CSCI 135
Data Types

Types

C/C++ Types (Key: optional words, *C++ only (not C)*)

Primitive Types:

Group	Name			
Integral	signed short int	signed int	signed long int	signed long long int
	any of above with unsigned instead of signed (or abbreviate unsigned as uns)			
	char	*Unicode types*		
	enum	*bool*		
Floating Point	float	double	long double	
Other	void	pointers		

Non-Primitive Types:

- Strings (C and C++ versions)
- C Composite Types: arrays, structures
- C++ Composite Types: many types of **containers**
- . . .

Enumerations

Example: you want a variable, x, that can take a value of either red, green, or yellow (*i.e.*, an enumeration of red, green, yellow)

Approaches:

1. Declare x as a string.
   - Space-inefficient: need 6 bytes (at least)
   - Time-inefficient: need to compare x against up to 6 characters)
   - Bug-prone: *e.g.*, what if you have `x = ‘‘yelen’’`
2. Declare x as an int, and let red/green/yellow correspond to 0,1,2.
   - Still bug-prone (*e.g.*, `x = 3`)
   - Requires programmer to keep correspondence in head (hopefully commented)
3. Declare x as a char, and use ’R’, ’G’, ’Y’.
   - Same problems as previous case, though a bit more understandable.
4. Enumerated types! A type that allows x to **only** take on values red, green, and yellow. (especially useful with switch)

Enumerated Types - Example

```
enum Color {red, yellow, green};          Color is a type name
Color x = red;                            x is a variable name
x = green;
switch(x) {
  case red    : cout << "Red"; break;
  case yellow : cout << "Yellow"; break;
  case green  : cout << "Green"; break;
  };
cout << endl;
x = blue;                                 compile time error
x = 3;                                    compile time error
```

- Common stylistic guideline: type name starts with uppercase character; *sometimes* enum tag is all uppercase
- Internally mapped to int by compiler, with red=0, yellow=1, green=2 (but programmer doesn't need to know this!). Mapping can be over-ridden by programmer (but seldom need to):
  `enum Color {red, yellow=10, green=20};`
- Better to use enum instead of int when you have a small number of values for a variable.

Boolean Types in C/C++

- C (before C99): bool is not supported; most programmers used 0 and 1 (ints) to simulate booleans.
- C99 and on: bool defined in `stdbool` library as:

  ```
  enum _Bool {false,true}.
  ```

  To use, include library and declare a variable of type bool as usual
- C++: bool is a primitive type.

⚠Many programmers still use old style (though you should avoid for new code)

Composite Types

[The type of] a collection (grouping) of entities that you would also like to treat as a single entity for organizational and programming purposes.

Examples:

- <name, id, gender> (*i.e.*, a collection containing a string, an int, and an enumeration type {male,female}
  ⇒ Structures (C/C++) or classes (C++)
- A roster of names (*i.e.*, an array of strings) with efficient access of arbitrary indexed elements (*e.g.*, the 5th name)
  ⇒ Arrays (C/C++) or vectors (C++)
- Various other composite types exist, depending on what types of accesses are supported efficiently (*e.g.*, linear access, random access)
- You can even have composite types of composites!
  Ex: an array of <name,id,gender> structures.

container-eps-conver

You will learn these **data structures** in CSCI 135, and again more formally in CSCI 235.

Containers

A **container** is a collection of entities:

- Typically a homogeneous collection – *i.e.*, each element of the collection has the same type.
- Different types of containers differ by which operations are possible and efficient. Ex:
  - Fixed size? Variable size? Fixed but changable size?
  - Storage overhead efficiency
  - Are container elements ordered?
  - Linear vs Random access: e.g., get the next element (given the 'current' element) vs. get the 27'th element.
- Terminology typically used only with object oriented languages such as C++ (not C), though C has container-like data structures.
- Strictly speaking, most C++ containers are provided by the Standard Template Library (STL), not the base C++ language.

Defining Your Own Type Name

Goal: Create a new type name

typedef <type> <name>;

Examples:

```
typedef unsigned long Mylong;   Define type Mylong
Mylong x;                       Declare variable x
unsigned long y;                Same type as x
y = x;                          (so no casting needed)
```

- Typically used for composite types, to avoid repeating [possibly long] type definitions.
- Only creates a new name for existing type (not a new type)
- Type name is local to defining scope
- Common naming conventions are to start with an upper case letter or end with t or _t.