CS 211 : Thurs 02/29 (lecture 17)

- **Topics**: **set, search trees, implementation**

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		

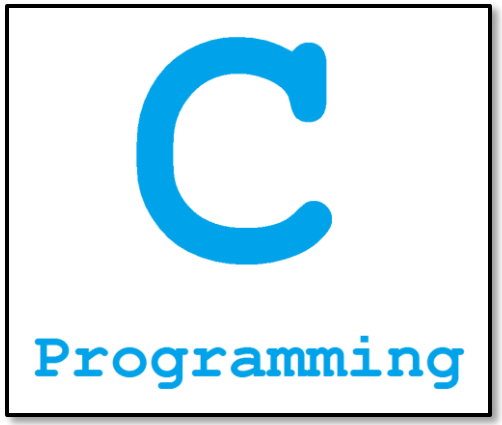
www.a-printable-calendar.com

Notes:

- *Lecture slides available on Canvas*
- *We are going to program in class today, and will* ***collect at the end of class*** *via Gradescope*
- ***Project 07*** *due Friday night (can submit as late as Sunday with late days). Gradescope is open.*

Northwestern University

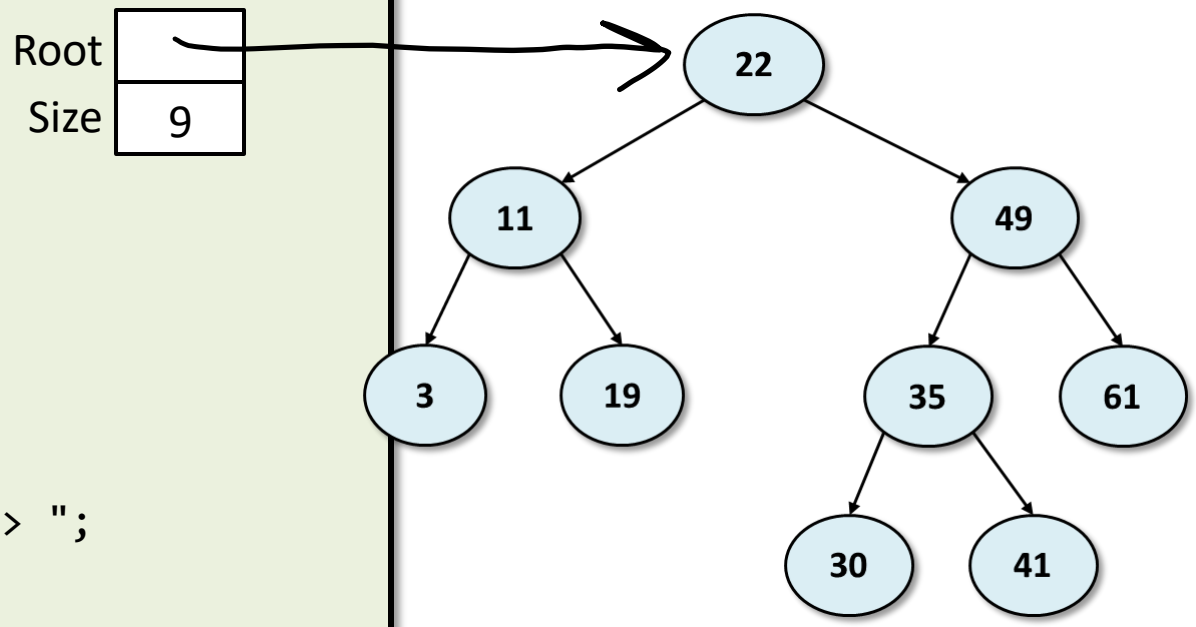

C++


C
Programming

set

- **set is another C++ abstraction for a search tree**
  - *Models mathematical set (no duplicates)*
  - *Efficient operations => O(lgN)*

```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .
  .

  int x;
  cout << "Enter an integer> ";
  cin >> x;

  if (S.contains(x))
    cout << x << " is a member" << endl;
```



```
template<typename TKey>
class set
{
private:
  struct NODE
  {
    TKey  Key;
    NODE* Left;
    NODE* Right;
  };

  NODE* Root;  // pointer to root node
  int   Size;  // # of nodes in tree

public:
  // default constructor:
  set()
   : Root(nullptr),
     Size(0)
  { }
```

