

# Great Ideas in Computing

University of Toronto CSC196  
Fall 2025

Class 4: September 15 (2025)

# Announcements and Agenda

## Announcements

- Change of schedule for next week. On Wednesday, September 24, Tristan will do the tutorial and I will do the class on Friday the 26th.
- By this Wednesday, I will post the third and final question for the first Assignment.
- Our first guest lecture will be given by Professor Nathane Wiebe on the subject of quantum computers.

## Today's Agenda

- A quick discussion of the response of ChatGPT for my question "Will ChatGPT make Wikipedia Obsolete?"
- The memory hierarchy and memory management.
- A brief mention of data flow, multi-core and parallel computation.
- Data types
- The Dictionary Data Type
- If time remains, that will lead us to different data structures for the dictionary data type.

# Will Large Language Models (Chat-GPT) make Wikipedia obsolete?

As I have indicated, I am generally very impressed with Wikipedia when I need to get some specific non-controversial information.

But why not get the same information from say Chat-GPT and thereby make Wikipedia obsolete?

# Will Large Language Models (Chat-GPT) make Wikipedia obsolete?

As I have indicated, I am generally very impressed with Wikipedia when I need to get some specific non-controversial information.

But why not get the same information from say Chat-GPT and thereby make Wikipedia obsolete?

I decided to ask ChatGPT what it “thought”

## The fall 2023 answer from ChapGPT3.5

Chat-GPT and similar AI models are unlikely to make Wikipedia obsolete. While these AI models can generate text and provide information on a wide range of topics, there are several reasons why Wikipedia will likely continue to be a valuable resource:

**Human Verification and Quality Control:** Wikipedia relies on a large community of human editors who review and verify the content. These editors ensure that the information presented on Wikipedia is accurate, neutral, and reliable. AI models like Chat-GPT, while powerful, can generate incorrect or biased information, and they lack the ability to fact-check or verify the quality of their content.

**Structured and Organized Information:** Wikipedia organizes information in a structured manner, with articles, sections, citations, and references. It provides a clear and organized format for users to access information. Chat-GPT, on the other hand, generates text in a conversational format, which may not be as structured or easy to navigate.

**Community Engagement:** Wikipedia encourages community engagement and collaboration. People from around the world contribute to Wikipedia articles, add references, update information, and discuss content on talk pages. This collective effort ensures that Wikipedia remains a dynamic and up-to-date resource.

**Comprehensive Coverage:** Wikipedia covers a vast array of topics, from historical events and scientific concepts to biographies of individuals and cultural references. While AI models like Chat-GPT have access to a broad range of information, they may not provide the same depth and breadth of coverage that Wikipedia offers.

**Multilingual Support:** Wikipedia is available in multiple languages and is a valuable resource for people worldwide. While AI models can generate text in various languages, the quality and accuracy of translation and language nuances can be challenging for AI to handle effectively.

**Educational Value:** Wikipedia is often used as an educational tool in schools and universities. It provides students with a platform for research and learning. AI models like Chat-GPT may be used as a complement to educational resources, but they are not a substitute for comprehensive educational materials.

**Curation and Citations:** Wikipedia emphasizes the importance of citing reliable sources. Information on Wikipedia articles is backed by citations to reputable publications and references. AI models do not have the capability to cite sources or provide the same level of rigor in referencing.

In summary, AI models like Chat-GPT can be valuable for generating text and providing information, but they are not a replacement for the unique qualities and human-driven aspects of Wikipedia. Wikipedia's collaborative and structured approach, along with its emphasis on accuracy and quality control, makes it a valuable resource that is likely to continue to coexist with AI-generated content.

---

# The fall 2025 answer from ChapGPT

The answer is very similar but still a little different. The conclusion (today) is basically the same as in the response two years ago:

The ChatGPT conclusion: ChatGPT may outcompete Wikipedia as an interface, but not as an institution of knowledge curation and verification.

ChatGPT gives the following reasons and comparisons differences between itself and Wikipedia

- 1 Authority and Verification
- 2 Scope and Depth
- 3 Trust and reliability
- 4 Complementary Roles
- 5 Long Term Evolution.

Will ChatGPT make Wikipedia Obsolete: What do you think?



