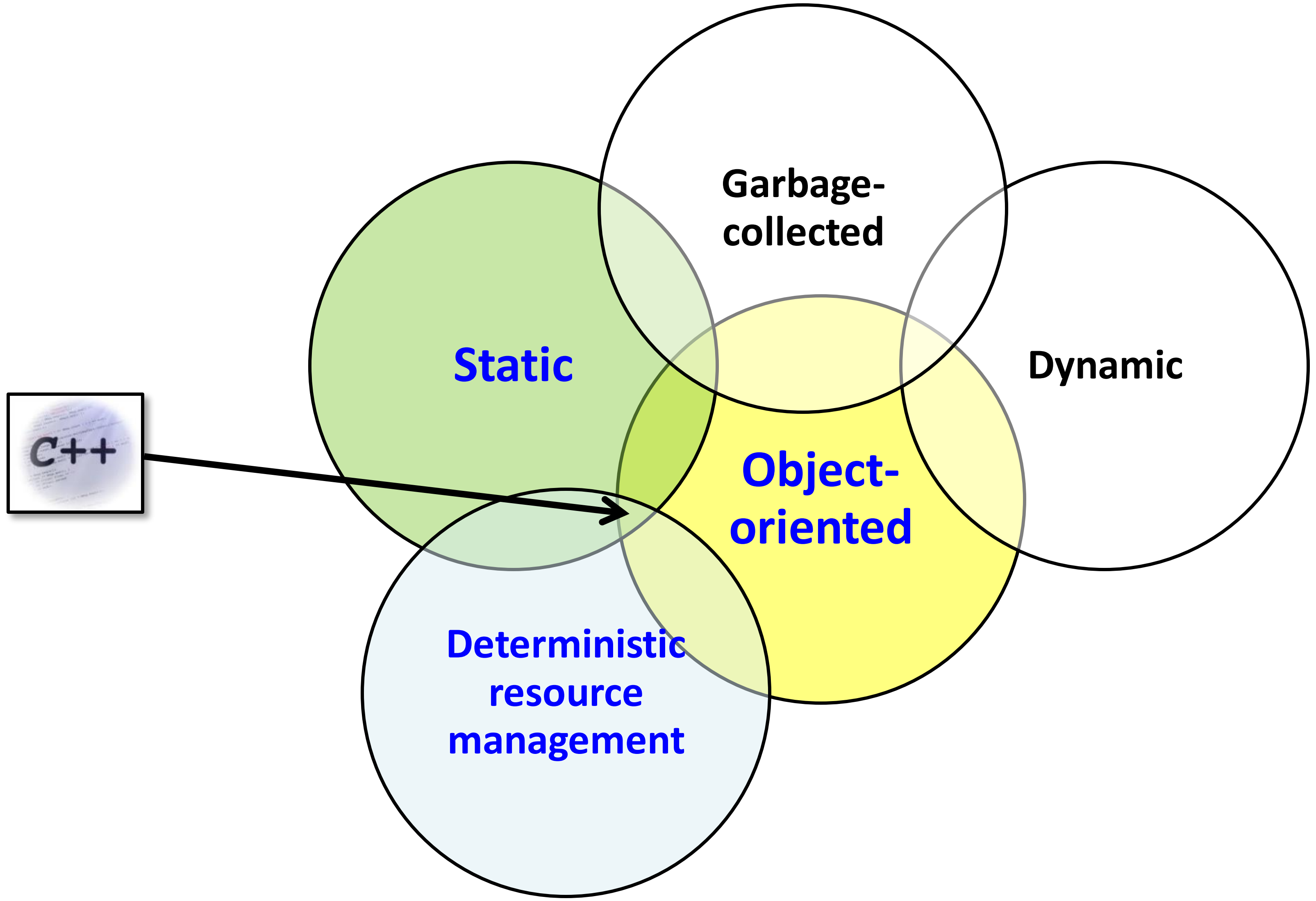
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

www.a-printable-calendar.com

Notes:

- *Lecture slides available on Canvas*
- ***Extra-credit C project** was released, due Sunday night (no late period)*
- ***HW 05** (intro to C++) due Tuesday 2/13*
- ***Project 05** (first C++ project) will be released tonight, due next Friday 2/16*

- **C++ is a unique language in today's landscape**



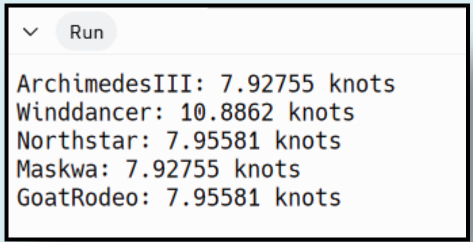
Example from last time

```
class Sailboat
{
public:
   string Name;
   double LengthOverall;
   double LengthWaterline;

   Sailboat(string name,
            double length,
            double lwl);

   double maxSpeedKts();
   double maxSpeedMPH();
};
```

```
int main()
{
  vector<Sailboat> boats;

  boats = readBoats("boats.txt");

  for (Sailboat s : boats) {
    cout << s.Name << ":  "
      << s.maxSpeedKts() << " kts"
      << endl;
  }

  return 0;
}
```

```
ArchimedesIII 37.73 35.00
Winddancer 72.00 66.00
Northstar 35.76 35.25
Maskwa 37.73 35.00
GoatRodeo 35.76 35.25
```

