# Sep / 14 / 2020

## C++ Types, Container

* C++ types, std::vector and std::map
  + Variables:
    - Variable: a name, a type, a value, an address in memory
    - Obtain the address: &var
  + Types:
    - Primitive data-types

E.g. to get the size of a data type: sizeof(<data\_type>)

* + - Size of a type is implementation defined
    - Structure: use it when only want to define a structure without functions use it
    - Class: fields + functions
    - Pointer and array type:
      * A pointer store the memory address
        + int\* p
        + dereference a null pointer will gives an error)
      * Pointer arithmetics
        + int\* p ; int\* p2 = p+1 // <- points the next int
        + Int arr[5]; x=arr[2] // or \*( arr + 2) <- point to the third element in the array
    - Bool and char type, auto keyword:
      * Conversion between char <-> int:
        + char a; a-’0’ ;
      * Auto: compiler infer the type itself:

E.g. int max (int x, int y);

auto m = max(x, y);

* + Class:
    - Constructor:

E.g (from word\_count wc++.cpp)

wc::wordCounter::wordCounter(const std::string& dir, uint32\_t numthreads) : dir( dir ),

num\_threads(num\_thread){ }

* + - Initialization:
      * wc::wordCounter word\_counter\_one(“/home/sagar/Documents”, 4);
      * wc::wordCounter word\_counter\_two = wc::wordCounter(“/home/sagar/Documents”, 4);
    - Keyword new will returns a pointer to the object
  + Type qualifiers (const, volatile)
    - Const : variable cannot change state after declaration
      * Define const in argument: telling compiler this function will not change the input variable
      * Object could be passed via const or reference
    - Constexpr : value known at compile time
  + Plain Old Data (POD):
    - Array size need to be constant and know at compile time

E.g. main.cpp:23:12: warning: ISO C++ forbids variable length array ‘args’ [-Wvla]

23 | string args[argc];

* + - POD type a class or struct without pointers, constructors/destructors and virtual member functions
    - Native arrays: c-arrays , not std:array
* Basic C++ philosophy
  + When to use pointers?
    - Prefer object over pointers
  + Standard Template Library
    - Collection of classes and functions for general purpose
  + std::vector<T>
    - A dynamic array: initial size 0 default ; resizable, no bound check
    - Memory: store data in continuous memory

(e.g. to access the next element in the array char\* a = std::vector<char> )

* + - Random access: arr[ind]; O(1)
    - insert : O(n) to ensure the contiguous memory placement of the elements in the vector
    - Reallocate : push\_back , resize (if not enough memory then reallocate ->amortize advantage)
      * If know the size beforehand, it is preferable to preallocate the size
  + std::list<T>
    - A collection of elements at non-contiguous memory
    - Insertion and deletion : O(1)

E.g. find all the files in the directory

Use list to collect the files

(avoid amortized memory allocation in vector)

Then convert the list to vector to allow random access

* + std::map<K, V>
    - Map keys to values: access element b a key, a vector when need to access by position
    - Implementation using tree: insert/remove/erase/search O(log n)
    - std::unordered\_map<K,V> -hash-based map,

O(1) , but unpredictable complexity, depends on the hash functions (how to handle collision)

Use it when want further optimize than O(log n) complexity

* + - std::insert : will be ignored if the key already exists

E.g.

using //pair initialization

{word, cnt};

Std::sort //construct a comparator

* + Convert between containers