```
/*
###############################################################################
###############################################################################
#<a rel="license" href="http://creativecommons.org/licenses/by-nc-
nd/4.0/"><img alt="Creative Commons Licence" style="border-width:0"
src="https://i.creativecommons.org/l/by-nc-nd/4.0/88x31.png" /></a><br
/>This work is licensed under a <a rel="license"
href="http://creativecommons.org/licenses/by-nc-nd/4.0/">Creative Commons
Attribution-NonCommercial-NoDerivatives 4.0 International License</a>.
###############################################################################
###############################################################################
#Course Authored By:
#-------------------------------------------------------------------------
----------------------------------
#K.Srinivasan
#NeuronRain Documentation and Licensing: http://neuronrain-
documentation.readthedocs.io/en/latest/
#Personal website(research): https://sites.google.com/site/kuja27/
#-------------------------------------------------------------------------
----------------------------------
###############################################################################
###############################################################################
*/

This is a non-linearly organized, code puzzles oriented, continually
updated set of course notes on C++ language. This
complements NeuronRain course materials on Linux Kernel, Cloud, BigData
Analytics and Machine Learning and covers
fundamentals of C++.
-------------------------------------------------------------------------
-----------------------------------------------------

22 February 2017
----------------
An example on C++ templates and Runtime type identification:
-----------------------------------------------------------
Example code snippet in code/templates.cpp implements a simple
templatized book class with book type string as template parameter.
Template
book is instantiated with template and typename keywords and type T can
be any subject type passed in as template parameter. Template class
EBook derives from base class Book<T>. A subtlety in this example is
absence of default constructor for Book<T> causes following compiler
error:

g++ -g -o templates -I/usr/local/include -L/usr/local/lib -std=c++14
*.cpp
templates.cpp: In instantiation of 'EBook<T>::EBook(T) [with T =
std::__cxx11::basic_string<char>]':
templates.cpp:47:28:   required from here
templates.cpp:36:2: error: no matching function for call to
'Book<std::__cxx11::basic_string<char> >::Book()'
  {
  ^
templates.cpp:18:2: note: candidate: Book<T>::Book(T) [with T =
std::__cxx11::basic_string<char>]
  Book(T type)
  ^
templates.cpp:18:2: note:   candidate expects 1 argument, 0 provided
```

```
templates.cpp:8:7: note: candidate: Book<std::__cxx11::basic_string<char>
>::Book(const Book<std::__cxx11::basic_string<char> >&)
 class Book
       ^
templates.cpp:8:7: note:   candidate expects 1 argument, 0 provided
templates.cpp:8:7: note: candidate: Book<std::__cxx11::basic_string<char>
>::Book(Book<std::__cxx11::basic_string<char> >&&)
templates.cpp:8:7: note:   candidate expects 1 argument, 0 provided

------------------------------------------------------------------------
---------------------------
Adding default constructor:
      Book()
         {
         }
removes compilation error and following is printed:
-----------------------------------------------------
Instantiating Book of type Maths
template type:NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
Instantiating Book of type ComputerScience
template type:NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
Instantiating Book of type Physics
template type:NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
Instantiating Book of type History
template type:NSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
Instantiating EBook of type English
Read book
Read book
Read book
Read book
Read book
-------------------------------------------------------------------
```

In above example, read_book() is a virtual function in superclass Book which can be overridden in derived classes. Previous output indicates how dynamic polymorphism works and read_book() of Book<T> is invoked from derived class EBook<T>. Type information is printed by typeid keyword of C++. This example is built using G++ with C++14 standard compiler option.