$\xrightarrow{}$ program $\xrightarrow{}$



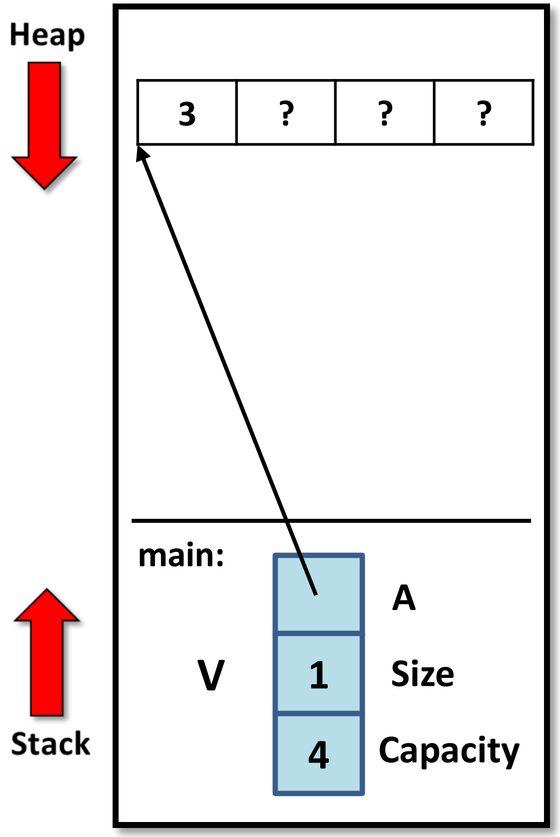
```
ArchimedesIII: 7.92755 knots
Winddancer: 10.8862 knots
Northstar: 7.95581 knots
Maskwa: 7.92755 knots
GoatRodeo: 7.95581 knots
```

vector<T>

- **A vector is a dynamic array**
  - *Array lives in the heap and doubles in capacity as needed*

```
template <typename T>
class vector
{
private:
   T*   A;          // ptr to underlying array
   int Size;        // # of elements currently
   int Capacity; // total # of locations

public:
   .
   .
   .
```

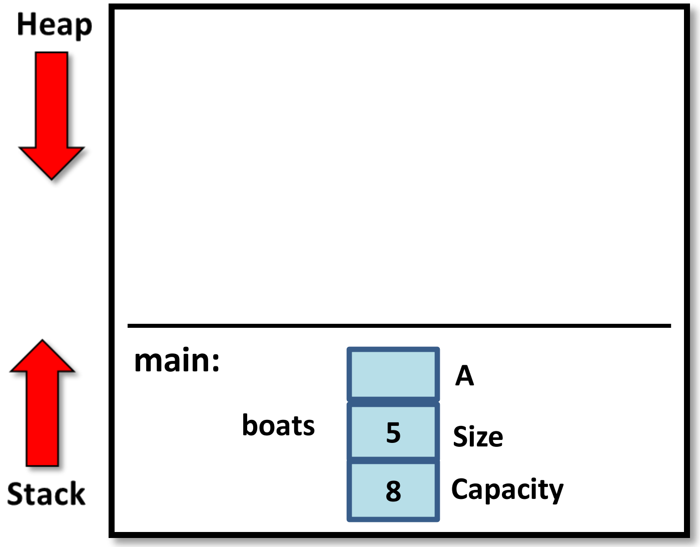
```
int main()
{
   vector<int>  V;

   V.push_back(3);
```



Pointers or no pointers…

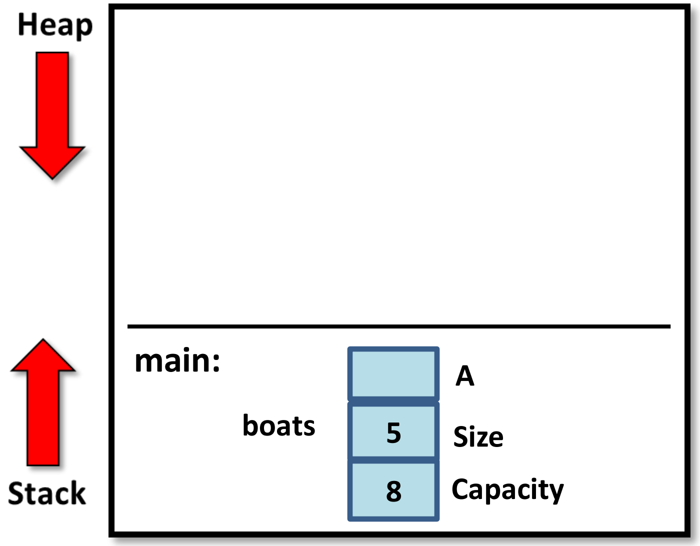
- **In C++ you have a choice of how objects are stored**
  - *Directly*

```
int main()
{
  vector<Sailboat> boats;

  boats = readBoats("boats.txt");
```



  - *Indirectly using pointers*

```
int main()
{
  vector<Sailboat*> boats;

  boats = readBoats("boats.txt");
```



Question

- **Consider this code fragment**

```
int main()
{
  Sailboat  s1("A3", 37.25, 36.0);

  Sailboat* s2 = new Sailboat("GR", 36.60, 35.80);
```

```
class Sailboat
{
public:
  string Name;
  double LengthOverall;
  double LengthWaterline;

  Sailboat(string name,
           double length,
           double lwl);

  double maxSpeedKts();
  double maxSpeedMPH();
};
```