Templates are the one exception where code goes into .h file --- "set.h"

```
bool contains(TKey key)
{
   NODE* cur = this->Root;

   while (cur != nullptr)
   {
     if (key == cur->Key) // found it!
       return true;
     else if (key < cur->Key)
       cur = cur->Left;
     else
       cur = cur->Right;
   }

   // if get here, not found
   return false;
}

void insert(Tkey key)
{
   NODE* prev = nullptr;
   NODE* cur  = this->Root;
   .
   .
   .

   this->Size++;
   return;
}
```

Replit

- Login to replit.com
- Open team…
- Open project "Lecture 17"

EECS Computers

```
mkdir set
cd set
cp -r /home/cs211/w2024/lecture17/* .
make
./a.out
```

(1) recursion

- **Let's rewrite contains to search recursively…**
  - *Sometimes recursion is necessary*
  - *In general it's a matter of preference whether to use iteration or recursion…*

```
bool contains(TKey key)
{
    return _contains(this->Root, key);
}
```

Recursive approach requires a helper function to do the actual recursion…

(2) Supporting [ ]

- **Let's allow the use of [ ] as well as contains…**

```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .
  .

  int x;
  cout << "Enter an integer> ";
  cin >> x;

  if (S[x])
    cout << x << " is a member" << endl;
```

```
bool operator[](TKey key)
{

    ?

}
```

(3) destructor

- Since the class dynamically allocates memory…
- We need a destructor to free memory

```
//
// destructor:
//
~set()
{
   _destroy(this->Root);
}
```

(4) parameter passing

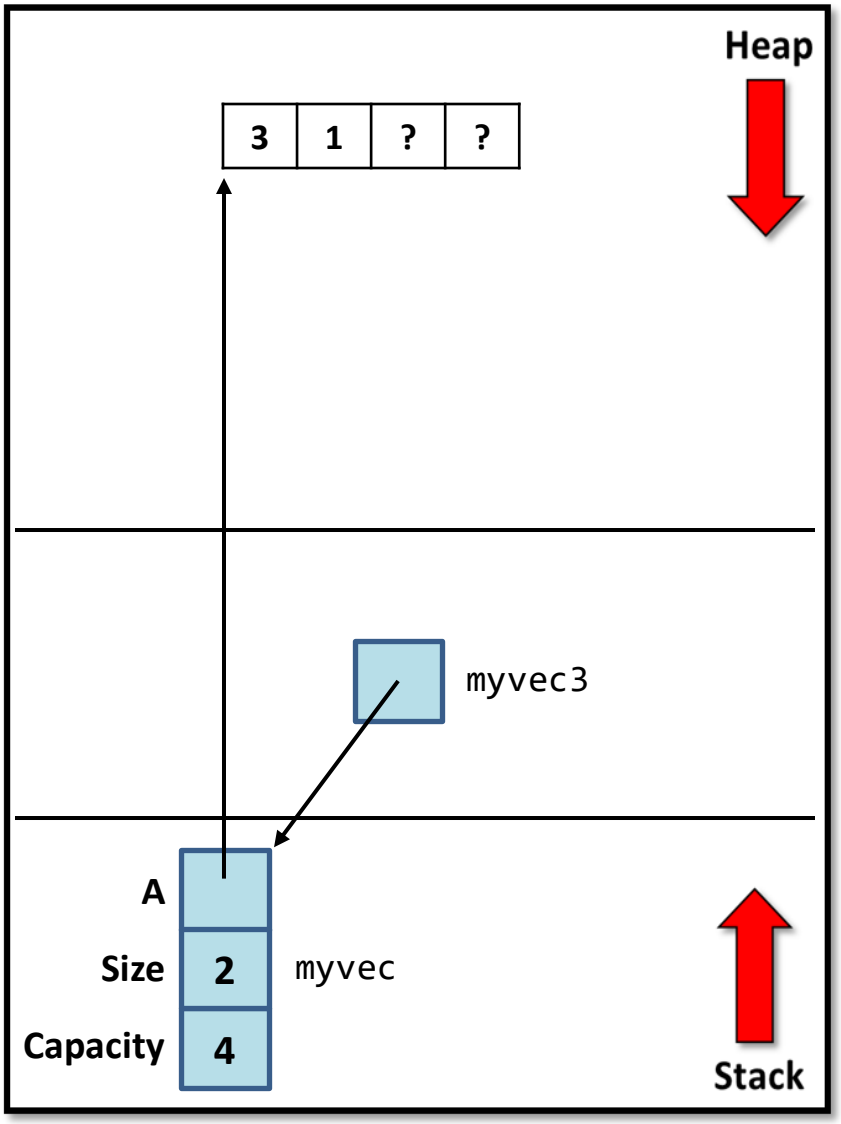
- **What happens if we pass a set by reference?**
- **What happens if we pass a set by value?**

```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .
  .

  interact_with_user(s);
```

```
void interact_with_user(set<int>& s)
{
  ...
}
```

```
void interact_with_user(set<int> s)
{
  ...
}
```