---

858. (THEORY and FEATURE) 17 July 2017,9 August 2020 - Self-Aware Software, Quines - related to all sections on Formal Languages, Program Analysis, Software Analytics

---

Question: How can a program print its source itself as output ? [Quine - self-aware code]

Answer: Theoretically, there exists a lambda function with a fixed point i.e f(x)=x. Unix/Linux binaries are stored in ELF format which have debugging information embedded in DWARF entries (as set of DIEs - Debugging Information Entries). There are utilities like objdump and dwarfdump which display the DIEs. For example, following is the DWARF dump of asfer executable pointing to the compilation source directory in DW_AT_comp_dir:

```
root@Inspiron-1545:/media/shrinivaasanka/6944b01d-ff0d-43eb-8699-
cca469511742/home/shrinivaasanka/Krishna_iResearch_OpenSource/GitHub/asfe
r-github-code/cpp-src# objdump --dwarf=info asfer |more

asfer:     file format elf32-i386
```

```
Contents of the .debug_info section:

  Compilation Unit @ offset 0x0:
   Length:        0x16f7e (32-bit)
   Version:       4
   Abbrev Offset: 0x0
   Pointer Size:  4
 <0><b>: Abbrev Number: 1 (DW_TAG_compile_unit)
    <c>   DW_AT_producer    : (indirect string, offset: 0x35560): GNU
C++14 5.2.1 20151010 -mtune=generic -march=i686 -g -std=c++14 -fstack-pro
tector-strong
    <10>   DW_AT_language    : 4  (C++)
    <11>   DW_AT_name        : (indirect string, offset: 0xe304):
DecisionTreeClassifier.cpp
    <15>   DW_AT_comp_dir    : (indirect string, offset: 0x7933):
/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea24/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfe
r-github-code/cpp-src
    <19>   DW_AT_ranges      : 0x210
    <1d>   DW_AT_low_pc      : 0x0
    <21>   DW_AT_stmt_list   : 0x0

There are libdwarf libraries for programmatically querying ELF DWARF
DIEs. Thus a wrapper reflection code invoking the dwarfdump on executable
can be written to print the source(assumes fresh compilation everytime).

------------------------------------------------------------------------
12 September 2017
------------------------------------------------------------------------
Question: Remove duplicates in a string in-place (no extra space) e.g
shrink aabbcba to abc

Answer: One possible solution is to sort the string (by an in place
sorting algorithm like quicksort) in ascending order of unicode value e.g
aabbcba is sorted to aaabbbc. Implement the string as linked-list of
literals, Scan the string linked list and remove repetitive alphabets and
repeat till duplicates are removed. Code example for this is at
code/remove_duplicates.cpp uses STL string sort() to sort the string, STL
erase() to simulate a linked list node erasure, and shows how STL
iterators for strings begin() and end() are applied.
--------------------------------------------------
str:9aaabbbcccddddddddeeeffggjjksss
str:9aaabbbcccddddddddeeeffggjjksss
str:9aaabbbcccddddddddeeeffggjjksss
str:9aabbbcccddddddddeeeffggjjksss
str:9aabbbcccddddddddeeeffggjjksss
str:9aabbcccddddddddeeeffggjjksss
str:9aabbcccddddddddeeeffggjjksss
str:9aabbccddddddddeeeffggjjksss
str:9aabbccddddddddeeeffggjjksss
str:9aabbccddddddeeeffggjjksss
str:9aabbccddddddeeeffggjjksss
str:9aabbccddddeeeffggjjksss
str:9aabbccddddeeffggjjksss
str:9aabbccddddeeffggjjksss
str:9aabbccddddeefggjjksss
str:9aabbccddddeefgjjksss
str:9aabbccddddeefgjksss
str:9aabbccddddeefgjksss
```

```
duplicateexists(): true
str:9aabbccddddeefgjkss
str:9aabbccddddeefgjkss
str:9aabbccddddeefgjkss
str:9aabbccddddeefgjkss
str:9abbccddddeefgjkss
str:9abccddddeefgjkss
str:9abcddddeefgjkss
str:9abcdddeefgjkss
str:9abcddeefgjkss
str:9abcddefgjkss
str:9abcddefgjkss
str:9abcddefgjkss
str:9abcddefgjkss
str:9abcddefgjkss
str:9abcddefgjks
str:9abcddefgjks
str:9abcddefgjks
str:9abcddefgjks
str:9abcddefgjks
str:9abcddefgjks
str:9abcdefgjks
str:9abcdefgjks
str:9abcdefgjks
str:9abcdefgjks
str:9abcdefgjks

------------------------------------------------------------------------
--------------
2 October 2017 - Placement New and Operator Overloading
------------------------------------------------------------------------
--------------
C++ provides mechanisms to override default storage allocation by
overloading operator new. There are two types of operator new(): Plain
overload and Placement Overload. Placement new supplies storage as
argument to operator new. Code example in code/placement_new.cpp
illustrates this
as below (this has been compiled to C++2017 standard). Older ways of
overriding pointer this have been described in comments. Recent compilers
do not allow direct *this overrides and prefer operator new. Operator new
facility is useful for writing new storage allocators and memory
debuggers which can instrument and bypass default memory allocation for
profiling. This example also explains the rvalue reference operator &&
for *this. Rvalue references alias the right side of an assignment while
Lvalue references(&) alias the left side of assignment.
====================================
auto allocation
====================================
this...0xffa08a90
overwriting this...
rvaluethis :0xffa08a90
====================================
operator new:
====================================
operator new overloaded and this is from a heap allocator
this...0x96a3e18
overwriting this...
rvaluethis :0x96a3e18
operator delete overloaded and this is freed to a heap allocator
====================================
```

```
placement operator new:
===================================
this...0xffa08aa8
overwriting this...
rvaluethis :0xffa08aa8
```