A) Creating s1 is faster and uses less memory than s2

B) Creating s2 is faster and uses less memory than s1

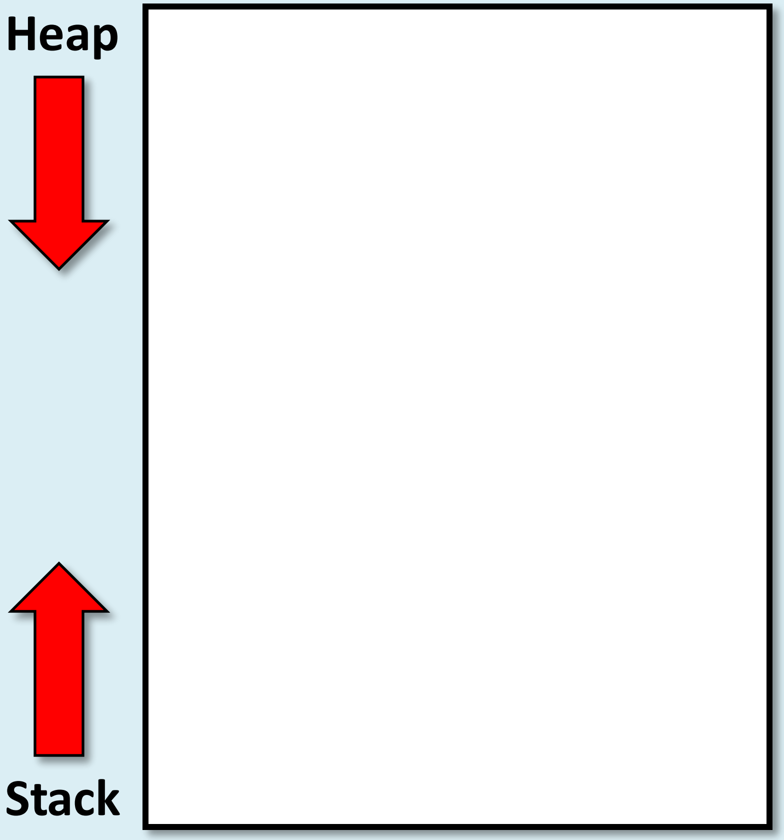
C) These are equivalent because pointers are used internally in the creation of s1

D) The code fragment is invalid because s1 and s2 do not match the constructor

Discussion

```
class Sailboat
{
public:
  string Name;
  double LengthOverall;
  double LengthWaterline;

  Sailboat(string name,
           double length,
           double lwl);

  double maxSpeedKts();
  double maxSpeedMPH();
};
```

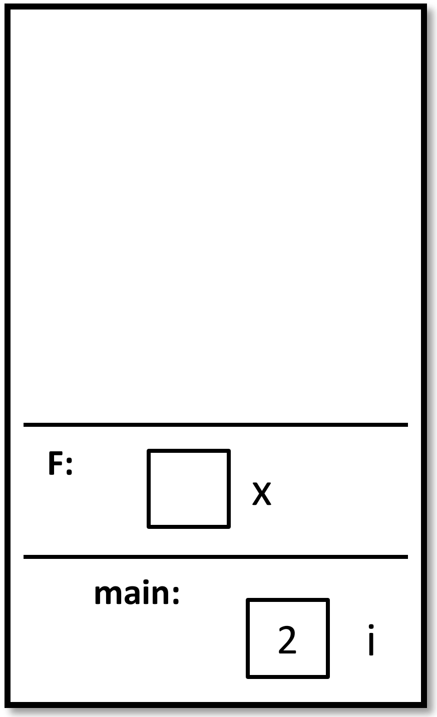
```
int main()
{
  Sailboat  s1("A3", 37.25, 36.0);

  Sailboat* s2 = new Sailboat("GR", 36.60, 35.80);
```



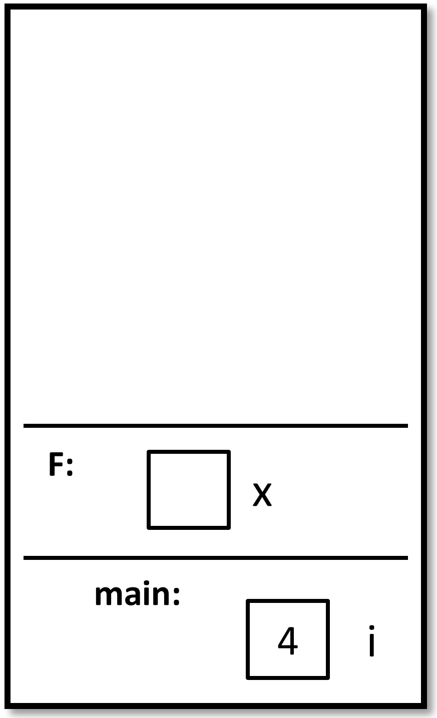
Parameter passing

- **C++ offers two parameter-passing mechanisms:**
  - ***Pass-by-value***: *C++ makes a copy. This is the default.*
  - ***Pass-by-reference (&)***: *C++ uses pointers, no copies.*

```
void F(int x)
{
   // stop here
   x++;
}

int main()
{
   int i = 2;

   F(i);
```

