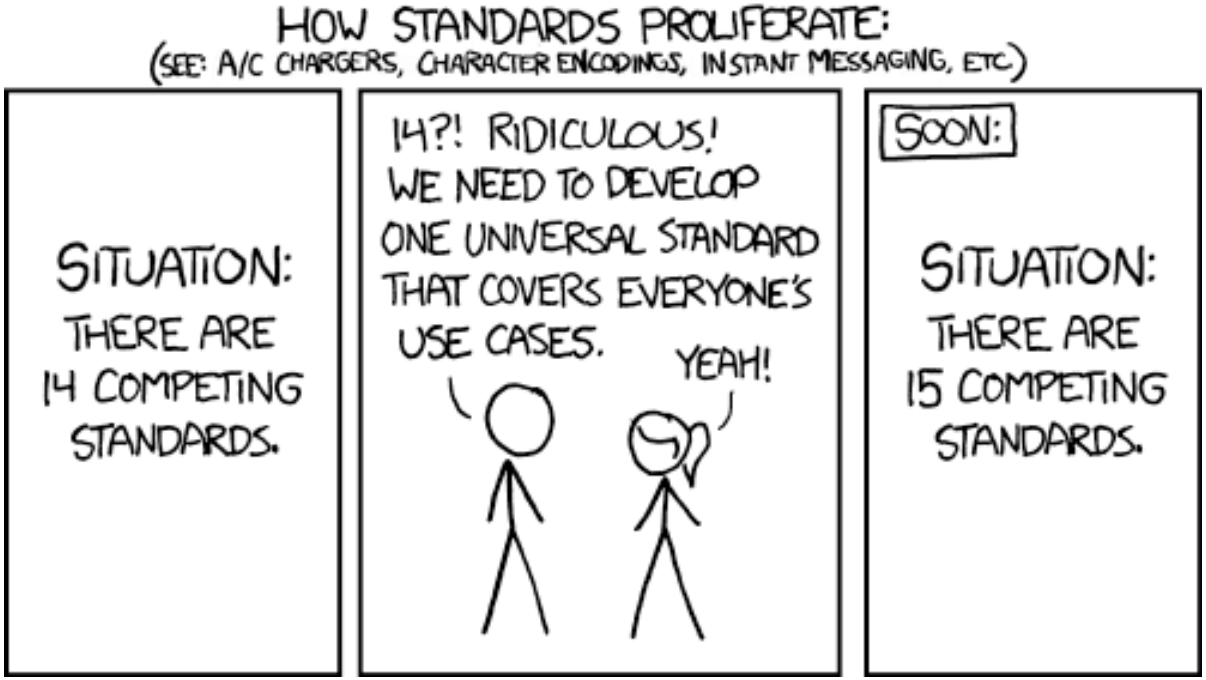
Programming Languages
CS 214

Exam: what to expect

~60 questions
- operator associativity
- operator precedence
- week 4 set operators: 5 questions
- 7 questions on history:
- first Language
- first implemented Implemented
- Multitasking
- Object Oriented Language
- Procedural Language
- Classes
- Lists
- Functional Language
- Block Structue (begin and end)
- Multi-Dimensional Arrays
- Logic Language
- Generic Types
- 24 questions on identifying the formalisms of different implementations of branching, selection and loopin
- prove ambiguity
- create a bnf

“Language Independent Description”



Types

A is a set *V* of values, and a set of operations onto *V*.

Examples from C++:

- The ___ type:

  V = {INT_MIN, ... -1, 0, 1, ..., INT_MAX-1, INT_MAX}

  O = {<<, >>, +, -, *, /, %, =, ++, --, ...}

- The ___ type:

  V = {NUL, ..., '0', …, '9', …, 'A', ..., 'Z', ..., 'a', ..., 'z', DEL}

  O = {<<, >>, =, ++, --, isupper(), islower(), toupper(), … }

- The ___ type:

  V = {"", "A", "B", "C", ..., "AA", "AB", "AC", ..., "AAA", ...}

  O = {<<, >>, +, +=, [], find(), substr(), … }

Fundamental Types

Let's assume the existence of some basic (primitive) types:

Name	V	C++	Ada	Smalltalk	Lisp
bool	false, true	*bool*	*boolean*	*Boolean*	*boole*
char	the set of chars	*char*	*character*	*Character*	*character*
int	the integers	*int*	*integer*	*Integer*	*integer*
real	the reals	*double*	*float*	*Float*	*real*

Such "primitive" types require little memory:

- *bool*—a single bit
- *char*—1-2 bytes
- *int*—a single word
- *real*—1-2 words

Non-Fundamental Types

New, non-fundamental types can be built from existing types:

- _____ types (arrays, vectors, lists, etc.) store multiple values of _____ type.
- _____ types (records, structs, classes, etc.) store multiple values of _____ types.

A _____is a formal mechanism for modeling a new type that is _____, using the mathematics of ____.

A type constructor has 3 components:

- The _____ used to denote that constructor;
- The _____ that constructor; and
- The _____associated with that constructor.