---

21 December 2017 - Rvalue References in C++ and Move semantics

---

Rvalue references were introduced in C++11 standard specification. Cloud move implementation of NeuronRain Neuro Currency applies the move semantics and rvalue references (client,server and header in https://github.com/shrinivaasanka/asfer-github-code/blob/master/cpp-src/cloud_move/). Traditionally lvalue refers to LHS of an assignment operator and rvalue to RHS of it. For example:

```
      int x=5
```

assigns rvalue 5 to lvalue x. Lvalue references are declared by alias operator & as:

```
      int& y=x
```

and Rvalue references are declared by && operator:

```
      int&& y=10
```

Move semantics in C++ specify moving an object by move constructor (std::move() and operator= overload) vis-a-vis copying an object by copy constructor. Move constructor is defined in Neuro currency as:

```
      T& operator=(cloudmove<T>&& rvalue) {
      ...
      }
```

and this move constructor is invoked by:

```
      cloudmove<currency::Currency> currency_src(&c1,"localhost");
      cloudmove<currency::Currency> currency_dest(&c2,"localhost");
      ...
      currency_dest = std::move(currency_src);
```

Move differs from Copy by returning rvalue of the argument to std::move() and renders the operand currency_src nullified by moving the resources to lvalue currency_dest.

---

7 August 2018 - Substring/Regular Expression Matcher

Matching a substring within a larger string is regular expression matching problem of writing a DFA. Deterministic Finite State Automatons are usually state transition tables on a graph. String is looped through and state transition table is looked up for next state till accept is reached. Designing this as a recursion saves lot of lines of code. An example recursive regexp substring matcher is in code/regexp.cpp which prints all matching positions of a substring as below:

```
:regexp matchks does not match at 0
 :regexp matchks does not match at 1
 :regexp matchks does not match at 2
 :regexp matchks does not match at 3
 :regexp matchks does not match at 4
 :regexp matchks does not match at 5
 :regexp matchks does not match at 6
 :regexp matchks does not match at 7
 :regexp matchks does not match at 8
```

```
 :regexp matchks does not match at 9
 :regexp matchks does not match at 10
 :regexp matchks matches at 11
 :regexp atchks matches at 12
 :regexp tchks matches at 13
 :regexp chks matches at 14
 :regexp hks does not match at 15
 :regexp matchks does not match at 16
 :regexp matchks does not match at 17
 :regexp matchks does not match at 18
 :regexp matchks does not match at 19
 :regexp matchks does not match at 20
 :regexp matchks matches at 21
 :regexp atchks matches at 22
 :regexp tchks matches at 23
 :regexp chks matches at 24
 :regexp hks matches at 25
 :regexp ks matches at 26
 :regexp s matches at 27
```

---

7 September 2018 - Unordered Map, Hash table buckets and Auto Iterator

---

C++ supports hashtables via unordered_map which is initialized either by emplace() or by {{...}} notation.
C++ from 2011 has new kind of iterators similar to Java 8 which automatically identify the type by auto keyword:

```
      auto& it: <container>
```

Bucket containing an entry in the map is accessed by bucket() member function. An example code which populates
an unordered_map by process-clockticks pairs, auto iterates them and prints the buckets is committed in:
code/unordered_map_auto_iter.cpp and logs are committed to code/logs/unordered_map_auto_iter.log.7September2018.