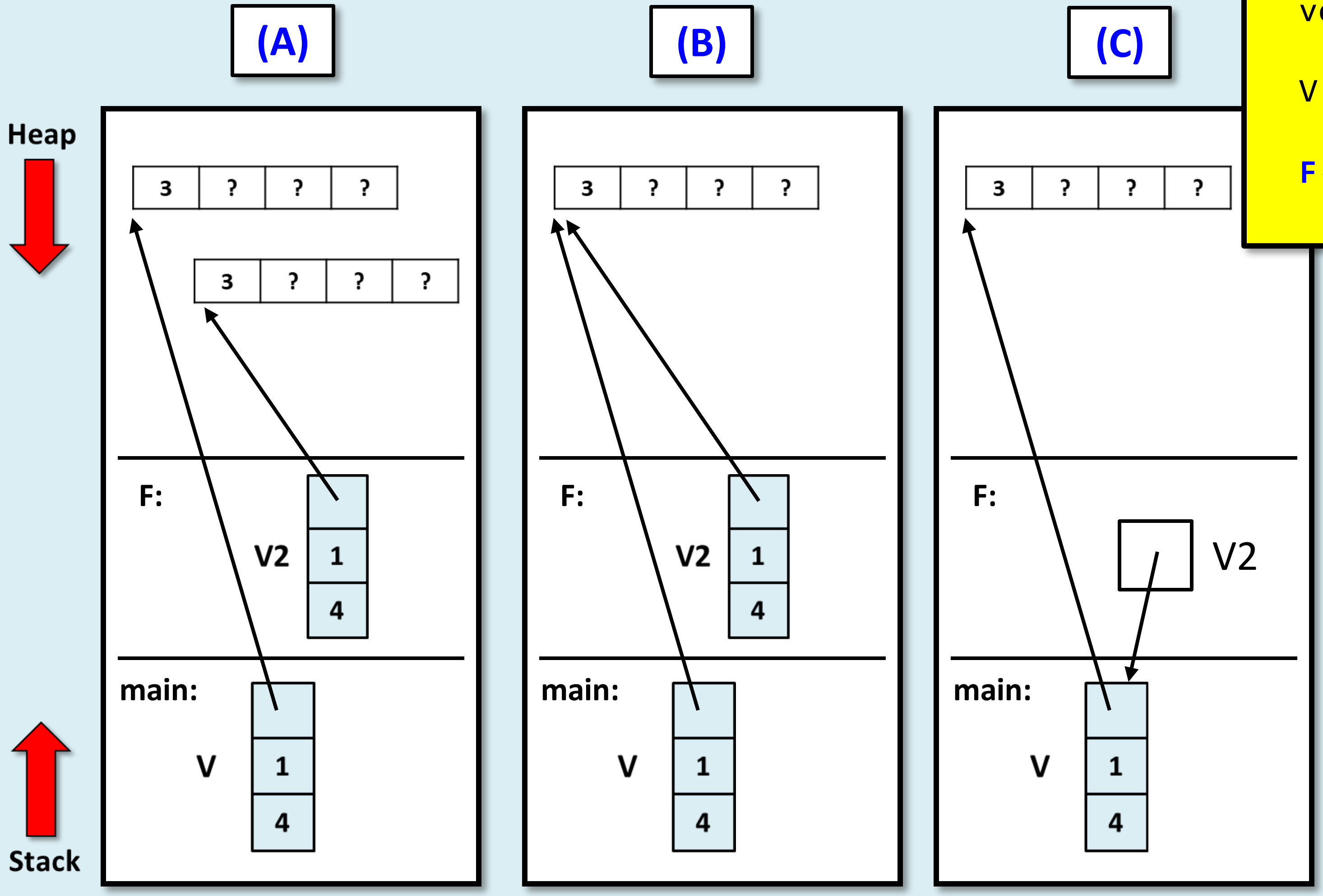


```
void F(int& x)
{
   // stop here
   x++;
}

int main()
{
   int i = 4;

   F(i);
```



