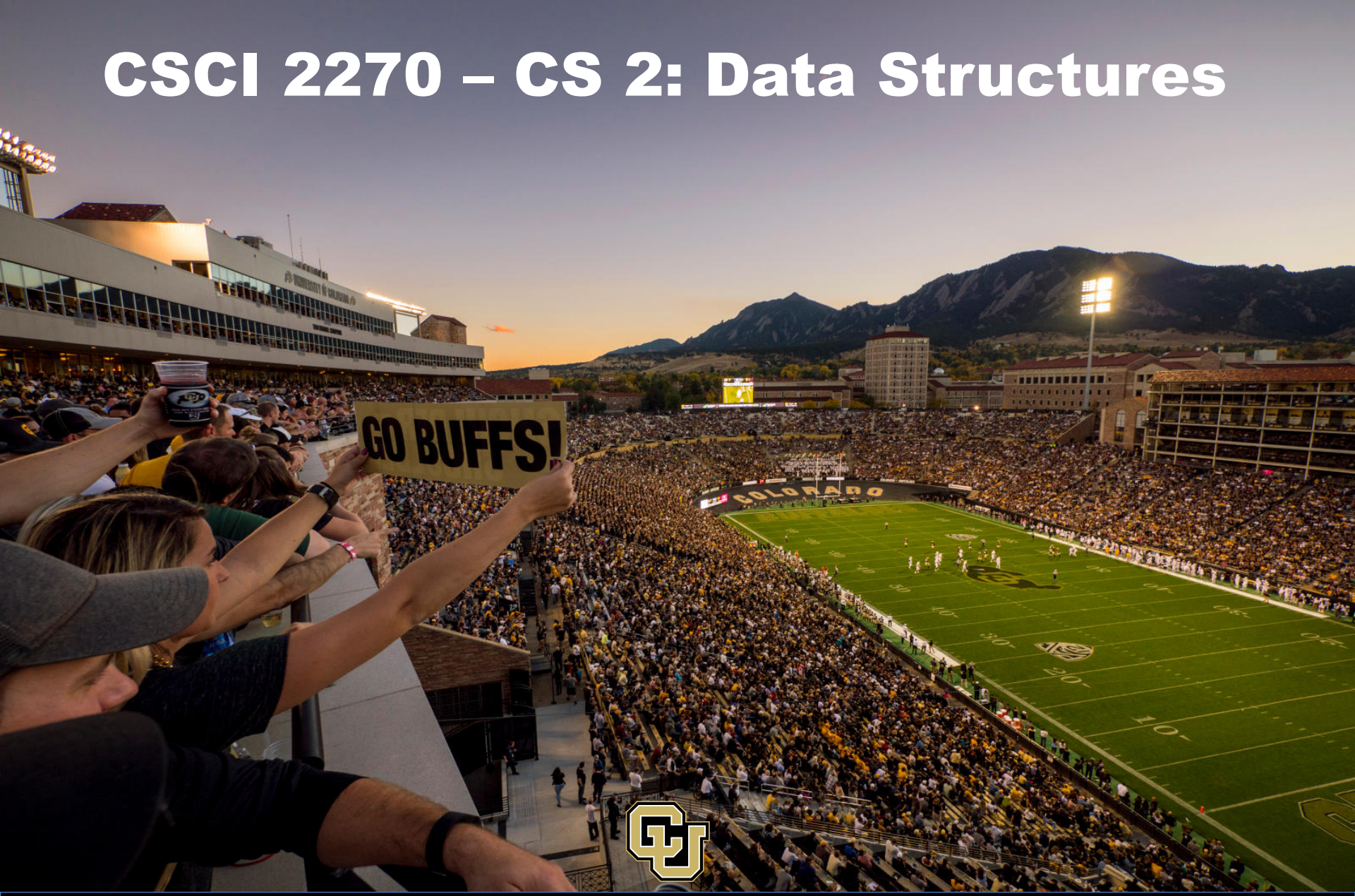
CSCI 2270 – CS 2: Data Structures



University of Colorado
Boulder

Topics

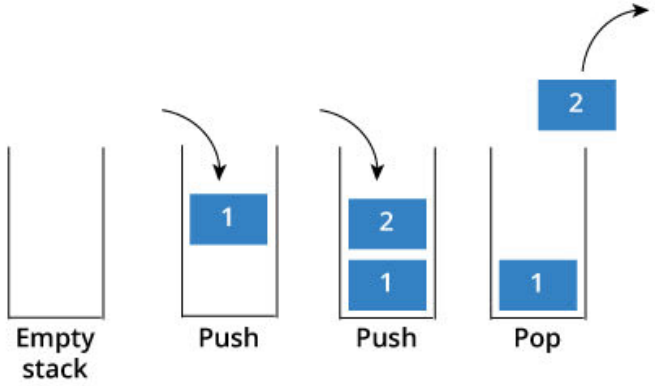
- Stacks

Stacks

- How are function calls in C++ implemented?
- Stack is a list of homogeneous elements in which the addition and deletion of elements occur only at one end, called the **top** of the stack.
- Elements at the bottom of the stack have been in the stack the longest.
- It is a Last-In First-Out (LIFO) data structure

Stack Operations

- push()
  - Add new element to the top of the stack
- top() or peek()
  - Return top element of the stack
- pop()
  - Removes the top element of the stack
- isFullStack()
  - Checks whether the stack is full
  - Returns bool
- isEmptyStack()
  - Checks whether the stack is empty
  - Returns bool
- initializeStack()
  - Stack must be empty before we start using it
  - This method initializes the stack to an empty state



Stack ADT

private:

top – keeps track of top element

count – current number of elements in stack

maxSize – limit on total size of stack (optional)

public:

initialize() – constructor

bool = isFull() – check whether stack is full

bool = isEmpty() – check if stack is empty

value = peek() – show top item (peak() ☺)

push(item) – add a new item to the top

pop() – discard item from the top

disp() – traverse entire stack

Stack – LL Implementation

```
struct Node{
   string item;
   Node *next;
};

class Stack {
   private:
      Node *top;          // pointer to top of the stack
      int count;          // number of nodes currently in stack
   public:
      Stack();
      ~Stack();

      bool isEmpty();
      void push(string newItem); // dynamically allocate new node and push onto stack
      void pop();   // remove node from top of stack and deallocate node's memory
      Node *peek();   // return pointer to the node that corresponds to top of stack
      void disp();
};
```

Stack – LL Implementation

```
struct Node{
   string item;
   Node *next;
};

Stack::Stack(){
   top = nullptr;
}
```

```
void Stack::push( string newItem )
{
   Node *temp = new Node;