---

10 September 2018 - Unordered Map and for_each()

Previous example for auto iterator has been changed to iterate the unordered map by for_each() primitive
from <algorithm>. This is C++ equivalent of map() in python which invokes a function on each element of the
container. Unordered map has std::pair<> elements accessed by .first and .second members.

---

855. (THEORY and FEATURE) 24 September 2018,9 August 2020 - Fowler-Noll-Vo Hashing, Custom Hash Functions in unordered_map, Nested Template Classes - related to all sections on Locality Sensitive Hashing, Separate Chaining Bucketization

---

FNV or Fowler-Noll-Vo Hashing is a non-cryptographic hash algorithm which has high dispersion and minimizes collisions in same bucket. It iterates through literals in text and multiplies their unicode values by a prime

and XORs with an offset. This has avalanche effect - hash is very sensitive to small change in input. An example FNV implementation based on Boost C++ example has been added to course material at code/fnv.cpp. This defines a namespace class and nested fnv templatized struct through which prime number and offsets can be passed as arguments. FNV hashing is widely used in search engines, text processing, MS Visual Studio, memcache etc.,

References:

855.1.Boost FNV example - https://www.boost.org/doc/libs/1_68_0/libs/unordered/examples/fnv1.hpp
855.2.Fowler-Noll-Vo - FNV - Hashing: http://www.isthe.com/chongo/tech/comp/fnv/
855.3.Go Lang FNV package - https://golang.org/pkg/hash/fnv/

---
5 October 2018 - C++ Move-Assign Threads, Unordered Map Rehash and Concurrent Access
---
In C++ threads can be created in C++ specific move-assign paradigm which moves RHS thread object to LHS and
destroys LHS. Move-assign is done by std:thread() operator= overloaded function which takes thread function
and arguments to it as parameters. An example C++ source file threads.cpp has been committed in code/ which
creates 50 thread objects, move-assigns thread objects to them by invoking a function to populate an unordered
map. populate_hashmap() waits for few nanoseconds, makes a key-value pair and places them in unordered map.
Load factor (number of items/number of buckets ratio) is recomputed to by invoking max_load_factor() and
rehash() functions alternately for odd and even values. This is a contrived example to demonstrate concurrent
accesses to a container in C++. Logs for this example are in code/logs/threads.log.5October2018.

---
854. (ThEORY and FEATURE) 28 October 2018,9 August 2020 - Three Distances Theorem and Fibonacci Hashing - related to all sections on Locality Sensitive Hashing, Separate Chaining Bucketization
---
Three Distances Theorem - Proof of Steinhaus Conjecture:
If Phi=(sqrt(5)-1)/2, and sequences of points {Phi}, {2*Phi}, {3*Phi}, ... are plotted in [0...1] y-interval,and successive line segments are inserted in [0...1] y-interval from (k, {k*Phi}) to (n, {k*Phi}) the line segments are of sets of 3 lengths.[{k*Phi} is the fraction obtained subtracting the integer floor(k*Phi) from k*Phi]

An example C++ code which implements this as a hash function to an unordered_map has been described in code/threedistances.cpp.