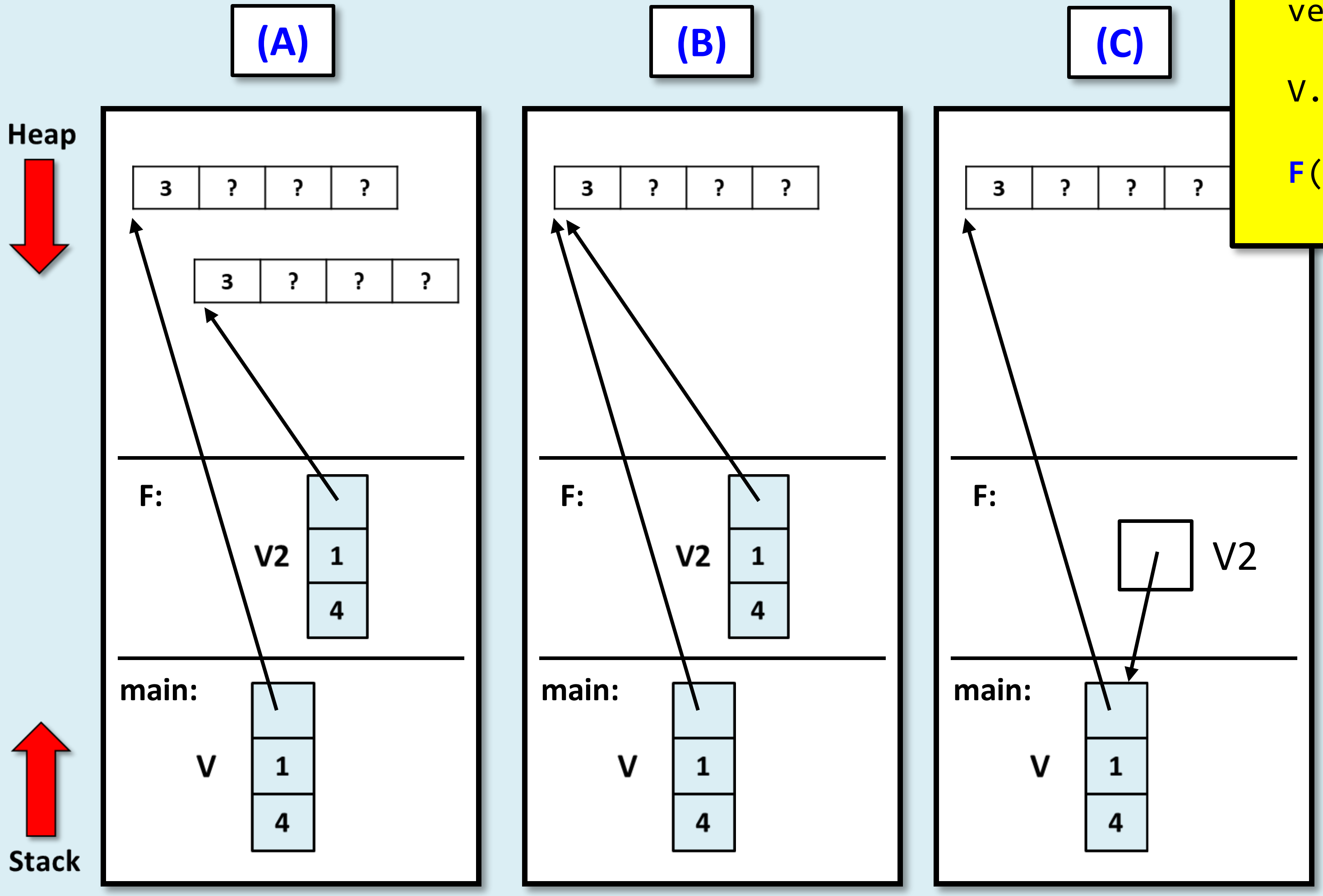
(1) So what happens when you pass a vector by value?

```
void F(vector<int> V2)
{
  // STOP HERE
  V2.push_back(5);
}

int main()
{
  vector<int>  V;

  V.push_back(3);

  F(V);
```