# The fall 2025 answer from ChapGPT

The answer is very similar but still a little different. The conclusion (today) is basically the same as in the response two years ago:

The ChatGPT conclusion: ChatGPT may outcompete Wikipedia as an interface, but not as an institution of knowledge curation and verification.

ChatGPT gives the following reasons and comparisons differences between itself and Wikipedia

- 1 Authority and Verification
- 2 Scope and Depth
- 3 Trust and reliability
- 4 Complementary Roles
- 5 Long Term Evolution.

Will ChatGPT make Wikipedia Obsolete: What do you think?

Does ChatGPT have an incentive to not be truthful in how it is answering you?

# Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is “addressable”. We conceptually think of each word being accessed (read or written) at some unit (of time) cost.

Is this how your lap top memory is organized?

# Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is “addressable”. We conceptually think of each word being accessed (read or written) at some unit (of time) cost.

Is this how your lap top memory is organized?

Take a look at the specifications of your lap top and you will almost surely see different types of memory. This is called the memory hierarchy. Lets just consider a two level hierarchy, where the fast memory might be called the cache and a slower memory which we can just call the “main memory”. (There can be more than two levels in the memory hierarchy including external memory which still might be addressable.)

# Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is “addressable”. We conceptually think of each word being accessed (read or written) at some unit (of time) cost.

Is this how your lap top memory is organized?

Take a look at the specifications of your lap top and you will almost surely see different types of memory. This is called the memory hierarchy. Lets just consider a two level hierarchy, where the fast memory might be called the cache and a slower memory which we can just call the “main memory”. (There can be more than two levels in the memory hierarchy including external memory which still might be addressable.)

The cache is a more costly, relatively small but a fast memory and main memory is considerably larger and slower in terms of accessing time. So, in fact, different accesses can take different amounts of time. However ....

## The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting “locality of reference” so that it is likely that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a **page**) at the same time.

The following **Least Recently Used (LRU)** algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

# The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting “locality of reference” so that it is likely that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a **page**) at the same time.

The following **Least Recently Used (LRU)** algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

Why not just have fast memory?

# The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting “locality of reference” so that it is likely that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a **page**) at the same time.

The following **Least Recently Used (LRU)** algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

**Why not just have fast memory?**

In computing (and life) there are almost inevitably tradeoffs to be made.

# The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting “locality of reference” so that it is likely that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a **page**) at the same time.

The following **Least Recently Used (LRU)** algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

## Why not just have fast memory?

In computing (and life) there are almost inevitably tradeoffs to be made.

The important thing is that you (the algorithm designer) usually do not need to know how this memory hierarchy is managed; that is, when data is brought into the cache and when it is ejected from the cache. This is referred to as caching or sometimes as paging.

Memory management also makes it possible to have several processes sharing the main memory.



## A little more about memory management

Another aspect of memory management is allocating memory to arrays and other data structures. For example, when we declare (in a programming language) that  $A$  is an array of some fixed size, for efficiency we want to allocate consecutive blocks of memory words. (We this lead us to discuss the idea of data types and data structures.)

### A False Economy

An algorithm designer may think they know their application and can better optimize storage management for their application.

Why is this a false economy?

This is similar to the false economy of programming an algorithm in the machine code (of a specific machine).

I think virtual memory (and other forms of “virtualization”) is a great idea.

# Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.

Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!

It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

# Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.

Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!

It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

Why is the von Neumann model such a great idea?

# Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.

Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!

It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

Why is the von Neumann model such a great idea?

Are we stuck in a "von Neumann tarpit?"

# The benefits of a well agreed upon abstract model of computation

One of the main reasons to consider the von Neumann model a great idea is that by being a well agreed upon model, coordination amongst different people is minimized. That is,

- A computer architect doesn't need to know which programming languages will be run on their specific architecture. The abstract von Neumann model doesn't specify the instruction set, the memory management, how interrupts are handled, etc. This is what the architects do.
- A compiler writer for a programming language  $\mathcal{L}$  doesn't have to know what algorithms will be implemented using the language  $\mathcal{L}$ .
- Without complete knowledge of the architecture, and the compiler, the algorithm designer can make a rough approximation for the memory and time requirements of their algorithm.

Progress in parallel computation had been relatively slow but there is now some common approaches (e.g., MapReduce for large scale parallel computation).