Following are the size of each line segment sets grepped from log:

```
# grep "big " logs/threedistances.log.28October2018 |wc -l
68
```

```
# grep "bigger " logs/threedistances.log.28October2018 |wc -l
66
# grep "biggest " logs/threedistances.log.28October2018 |wc -l
66
```

References:

854.1.The Art of Computer Programming - Volume 3 - Sorting and Searching - Page 518 - [Don Knuth] - Proof of Steinhaus Conjecture - Theorem S - [Vera Turan Sos]

---
1 November 2018, 2 November 2018, 3 November 2018 - Polymorphism, RTTI, Pure Virtual Functions, Friend classes, Scope Resolution operator, protected and private members, Initializers in Constructors, const correctness
---
C++ specifies polymorphic classes by deriving a base super class by syntax:

```
      class <derived> : <qualifier> <super>
```

code example in code/polymorphism.cpp defines a base class Animal and 2 derived classes: Tiger and Lion.
Keyword protected in derived classes imply the derived class access to base class's protected members. Base
class Animal has a private member which is accessible by the derived classes through friend class declarations
in base class. There are two virtual functions in base class one of which is declared pure and makes Animal
an Abstract Data Type. Derived classes Tiger and Lion implement the pure virtual function in abstract base
class and override the other virtual function. Runtime Type Identification (RTTI) is from typeinfo infrastructureprovided by C++ standard for inferring the type of the object at runtime - typeid() keyword prints the typename
of the object. Constructor Initializers are mentioned by a list of variables suffixed by () operators and values assigned to private member variables. Const qualifier informs the compiler that the function should not alter
the variables (immutables).

Scope resolution operator :: resolves the private (by friendship) and protected members of the super class(by
protected derivative classing). Header cxxabi.h has been included for C++ ABI name demangling of RTTI typenames.
const disambiguation has been demonstrated by two functions legs() with const and without const qualifier. Both
legs() are invoked by base class pointer Animal* (->legs()) and as member invocation (.legs()) and difference
in behaviour is obvious from logs/polymorphism.log.3November2018.

References:
---
1. The C++ Programming Language - [Bjarne Stroustrup]
2. Essential C++ - C++ in depth series : Bjarne Stroustrup - [Stanley Lippman, Dreamworks] - const example - Section 5.9 - Page 161 - Previous example differs because of g++-6 idiosyncracy: const Animal* is required to invoke legs() having const qualifier.

---
29 November 2018 - Pointers and References (Lvalue and Rvalue)

An example C++ code for miscellaneous permutations of pointers and aliases usage has been committed at code/pointerstew.cpp. C++ pointers which are supersets of C pointers have additional facilities for aliasing to an object location in the form of right value references(&& operator) and left value references(& operator). References or aliases do not consume extra memory storage as opposed to pointers which are object memory locations themselves. Points-to and Reference-to graph of the variables declared in pointerstew.cpp is below (legend: pointer ######>, rvaluereferences: ==========>, lvaluereferences: ---------->):

```
           psptr ############> ps <========= psrvalueref

           pint1 ############> ps.rvaluerefx1 ====> ps.x
           pint2 ############> ps.lvaluerefx2 -----> ps.x
           func1() parameter y =========> forwarded rvalue of arg to
func1()
```

Assigning an lvalue to rvalue of same datatype throws following GCC error:

```
pointerstew.cpp: In function ‘int main()’:
pointerstew.cpp:33:28: error: cannot bind ‘pointerstew’ lvalue to
‘pointerstew&&’
  pointerstew&& psrvalueref=ps;
```

Logs for this example code have some surprising values for rvalue references. Assigning values directly to rvalue references corrupts the rvalue in GCC (shown in logs):

```
     int&& rvaluerefx1=1;
```

whereas std::Forward<int>(1) is required to forward the rvalue to lvalue for any assignment and across function invocations as parameters:

```
     int&& rvaluerefx1=std::forward<int>(x);
```

func1() has been overloaded with parameters and without them. Difference between effect of post-increment of rvalue (ps.rvaluerefx1++;) with and without std::forward() (previous two ways of initializing rvaluerefx1 within pointerstew object) is evident. One time std::move of rvalue and std::forward() of rvalue is demonstrated by static value of xx across multiple invocations in std::move() while rvalues always reflect x dynamically.