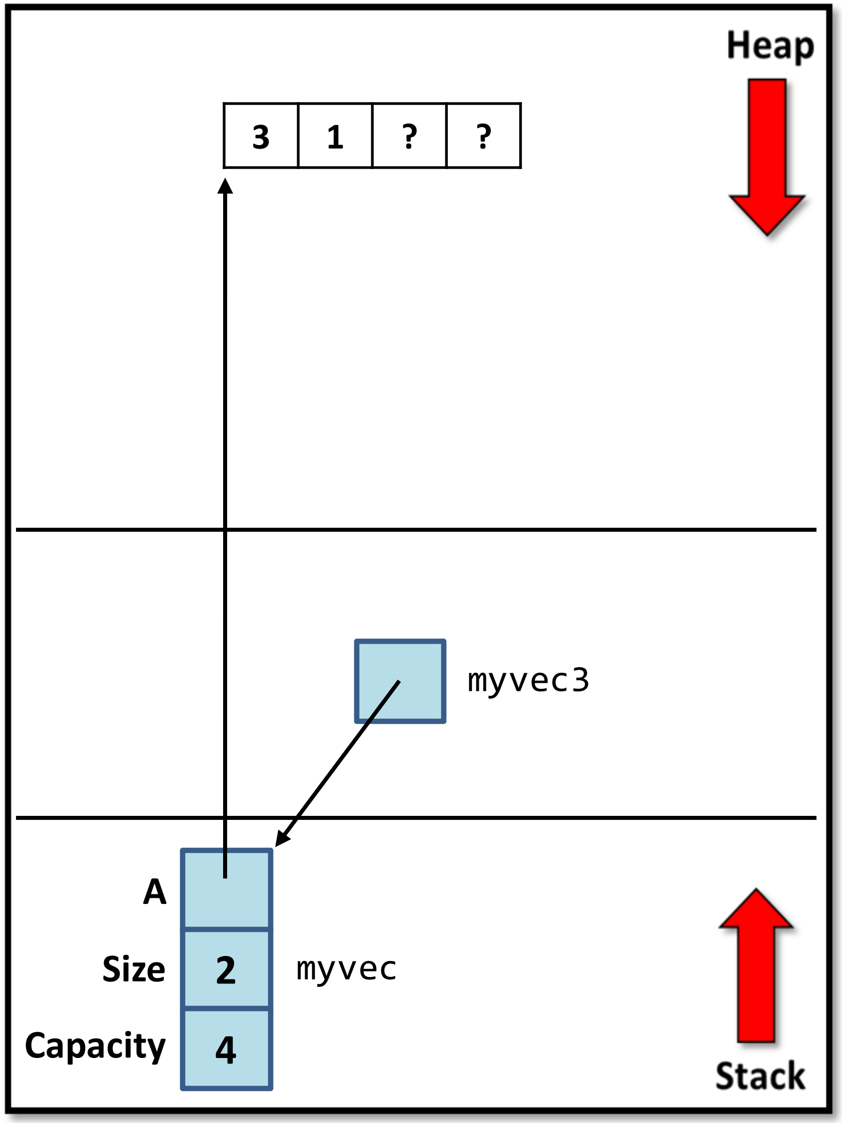


(2) What happens when you pass a vector by reference?

```
void F(vector<int>& V2)
{
  // STOP HERE
  V2.push_back(5);
}

int main()
{
  vector<int>  V;

  V.push_back(3);

  F(V);
```



Vectors : pass-by-reference

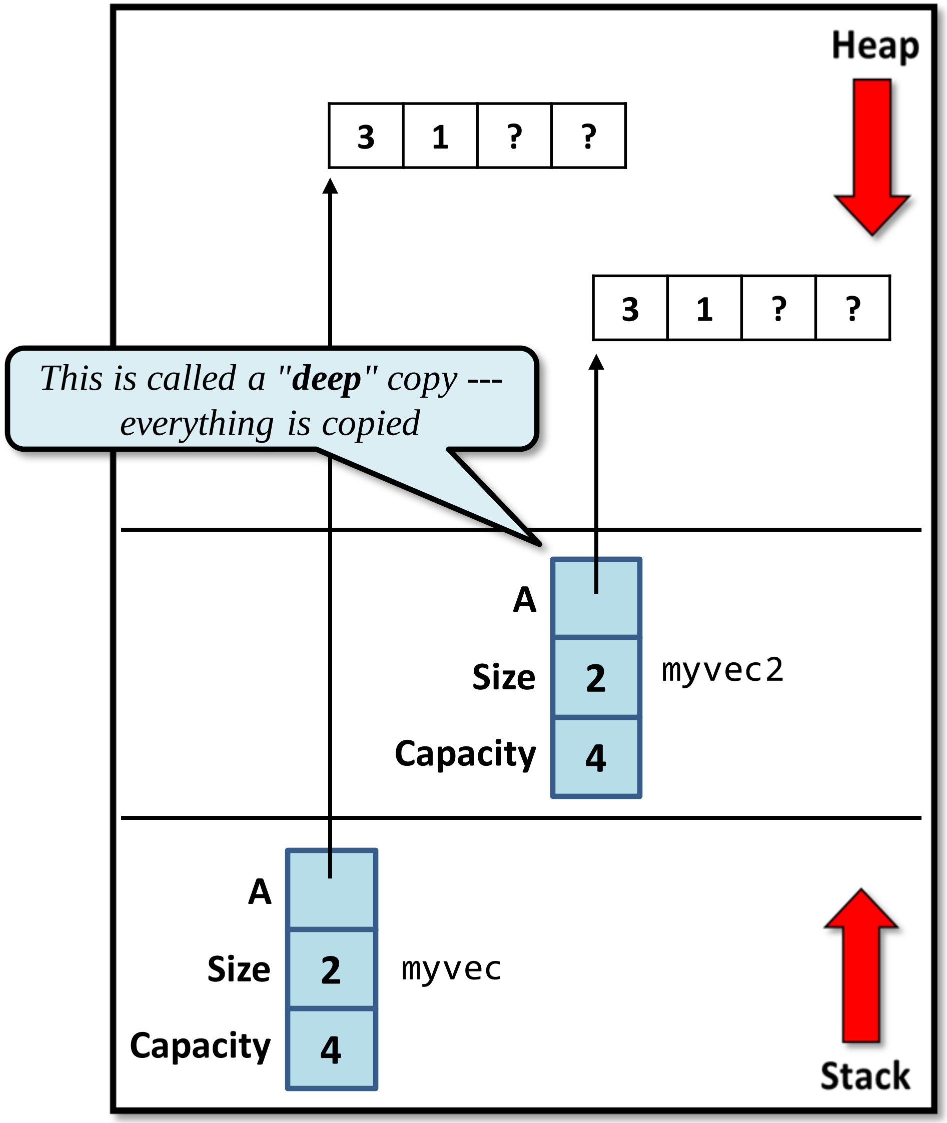


```
vector<int> myvec3 = &myvec;
```

```
void G(vector<int>& myvec3)
{
   myvec3.push_back(7);
   ...
}
```

```
int main()
{
  vector<int> myvec = {3, 1};

  G(myvec);
```

Vectors : pass-by-value



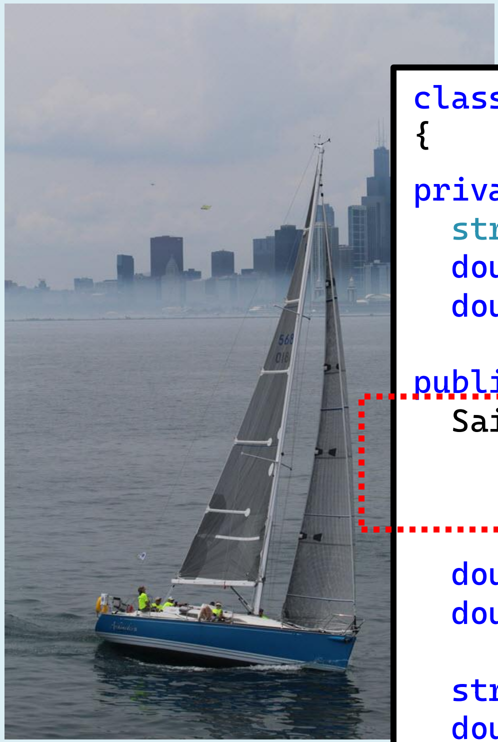
```
vector<int> myvec2 = myvec;
```

```
void F(vector<int> myvec2)
{
  myvec2.push_back(7);
  ...
}
```

```
int main()
{
  vector<int> myvec = {3, 1};

  F(myvec);
```