# Dataflow architecture

Direct from Wikipedia:

Dataflow architecture is a computer architecture that directly contrasts the traditional von Neumann architecture or control flow architecture. Dataflow architectures do not have a program counter (in concept): the executability and execution of instructions is solely determined based on the availability of input arguments to the instructions,[1] so that the order of instruction execution is unpredictable, i.e. behavior is *nondeterministic*. (My emphasis to indicate that event are not following some deterministic or well understaood random distribution.)

Although no commercially successful general-purpose computer hardware has used a dataflow architecture, it has been successfully implemented in specialized hardware such as in digital signal processing, network routing, graphics processing, telemetry, and more recently in data warehousing.[citation needed] It is also very relevant in many software architectures today including database engine designs and parallel computing frameworks.[citation needed]

## More on data flow architecture

From J. Paul Morrison's Flow-Based Programming text

The von Neumann machine is perfectly adapted to the kind of mathematical or algorithmic needs for which it was developed: tide tables, ballistics calculations, etc., but business applications are rather different in nature. . . .

Business programming works with data and concentrates on how this data is transformed, combined, and separated. . . . Broadly speaking, whereas the conventional approaches to programming (referred to as “control flow”) start with process and view data as secondary, business applications are usually designed starting with data and viewing processes as secondary—processes are just the way data is created, manipulated, and destroyed. We often call this approach “data flow.” (21)

# Multicore and Parallel Computation

- As you may already know, computers today are often multicore machines meaning that some “small” constant number of processes can be running simultaneously.
- When people refer to large scale parallelism they have in mind that the number of processes running in parallel can depend (at least conceptually) on the computation.
- The von Neumann architecture is an abstract model for *sequential computation*.
- In contrast to the well accepted von Neumann model for sequential computation, the situation for *parallel computation* is more nuanced.
- There is the issue of a constant number of parallel processes vs a number of processes that depends on the size of the data and/or the computation as it evolves.
- Do the processes run synchronously (i.e. according to some global clock) or asynchronously?
- Do the processes communicate mainly through a shared memory or via some communication bus?
- How do we maintain consistency of the information being shared?



# Data Types

Let's give an informal definition of what we mean by a data type. Namely, a data type is a collection of items (data) and the allowable operations, relations and queries involving those items. So as an example we can have a data type called Float where the data is numbers represented in floating point representation, the operations are the standard arithmetic operations  $+$ ,  $-$ ,  $*$ ,  $\div$ , exponentiation and perhaps logarithms. We also have the relations  $<$ ,  $=$ ,  $>$ .

You can check how Wikipedia states what is a data type.

We will next introduce the Dictionary data type.

## Looking up a record

Suppose with every person in an organization we have information stored in some “device”. It could be an old printed telephone directory, a folder in a physical cabinet, or a file in a cell phone or computer.

Lets think about how it could be stored in a von Neuman architecture type computer in analogy to a file cabinet. Namely, think about one folder placed after the other. And for simplicity, say we have the same amount of information on each person.

In a computer one way to do this is to think of each person taking up some  $p$  consecutive words in memory (i.e., an array), one word for the name of the person, and  $p - 1$  words for the information. The information about a person can be called a *record*. If there are  $n$  people in the organization then we would be taking up  $n \cdot p$  words of memory if we stored this information in an array.

Instead of the name of the person we could have some other (unique) identifier (e.g., their social insurance number).

# Dictionaries

What are the most basic operations we want to associate with such a collection of information?

# Dictionaries

What are the most basic operations we want to associate with such a collection of information?

- *Search*: Look up if someone is in the organization and if so retrieve the information for this person.
- *Update content*: Change the information regarding an item
- *Insert*: Add a new person to the organization
- *Delete*: Remove a person from the organization.

Sets of objects with these operations are referred to as a *Dictionary* data type. It is a static dictionary if we only want to look up and possibly modify records and a dynamic dictionary if we also want to add and delete. We can use different *data structures* to implement such a data type.

There can be many more operations that we want to perform on collections of data. More generally how one maintains and operates on data is known as the field of *data bases*. Analyzing data and extracting new (often statistical) information from collections of data is now called *data science* or *data analytics*. More ambitious learning of new information from data can be called *machine learning*.

## End of 4th class

We ended having just introduced the dictionary data type and we were discussing how we (as humans) might go about looking some one up in a file.

We got suggestions for different ways we might do this. Giving names to these suggestions, linear and binary search were suggested.