References:

1.C++ Programming Language - [Bjarne Stroustrup]
2.C Puzzles - Pointer Stew - [Alan Feuer]

---
856. (THEORY and FEATURE) 26 December 2018 - Bridge and Iterator Design Patterns - related to all sections on Software Analytics, Program Analysis, Survival Index Timeout and Scheduler of NeuronRain Theory Drafts
---
Bridge is a design pattern mentioned in Gang-of-Four catalog of C++ Design Patterns. Bridge separates implementations and the interfaces by

defining implementation itself as an abstract data type. Iterators are the patterns to enumerate iterable containers - arrays, hashmaps, linkedlists etc.,C++ code example bridgeiteratordesignpatterns.cpp which demonstrates how timeout pattern described in https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt fits as an amalgamation of Bridge and Iterator Design Patterns, has been committed in C++/code. Timeout is a dictionary of timeout values to lists of objects to timeout. Timeout as a pattern has universal occurrence across whole gamut of software engineering. Code example defines following classes - Timeout and TCPTimeout are interfaces and TimeoutImp and TCPTimeoutImp are implementations which are bridged by a pointer reference Timeout holds to TimeoutImp. This Decoupling is by passing any derivative TimeoutImp object in TCPTimeout constructor which in turns assigns to timeoutimp reference in Timeout. timeout() overridden virtual member function in TCPTimeout invokes imptimeout() unaware of implementation TimeoutImp :

Timeout ---------- implemented-by ---------- TimeoutImp (derived by TCPTimeoutImp)
(derived by TCPTimeout)

References:

856.1.Design Patterns - Elements of Reusable Object Oriented Software - [1995] - [Erich Gamma - Richard Helm - Ralph Johnson - John Vlissides] - Bridge and Iterator Patterns - Page 160 - Shared Strings Class - [Coplien] and [Stroustrup]

---
851. (THEORY and FEATURE) 12 February 2019, 9 August 2020 - Software Transactional Memory - related to Program Analysis, Software Analytics, Software Transactional Memory, Lockfree datastructures, Bakery Algorithm, Read-copy-update sections of NeuronRain Theory Drafts
---
Software Transactional Memory is supported by C++ by synchronized blocks of compound statements and transaction_safe directive in function declaration. Software Transaction Memory is the intrinsic facility for transactional rollback or commit of set of statements similar to RDBMS - either all are executed or none. An example transactional memory code has been committed to code/softwaretransactionalmemory.cpp. It declares two functions - function1() executing a synchronized block and function2() declared transaction_safe which is a tighter restriction preventing unsafe code. Compiler error flagged for unsafe function calls have been added in code comments. Curious statement in the code is:

```
t=std::thread([]{for(int n=0; n < 10;n++) function1(n);});
```

which is a lambda expression doing null capture by [] operator and just invoking the function function1() within lambda expression loop block. Auto iterator variable t is assigned to the thread object. Logs in logs/softwaretransactionalmemory.log.12February2019 show serialized execution of 10 threads instantiated. VIRGO32 and VIRGO64 kernels implement a Bakery algorithm locking primitive as kernel driver while in userspace VIRGO system calls could be wrapped by C++ transaction memory primitives.

References:

851.1.C++ Lambda Expressions -
https://en.cppreference.com/w/cpp/language/lambda
851.2.C++ Software Transactional Memory -
https://en.cppreference.com/w/cpp/language/transactional_memory

---
---
15 February 2019 - Lambda Functions and Capture, Functional Programming - std::function
---

C++ supports lambda functional programming constructs similar to other languages like Python and Java.
An example C++ code which dynamically creates lambda functions and returns them is shown in code/lambdafunctions.cpp. It defines a struct and member function dynamicfunctions() which populates an unordered_map of string-to-int by parameters defined by () operator. It also captures this object from its present scope by [] operator which is internally accessed by on-the-fly lambda function block for the hashmap member. Capturing is intended for data communication between lambda function block and external scope. Member function dynamicfunctions() is returned as std::function object function1 returning int and taking (string,int) as arguments. Returned dynamic function object function1 is invoked like any other function by passing (string,int) arguments twice. Resultant hashmap is printed by auto iterator. Logs for this are committed to logs/lambdafunctions.log.15February2019.

