Summary of Key C++ Capabilities

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

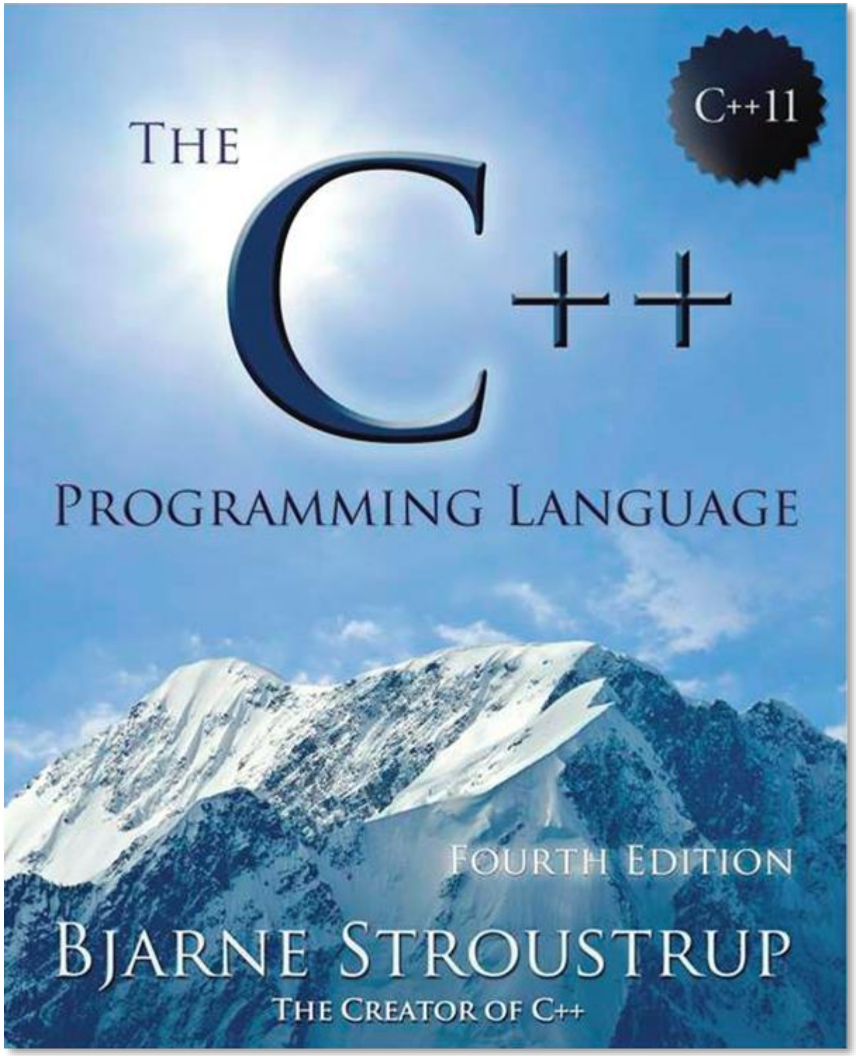
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Lesson

- Recognize the key capabilities of C++
  - Stronger type-checking (than C)
  - Support for data abstraction
  - Support for object-oriented programming
  - Support for generic programming



See www.stroustrup.com/4th.html

Summary of Key C++ Capabilities

Summary of Key C++ Capabilities

- C++'s multi-paradigm language capabilities span a range of features
  - Stronger data typing (than C)
  - Data abstraction & encapsulation
  - Generic programming
  - Sophisticated error handling
  - Object-oriented programming features
  - Identifying an object's type at runtime

Summary of Key C++ Capabilities

- C++'s multi-paradigm language capabilities span a range of features
  - Stronger data typing (than C)
    - e.g., type checking is done at compile time, function prototypes etc.

```
void foo() { ... }
/* function call arguments must
   match function prototype. */
foo(10.5, 10); // compile-error


void *ptr;
/* Implicit conversion
   from void* to int* */
int *i = ptr; // compile-error


/* Implicit conversion
   from void* to int* */
int *j = // compile-error
  malloc(5 * sizeof *j);
```

See www.careerride.com/Interview-Questions-C++-Type-checking.aspx

Summary of Key C++ Capabilities

- C++'s multi-paradigm language capabilities span a range of features
  - Stronger data typing (than C)
  - Data abstraction & encapsulation
    - e.g., classes, access control, & name spaces

```
typedef int T;
class stack {
public:
  stack (size_t size);
  stack (const stack &s);
  stack &operator=(const stack &);
  ~stack (void);
  void push (const T &item);
  void pop (void);
  const T &top () const;
  T &top ();
  bool is_empty (void) const;
  bool is_full (void) const;
private:
  size_t top_, size_; T *stack_;
};
```

Summary of Key C++ Capabilities

- C++'s multi-paradigm language capabilities span a range of features
  - Stronger data typing (than C)
  - Data abstraction & encapsulation
  - Generic programming
    - e.g., parameterized classes & functions

```
template<typename T>
class stack {
public:
  stack (size_t size);
  stack (const stack<T> &s);
  stack<T> &operator=(const
                      stack<T> &);
  ~stack (void);
  void push (const T &item);
  void pop (void);
  const T &top () const;
  T &top ();
  bool is_empty (void) const;
  bool is_full (void) const;
private:
  size_t top_, size_; T *stack_;
};
```

Summary of Key C++ Capabilities

- C++’s multi-paradigm language capabilities span a range of features
  - Stronger data typing (than C)
  - Data abstraction & encapsulation
  - Generic programming
  - Sophisticated error handling
    - e.g., exception handling

```
template<typename T>
class stack {
public:
  class overflow {};
  class underflow {}
  ...
  void push (const T &item) {
    if (is_full ())
      throw stack::overflow ();
    stack_[top_++] = item;
  }

  void pop (void) {
    if (is_empty ())
      throw stack::underflow ();
    --top_;
} ...
```

Summary of Key C++ Capabilities

- C++’s multi-paradigm language capabilities span a range of features
  - Stronger data typing (than C)
  - Data abstraction & encapsulation
  - Generic programming
  - Sophisticated error handling
  - Object-oriented programming features
    - e.g., abstract classes, inheritance, & virtual methods

```
template<typename T>
class stack {
public:
  virtual ~stack (void);
  virtual void push (const T
                     &item) = 0;
  virtual void pop (void) = 0;
  virtual T &top (T &item) = 0;
  virtual const T &top (T &item)
                        const = 0;
  virtual bool is_empty (void)
                      const = 0;
  virtual bool is_full (void)
                      const = 0;
};
```

Summary of Key C++ Capabilities

- C++'s multi-paradigm language capabilities span a range of features
  - Stronger data typing (than C)
  - Data abstraction & encapsulation
  - Generic programming
  - Sophisticated error handling
  - Object-oriented programming features
  - Identifying an object's type at runtime
    - e.g., Run-Time Type Identification (RTTI)

```
template<typename T>
class v_stack :
  public stack<T> { ... }

template<typename T>
class l_stack :
  public stack<T> { ... }

stack<int> *s = make_stack(...);

if (dynamic_cast
      <l_slack<int> *>(s))
  ...
else if (dynamic_cast
           <v_stack<int> *>(s))
  ...
```

End of Overview of Key C++ Capabilities