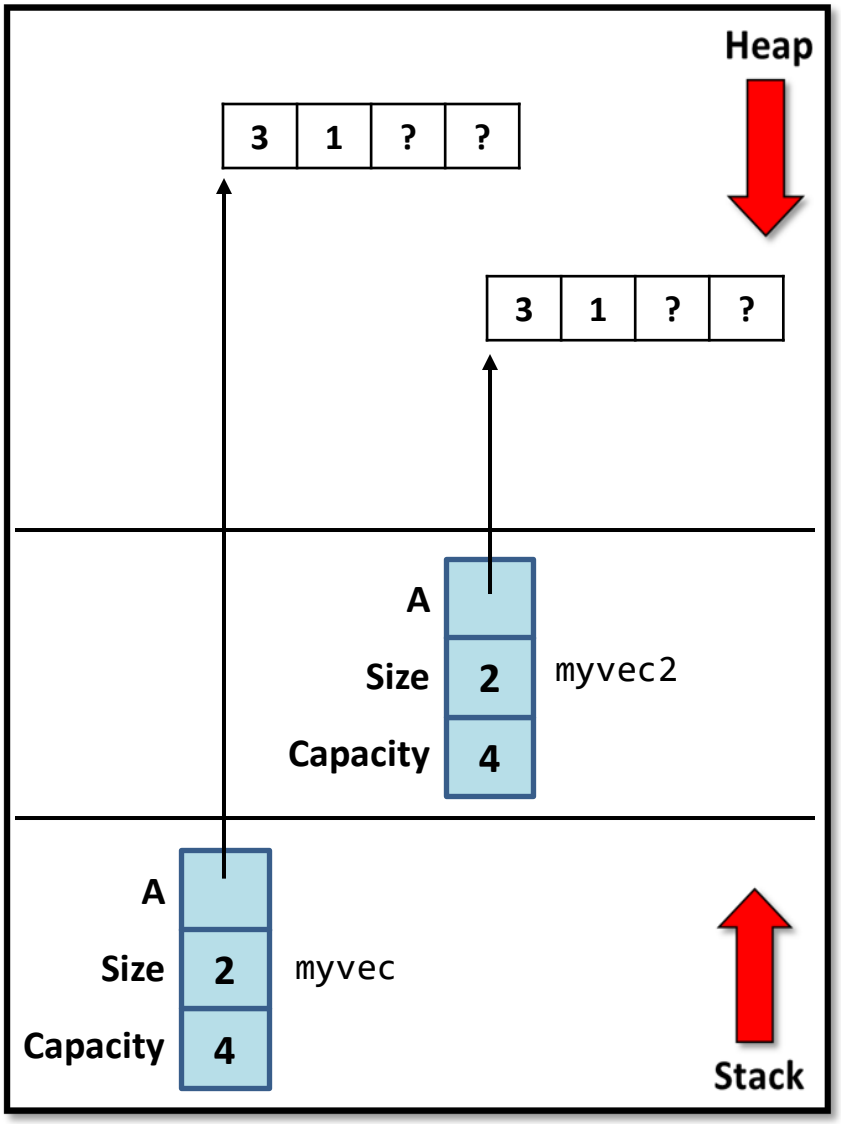
Vectors : pass-by-reference




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

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

  G(myvec);
```

Vectors : pass-by-value



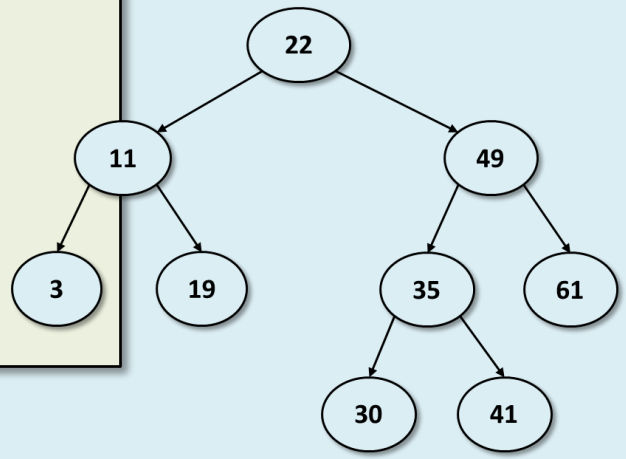

```
void F(vector<int> myvec2)
{
  myvec2.push_back(7);
  ...
}
```

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

  F(myvec);
```

pass-by-value => copy constructor

```
void interact_with_user(set<int> s)
{
  ...
}


int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .
  .

  interact_with_user(s);
```

```
//
// copy constructor:
//
set(const set& other)
{
   this->Size = other.Size;
   this->Root = _copy(other.Root);
}
```



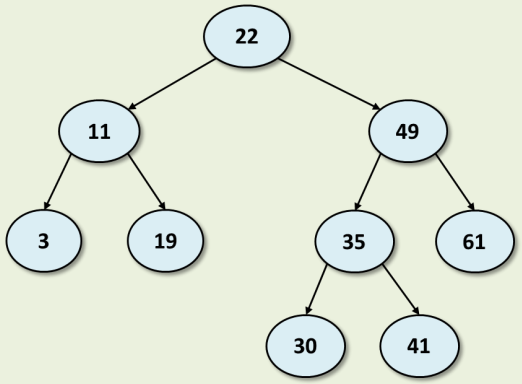
(5) iterators

- **Recall we searched maps using find( ), and it returned this pointer-like object called an iterator**
- **Why?**

```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .

  cout << "Enter an integer> ";
  cin >> x;

  auto ptr = s.find(x);
  if (ptr == s.end())
    cout << "not found" << endl;
  else
    cout << "found " << *ptr << endl;
```

Why not return a pointer?