Set Constructor I: _____

Kleene Closure is a formalism for modeling _____.

– The Kleene Closure of a set A is denoted _____

– The Kleene Closure of a set is the set of all tuples that can be formed using elements of that set.

Example: The Kleene Closure of bool -- _____ -- is the infinite set:

```
{ (), (false), (true), (false, false), (false, true),
 (true, false), (true, true), (false, false, false), … }
```

– For a tuple t ∈ A*, the operations include:

null(A*) → bool

```
null(())          → true
null((false))     → false
null((true))      → false
```

first(A*) → A

```
first((true, false))   → true
first((false, true))   → false
```

rest(A*) → A*

```
rest((true, true, false)) → (true,false)
rest((false, true, true)) → (true, true)
```


Kleene Closure Examples

If _____ is the set of ASCII characters, what is _____ ?

– The infinite set of all tuples formed from ASCII characters.
(in C, the set of all character strings).

The C/C++ notation: "Hello"
is just a different syntax for: ('H', 'e', 'l', 'l', 'o')

Thus, ____ denotes a sequence (array, list, …) of integers;

```
int intStaticArray[32];
int * intDynamicArray = new int[n],
vector<int> intVec;
list<int> intList;
```

_____ denotes a sequence (array, list, …) of reals;
and so on.

Sequence Operations

Sequence operations can be built via *null()*, *first()*, and *rest()*

- An output operation can be defined like this (pseudocode):

```
void print(ostream& out, int * a) {
  if ( !null(a) ) {
    out << first(a) << ' ';
    print(out, rest(a));
  }
};
```

- A subscript operation can be defined like this (pseudocode):

```
char& operator[](char * a, int i){
  if (i <= 0)
    return first(a);
  else
    return operator[](rest(a), i-1);
};
```

In Lisp:

___ is called *car*

___ is called *cdr*.

Set Constructor II: _____

Product is a formalism for modeling _____.

- The product of two sets A and B is denoted $A \times B$.
- $A \times B$ consists of all ordered pairs (a, b): $a \in A, b \in B$.

  $A \times B \times C$ consists of all ordered triples (a, b, c): $a \in A, b \in B, c \in C$.

  $A \times B \times \ldots \times N$ consists of all ordered n-tuples (a, b, …, n): $a \in A, b \in B, \ldots, n \in N$.

  Example: the set *bool* × *char* has 256 elements:
  { …, (true, 'A'), (false, 'A'), (true, 'B'), (false, 'B'), …, }.
- Operations associated with product are the *projection* operations:
  - _____, applied to an n-tuple $(s_1, s_2, \ldots, s_n)$ returns s_1.
  - _____, applied to an n-tuple $(s_1, s_2, \ldots, s_n)$ returns s_2.
  - _____, applied to an n-tuple $(s_1, s_2, \ldots, s_n)$ returns s_n.

Product Example: C++ structs

```
struct Student
{
    int id;
    double gpa;
    char gender;
};
Student aStudent;
```

Formally, a Student consists of:

Formally, a particular Student:

```
aStudent.id = 12345;
aStudent.gpa = 3.75;
aStudent.gender = 'F';
```

is the 3-tuple: (12345, 3.75, 'F').

The C++ "dot-operator" is a projection operation:

```
cout << aStudent.id          // extract id
     << aStudent.gpa         // extract gpa
     << aStudent.gender      // extract gender
     << endl;
```


Set Constructor III: _____

Function is a formalism for *subprograms* (type operations).

– The set of all functions from a set A to a set B is denoted _____.

– A particular function *f* mapping A to B is denoted _____.

Examples:

– The set _____contains all functions that map ___ values into ____ values, some C examples of which include:

```
isupper('A') → true        islower('A') → false
isalpha('A') → true        isdigit('A') → false
isalnum('A') → true        isspace('A') → false
```

– The set _____contains all functions that map ____ values into ____ values, some C examples of which include:

```
tolower('A') → 'a'         toupper('a') → 'A'
```


Function and Product

What does this set contain? _____

– All functions that map pairs of integers into an integer.

Examples? `+((2, 3))` → `5` `-((2,3))` → `-1`
`*((2, 3))` → `6` `/((2,3))` → `0`

Suppose we define an aggregate named *IntPair*:

```
struct IntPair {
   int a,
       b;
};
```

and then define a function named add():

```
int add(IntPair ip) {
  return ip.a + ip.b;
};
```

add() is a member of the set: _____

– The function constructor let us create new operations for a type.

Function Arity

Product serves to denote an aggregate or an argument-list.

What does this set contain? _____

– All functions that map pairs of integers to a boolean.

Examples? `==((2, 3))` → **`false`** `!=((2,3))` → **`true`**
`<((2, 3))` → **`true`** `>((2,3))` → **`false`**

Definition:

The number of operands an operation requires is its _____.

– Operations with 1 operand are _____ operations, with _____.

– Operations with 2 operands are _____ operations, with _____.

– Operations with 3 operand are _____ operations, with _____.

– ...

Example Ternary Operation

The C/C++ conditional expression has the form:

_____

producing _____ if _____ is true, and
producing _____ if _____ is false.