---
852. (THEORY and FEATURE) 23 February 2019,9 August 2020 - Concurrency, Promise and Future Asynchronous I/O - related to Program Analysis, Software Analytics, Software Transactional Memory, Lockfree datastructures, Bakery Algorithm, Read-copy-update, Drone Autonomous Delivery sections of NeuronRain Theory Drafts
---

C++ facilitates asynchronous communication between concurrent threads by Promise and Future. Promise is instantiated by std::promise template and passed on as arguments to threads similar to shared_ptr which are shared mutables within thread functions. Future associated to Promise is acquired at a later time point asynchronously and value set by threads is readable. Code example at code/promisefuture.cpp describes two fictitious train threads having access to Promise and sets the nanoseconds time duration between present and an epoch as its value. std::chrono high resolution clock is invoked for time duration in nanoseconds. Future value for this Promise is later read by get() on Future object. Logs are shown in logs/promisefuture.log.23February2019. C++ SDK asynchronous I/O code for Drone telemetry could be augmented by Promise and Future code blocks.

---
853. (THEORY and FEATURE) 6 February 2020, 9 August 2020 - Read-Copy-Update mentioned in VIRGO Design Document of NeuronRain Theory Drafts has been implemented in userspace - related to Program Analysis, Software Analytics, Software Transactional Memory, Lockfree datastructures, Bakery Algorithm, Read-copy-update sections of NeuronRain Theory Drafts
---

Read-Copy-Update (RCU) has been mentioned as a feature in VIRGO32 and VIRGO64 design documents. Read-Copy-Update which is an efficient synchronization primitive implemented in most OS kernels works quite similar to local CPU caches of global RAM memory:

(*) READ - Read the variable
(*) COPY - Copy it to a temporary variable
(*) UPDATE - Update the temporary variable
(*) WRITEBACK - Write back temporary variable to the actual source

Advantage of Read-Copy-Update is the lack of necessity of mutexes:

(*) multiple concurrent readers have access to an older version of variable while a writer updates the copy of it
(*) older version of the variable is updated by new after all existing reads of older versions are done and no new read is allowed.
(*) older version is updated by the new version of the writer's working copy.

All the previous steps require no synchronization though it has to be ensured no new reads are performed while older version is updated. This kind of lockfree synchronization is quite useful for multiple readers of a linked list while some writer removes an element of the linked list. [Example of such a necessity is the WCET EDF Survival Index Scheduler design in GRAFIT course material - https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt - which is a set partition of linked list of process id(s) where frequently process id(s) are read by almost every component of OS kernel and deleted by scheduler]. Writer marks the node to delete and updates the links bypassing the deleted node. Thus both node pointed to by new link and deletion-marked node of the linked list are available to existing readers. After all existing queued reads are over, node marked for deletion is really deleted. Efficiency stems from the fact that no locks are necessay for concurrent RCU.

Code example in code/readupdatecopy.cpp implements a C++ class and wraps the RCU assign functionality as its operator= overloaded member function. As evident from example synchronized blocks for software transactional memory and mutexes have been commented. It can be compiled by commandline - g++ -g readcopyupdate.cpp -fgnu-tm -lpthread -o readcopyupdate. Three kinds of copy have been shown - invoking operator=, copy assign of RCU object and copy assign of members. Third clause for copy assign of member has the following schematic:

```
                        valuecopy=value; //READ and COPY
                                        cout<<"COPY:
valuecopy="<<valuecopy<<endl;
                                        valuecopy=rvalue.value; //UPDATE
                                        cout<<"UPDATE: lvaluecopy
updated="<<valuecopy<<endl;
                                        value=valuecopy; //WRITEBACK

                        ==================

                        value
                        | (READ)
                        V
                   (COPY) valuecopy <- rvalue.value (UPDATE)
                        |
                        V
                        value (WRITEBACK)
```

Logs in code/logs/readcopyupdate.log.6February2020 demonstrate a concurrent read-write by RCU of 200 threads. This code example is in effect a userspace implementation of RCU in VIRGO linux kernel (and is a spillover code of VIRGO repositories in GRAFIT) because reads and writes in thread functions can be replaced by syscall invocations of virgo_get() and virgo_set() preceded and succeeded by a global virgo_malloc() and virgo_free() respectively and no kernel codechange is necessary.