   temp->item = newItem;

   if( isEmpty() ){
      top = temp;
      top->next = nullptr;
   }
   else{
      temp->next = top;
      top = temp;
   }
```

Stack – LL Implementation

```
struct Node{
   string item;
   Node *next;
};
```

```
void Stack::pop(){
   Node *temp;
   if(!isEmpty()){
      temp = top;
      top = top->next;
      delete temp;
   }
   else{
      cout << "underflow, stack empty" << endl;
   }
}
```

Stack – LL Implementation

```
Stack::~Stack(){
   Node *current;
   while( !isEmpty() ){
      current = top;
      top = top->next;
      delete current;
   }
}

bool Stack::isEmpty(){
   if( top == nullptr )
      return true;
   else
      return false;
}
```

```
Node* Stack::peek(){
   return top;
}

void Stack::disp(){
   Node *current = top;
   cout << "Top of the stack: " << endl;
   while( current != nullptr ){
      cout << current->item << endl;
      current = current->next;
   }
   cout << endl;
}
```

Stack – Array Implementation

```
#define MAXSIZE 2 // set max size for stack

class StackArr{
   private:
      int top, count; // Index for top element and total count
      string a[MAXSIZE]; // Stack array

   public:
      StackArr();

      bool isEmpty();
      bool isFull();
      void push( string newItem );
      string pop();
      void disp();
};
```

Stack – Array Implementation

```
StackArr::StackArr(){
   top = 0;
}

bool StackArr::isFull(){
   if ( top == MAXSIZE )
      return true;
   else
      return false;
}
```

```
void StackArr::push( string newItem ){
   if( !isFull() )
   {
      a[top] = newItem;
      top++;
   }
   else
   {
      cout << "Stack overflow: " << endl;
   }
}
```

Stack – Array Implementation

```
bool StackArr::isEmpty(){
   if ( top == 0 )
      return true;
   else
      return false;
}

void StackArr::disp(){
   cout << "top = " << top << endl;
   for( int i = top-1; i >= 0; i-- )
      cout << a[i] << endl;
}
```

```
string StackArr::pop(){
   string out;

   if( !isEmpty() )
   {
      out = a[--top];
   }
   else
   {
      cout << "Stack is empty " << endl;
      out = "";
   }

   return out;
}
```

Questions