Here is a simple *minimum()* function using it:

```
int minimum(int first, int second) {
  return (first < second) ? first : second;
};
```

The C/C++ conditional expression is a _____;
the one above can be formally described by:

_____

Example Ternary Operation: Pluralization

```
cout << "Enter the number of values you have to process: ”;
int n;
cin >> n;
     … read and process the n values
cout << "You entered " << n << "value"
     << (n == 1) ? "." : "s." // based on n, pluralize ‘value’
     << endl;
```

This ternary operation is from (bool $\times$ char* $\times$ char*) $\rightarrow$ char*

Operator Positioning

Operators are also categorized by their position relative to their operands:

- _____ operators appear _____ their operands: `1 + 2`
- _____ operators appear _____ their operands: `+ 1 2`
- _____ operators appear _____ their operands: `1 2 +`

`* + 2 3 - 4 2  ≡ (2 + 3) * (4 - 2) ≡ 2 3 + 4 2 - *`

Prefix, infix, and postfix notation are different conventions for the same thing; a language may choose any of them:

C++ Expr	Category	Value	Lisp Expr	Category	Value
x < y	binary, infix	true, false	(< x y)	binary, prefix	true, false
++x	unary, prefix	x+1	(incf x)	unary, prefix	x+1
11 + 12	binary, infix	23	(+ 11 12)	binary, prefix	23
!*flag*	unary, prefix	neg. of *flag*	(not *flag*)	unary, prefix	neg. of *flag*
cout << x	binary, infix	cout	(princ x *str*)	binary, prefix	x
x++	unary, postfix	x	None		

Review

There are four fundamental types:

– *bool, char, int, real*

New types can be modeled using three set formalisms:

- _____(sequence types): ___
  Operations: _____
- _____ (aggregate types): _____
  Operations: _____
- _____ (type operations): _____

These _____ provide a mechanism for _____.

Practice Using Constructors

Give formal descriptions for:

- The *logical and* operation (&&):
  - How many operands does it take? _____
  - What types are its operands? _____
  - What type of value does it produce? _____

  So && is a member of _____
- The C++/*STL substring* operation (*str*.substr(*i*,*n*)):
  - How many operands does it take? _____
  - What types are its operands? _____
  - What type of value does it produce? _____

  So substr() is a member of: _____
- For you: The *logical negation* operation (!):

Python's Genius

By including **self** in the list of parameters, python makes the context of the function abundantly clear

```
Class MyClass:
        def __init__(self):
                self.total =0
        def add(self, a, b)
                self.total = a + b
                return self.total
```


Logical Negation (!)

How many operands does it have? _____

What types are they? _____

What type of value does it produce? _____

Then ! is a member of? _____

More Practice

- For you: this *C++ record:*

```
struct Student {
   int myID;
   string myName;
   bool iAmFullTime;
   double myGPA;
};
```


C++ Student Structure

As an aggregate type,

Student is a member of: _____

More Practice

- For you: this *C++ record:*

```
struct Student {
  int myID;
  string myName;
  bool iAmFullTime;
  double myGPA;
};
```

- For you: an *accessor* method:

```
struct Student {
  int myID;
  int getId() const;
  string myName;
  bool iAmFullTime;
  double myGPA;
};
```


C++ Accessor Method

As a method/function:

How many operands? _____

What are their types? _____

What type of value does it return? _____

Then getID() is a member of: _____

More Practice

- For you: this *C++ record:*

```
struct Student {
  int myID;
  string myName;
  bool iAmFullTime;
  double myGPA;
};
```

- For you: an *accessor* method:

```
struct Student {
  int myID;
  int getId() const;
  string myName;
  bool iAmFullTime;
  double myGPA;
};
```

- How does this affect our *Student* description?

Adding an Accessor to Student

```
struct Student {
  int myID;
  int getId() const;
  string myName;
  bool iAmFullTime;
  double myGPA;
};
```

More Practice (ii)

- For you: A "complete" class:

```
class Student {
 public:
  Student();
  Student(int, string, bool, double);
  int getID() const;
  string getName() const;
  bool getFullTime() const;
  double getGPA() const;
  void read(istream &);
  void print(ostream &) const;
 private:
  int    myID;
  string myName;
  bool   iAmFullTime;
  double myGPA;
};
```


Class Student

```
class Student {
 public:
  Student();
  Student(int, string, bool, double);
  int getID() const;
  string getName() const;
  bool getFullTime() const;
  double getGPA() const;
  void read(istream &);
  void print(ostream &) const;
 private:
  int    myID;
  string myName;
  bool   iAmFullTime;
  double myGPA;
};
```

Summary

A type consists of _____ and _____.

There are four *primitive types*: _____

The set constructors:

- _____ (to represent sequences)
- _____ (to represent aggregates and function parameters)
- _____ (to represent operations)

provide a formal way to model new (non-primitive) types, independent of any particular programming language:

→ Use the _____ and _____ to represent the type's ____

→ Use the _____ constructor to represent the type's _____