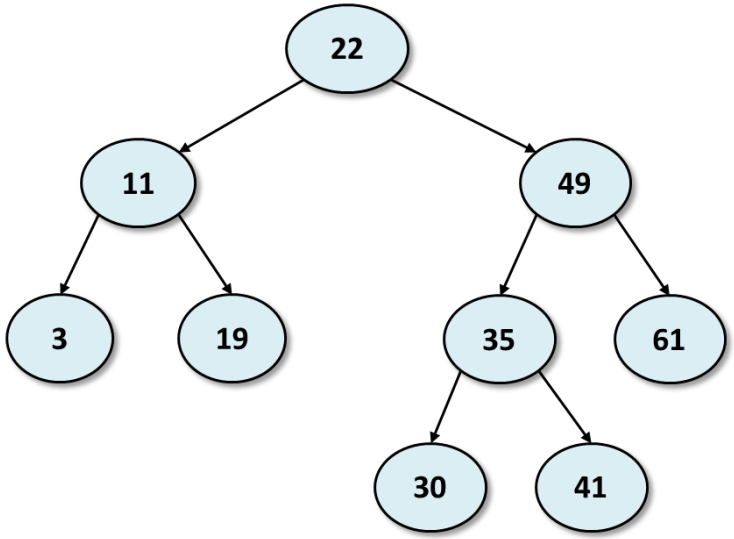
```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .

  cout << "Enter an integer> ";
  cin >> x;

  auto ptr = s.find(x);
  if (ptr == nullptr)
    cout << "not found" << endl;
  else
    cout << "found " << *ptr << endl;
```




```
TKey* find(TKey key)
{
   NODE* cur = this->Root;

   while (cur != nullptr)
   {
     if (key == cur->Key) // found it!
       return &cur->Key;
     else if (key < cur->Key)
       cur = cur->Left;
     else
       cur = cur->Right;
   }

   // if get here, not found
   return nullptr;
}
```

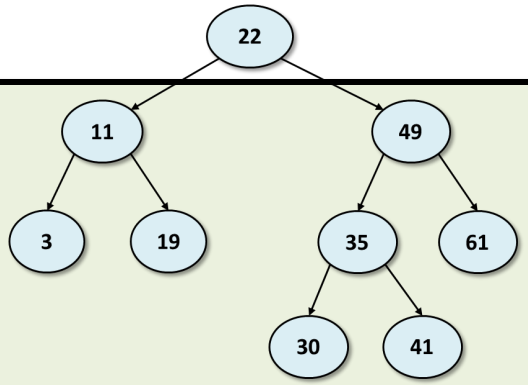
iterator => "safe" pointer

```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .

  cout << "Enter an integer> ";
  cin >> x;

  auto iter = s.find(x);
  if (iter == s.end())
    cout << "not found" << endl;
  else
    cout << "found " << *iter << endl;
```



class iterator

```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .

  cout << "Enter an integer> ";
  cin >> x;

  auto iter = s.find(x);
  if (iter == s.end())
    cout << "not found" << endl;
  else
    cout << "found " << *iter << endl;
```




```
class iterator
{
private:
  TKey* Ptr;

public:
  iterator(TKey* ptr)
   : Ptr(ptr)
  { }

  TKey operator*()
  {
    return *this->Ptr;
  }

  bool operator==(iterator other)
  {
    if (this->Ptr == other.Ptr)
      return true;
    else
      return false;
  }
};
```

find() and end()

```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .

  cout << "Enter an integer> ";
  cin >> x;

  auto iter = s.find(x);
  if (iter == s.end())
    cout << "not found" << endl;
  else
    cout << "found " << *iter << endl;
```

```
iterator find(TKey key)
{
   NODE* cur = this->Root;

   while (cur != nullptr)
   {
     if (key == cur->Key) // found it!
       return iterator(&cur->Key);
     else if (key < cur->Key)
       cur = cur->Left;
     else
       cur = cur->Right;
   }

   // if get here, not found
   return iterator(nullptr);
}

iterator end()
{
  return iterator(nullptr);
}
```

Submit your work to Gradescope

- **EECS computers?**

  make submit

- **Replit?**

  – *download set.h*

  – *upload to Gradescope "Lecture 17"*

What's due?

*Submit "**set.h**" to Gradescope*

***Project 07** is due Friday night*