Constructors the C++ way

- We should be writing constructors the "C++" way



```
class Sailboat
{
private:
  string Name;
  double LengthOverall;
  double LengthWaterline;

public:
  Sailboat(string name,
           double length,
           double lwl);

  double maxSpeed();
  double maxSpeedMPH();

  string getName();
  double getLength();
  double getLengthWaterline();

};
```

```
Sailboat::Sailboat(string name,
                   double length,
                   double lwl)
{
  //
  // The "typical" way:
  //
  this->Name = name;
  this->LengthOverall = length;
  this->LengthWaterline = lwl;
}
```

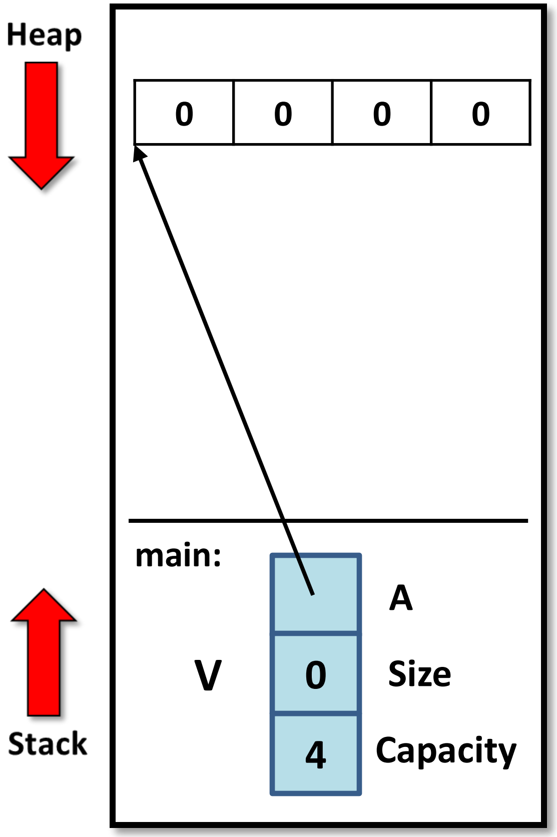
```
//
// The C++ way: initializer list
//
Sailboat::Sailboat(string name,
                   double length,
                   double lwl)
  : Name(name),
    LengthOverall(length),
    LengthWaterline(lwl)
{ }
```

Allocating memory the C++ way

- **Vector needs to allocate an initial array**
  - *malloc( ) just allocates…*
  - ***new** allocates *and* initializes each element via T's constructor*

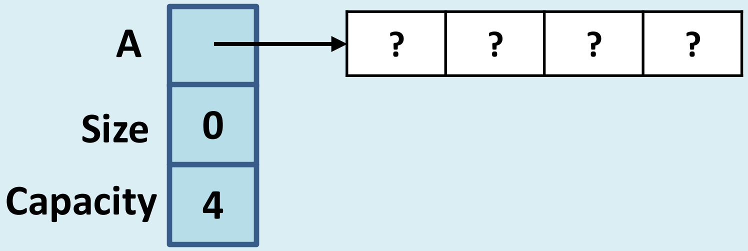
```
int main()
{
  vector<int>  V;
```

```
template <typename T>
class vector
{
private:
   T*  A;          // ptr to underlying array
   int Size;       // # of elements stored in array
   int Capacity;   // # of locations in array

public:
   vector()
     : A(new T[4]), Size(0), Capacity(4)
   { }
```



3) How does memory get freed? Enter C++ destructor…

```
template <typename T>
class vector
{
private:
  T*  A;          // ptr to underlying array
  int Size;       // # of elements stored in array
  int Capacity; // # of locations in array

public:
  vector()
    : A(new T[4]), Size(0), Capacity(4)
  { }

  ~vector()
  {
     // ???
  }
  .
  .
  .
```



(A) `free(A);`

(B) `delete T[4];`

(C) `delete[] A;`

(D) `~A();`

4) Consider this code… When is V2's destructor called?

```
int main()
{
  vector<int>  V1;

  V1.push_back(123);
  V1.push_back(88);
  V1.push_back(456);
  V1.push_back(42);

  if (V1.size() > 0)
  {
     vector<int>  V2;

     for (int x : V1)
       if (x > 100)
         V2.push_back(x);

     cout << V2.size() << endl;
  }//(1)

  return 0;  //(2)
}//(3)
```

A) @ (1)

B) @ (2)