References:

853.1.Read-Copy-Update - https://en.wikipedia.org/wiki/Read-copy-update

---
857. (THEORY and FEATURE) 28 April 2020 - Name filter - C++ STL containers and algorithms - copy,copy_if,shared_ptr,tokenizer - related to all sections on People Analytics, Named Entity Recognition, Name filters (learning proper nouns in a text)

C++ Standard Templates Library implements algorithms for manipulating containers - for copying,filtering
and erasure. A contrived example C++ name filter class has been defined in namefilter.cpp which accepts a textfile and parses it to filter the words having a substring name pattern. An example list of names from linkedin profile of author is namefiltered for a certain prefix. Lines are tokenized by stringstream iterator and copied to a vector by copy(). Name filter is done twice - for non-zero length of strings in lambda function capture of copy_if() and in auto iterator loop by find() of the pattern. Shared pointers are C++ STL facility for refcounted pointers. Wordcount of the strings containing pattern is incremented via a shared_ptr. Arbitrary filtering implementation can be plugged-in to lambda function capture block of copy_if() - Most names are of persons,organizations,locations and namefiltering or proper noun extraction has multiple solutions:

(*) NER PoS tagging by Conditional Random Fields(Supervised-costly-requires culture neutral training corpora e.g https://www.aclweb.org/anthology/P95-1032.pdf)

(*) Natural Language Dictionary or Ontology lookup(Unsupervised-preferred-no training data-if word is not in dictionary or semantic network - WordNet,ConceptNet,NameNet - https://pdfs.semanticscholar.org/56f9/cf53333a46c9ea355578f6b7b9424a4737e2.pdf - it is most likely a proper noun) are some of them. NeuronRain AstroInfer People Analytics implements dictionary filter.

---
1 November 2020, 2 November 2020 - Mediator-Colleague Design Pattern - C++ example
---
Mediator Design Pattern encapsulates the set of colleague objects and a director object which mediates the interactions between colleague objects. Colleagues do not interact among themselves directly but are moderated by the mediator object. C++ code example mediatordesignpattern.cpp implements two classes for director-mediator and colleagues - colleague objects invoke the singleton mediator for interaction amongst them and do not communicate with each other. Such a pattern is necessary in GUI event oriented programming - set of widgets

which do not know each other notify a mediator about an event and mediator acts accordingly issuing further directives.

References:

1..Design Patterns - Elements of Reusable Object Oriented Software - [1995] - [Erich Gamma - Richard Helm - Ralph Johnson - John Vlissides] - Mediator Behavioural Pattern - Page 273

---
25 November 2020 - Reference wrappers, Arrays of references, Array move, C++ Array Objects, Reference to C++ Array object, Array Rotation
---
Code example arraymove.cpp demonstrates the following on C++ bounded array objects:

(*) Define a primitive integer array
(*) Perform memmove() on elements of integer array
(*) Instantiate a vector from integer array (copies array to a vector)
(*) Rotate the vector data and print them (would not affect source array)
(*) Define & alias operator to std::string type as reference_wrapper<string> object - from <functional> library
(*) Define array object of reference_wrapper<string> objects
(*) Invoke a function and pass bounded array object of reference_wrapper<string> objects by reference to function - (&array)[length]
(*) Perform memmove() on elements of array object of reference_wrapper<string> objects
(*) auto iterate the memmove()-ed integer array object and reference_wrapper<string> array objects

References:

1.Clockwise-Spiral rule for C type inference - http://c-faq.com/decl/spiral.anderson.html
2.C++ Reference wrapper - https://en.cppreference.com/w/cpp/utility/functional/reference_wrapper
3.C++ Rotate - https://en.cppreference.com/w/cpp/algorithm/rotate