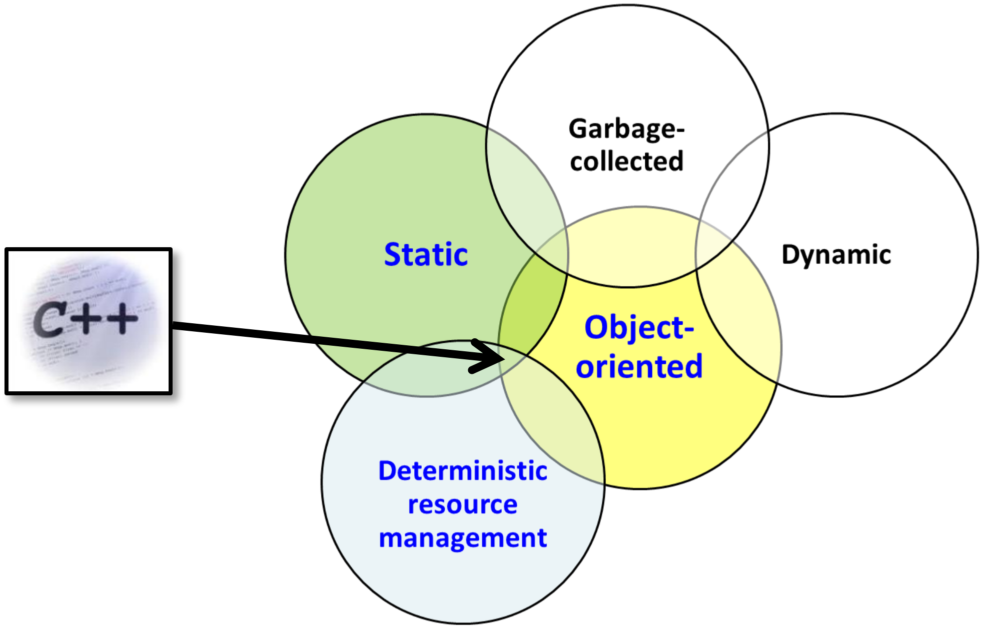
C) @ (3)

D) Never, the program ends without freeing the memory associated with V2

- **In C++, you know exactly when vector is going to be destroyed**

  ==> ***deterministic***

```
void F()
{
   vector<int> V;

   if (…)
     return;

   while(…)
   {
     …
   }
}

int main()
{
   F();
```



- **Contrast this with other modern languages — Java, C#, Python, JavaScript, …**
  - *They rely upon* ***garbage collection*** *to free resources*
  - ***Non-deterministic*** — *you don't know when it will happen*

RAII: Resource Acquisition Is Initialization

– *RAII is one of the major design rules in C++*

– *Classes designed to manage "resources" for you*

  - Object acquires resource during initialization --- i.e. constructor
  - Object releases resource when done --- i.e. destructor

```
void ReadFile(string filename)
{
   ifstream  file(filename);
   .
   .
   .
}

int main()
{
   ReadFile("data.csv");
```

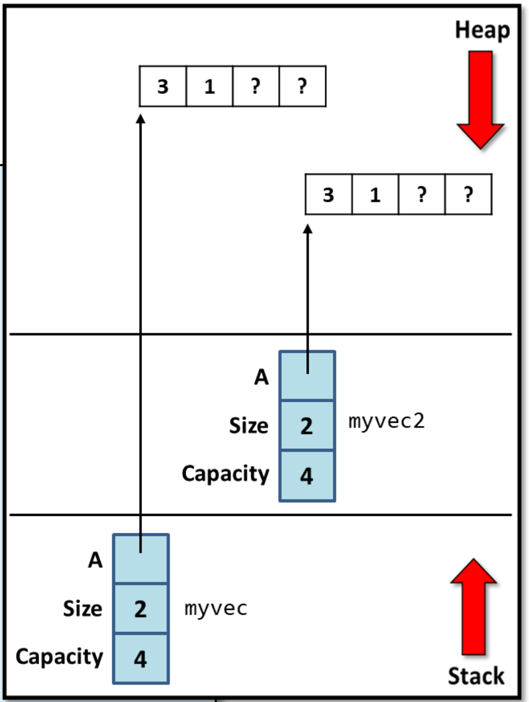
```
class ifstream  // input file
{
private:
  FILE *file;

public:
  ifstream(string filename) {
     file = fopen(filename, "r");
  }

  ~ifstream() {
     if (file != nullptr)
       fclose(file);
  }
```

- C++ seems like a really good language…
- Why does C++ have a reputation of being too complex?

Example: copy constructors

```
template <typename T>
class vector
{
private:
  T*  A;          // ptr to underlying array
  int Size;       // # of elements currently in A
  int Capacity; // total # of locations in A

public:

vector()
  : A(new T[4]), Size(0), Capacity(4);
{ }

~vector()
{ delete[] A; }
```

```
vector(const Vector& other)  // copy constructor
  : A(new T[other.Capacity]),
    Size(other.Size),
    Capacity(other.Capacity)
{
  for (int i=0; i<other.Size; ++i)  // copy elems
    this->A[i] = other.A[i];
}
```



```
vector<int> myvec2 = myvec;
```

```
void F(vector<int> myvec2)
{
  myvec2.push_back(7);
  ...
}
```

```
int main()
{
  vector<int> myvec = {3, 1};

  F(myvec);
```

What's due?

Extra-credit C project *due Sunday if you're interested*

HW #05 *due Tuesday 02/13*

Watch for release of ***Project 05****, due next Friday 02/16*