Welcome to Data Structures(ECE20010/ITP20001)

Youngsup Kim
idebtor@gmail.com
Handong Global University

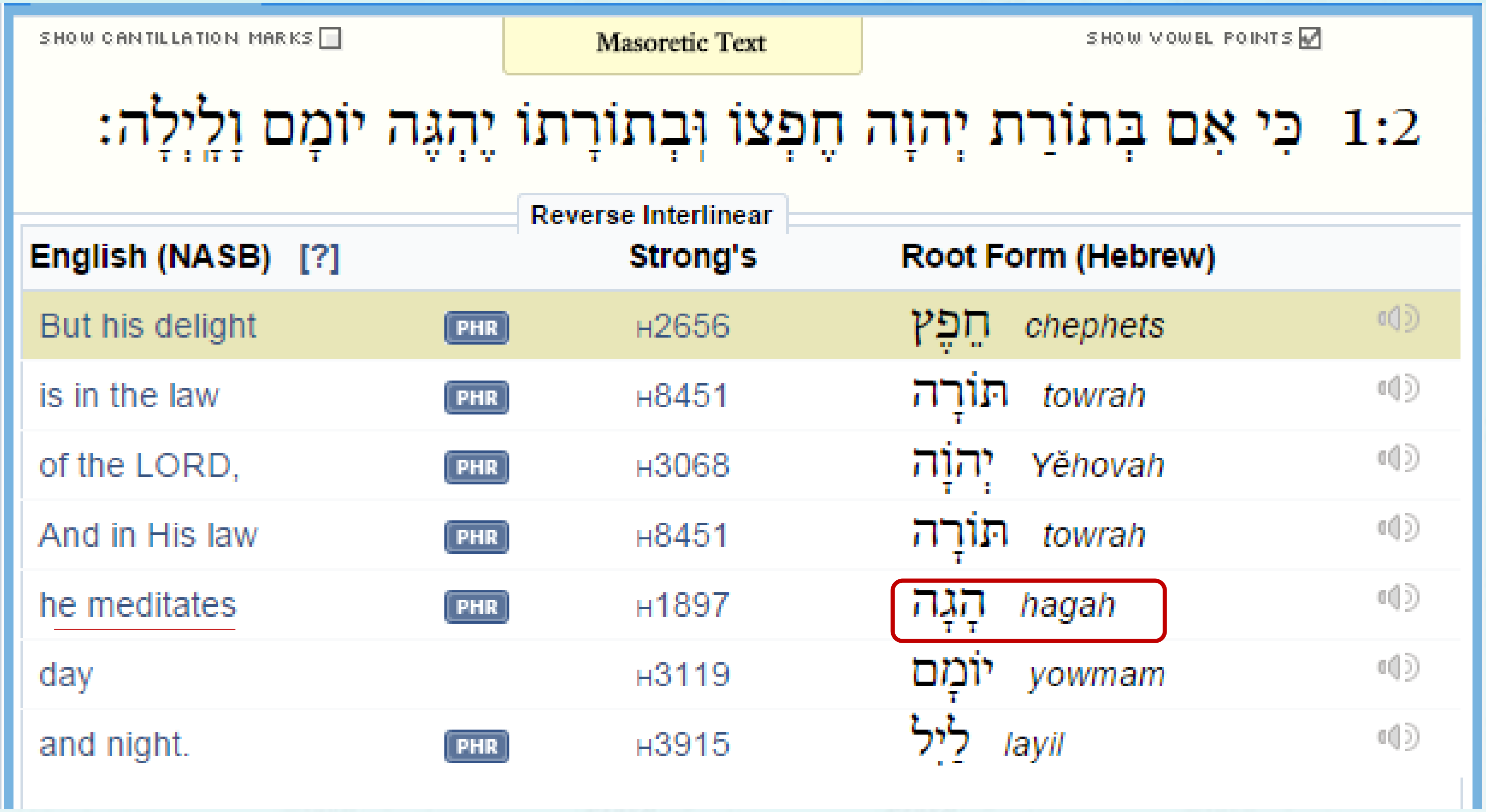
Data Structures

- *overview*
    - *why DS **in C**?*
    - *syllabus*
- *why study data structures?*
- *resources*

Youngsup Kim
idebtor@handong.edu
Handong Global University

[시1:1-2] **복 있는 사람**은 악인들의 꾀를 따르지 아니하며 죄인들의 길에 서지 아니하며 오만한 자들의 자리에 앉지 아니하고, 오직 여호와의 율법을 즐거워하여 그의 율법을 주야로 묵상하는도다

(Psalm1:1-2) **Blessed is the one** who does not walk in step with the wicked or stand in the way that sinners take or sit in the company of mockers, but whose delight is in the law of the LORD, and who meditates on his law day and night.



SHOW CANTILLATION MARKS □ Masoretic Text SHOW VOWEL POINTS ☑

כִּי אִם בְּתוֹרַת יְהוָה חֶפְצוֹ וּבְתוֹרָתוֹ יֶהְגֶּה יוֹמָם וָלָיְלָה׃ 1:2

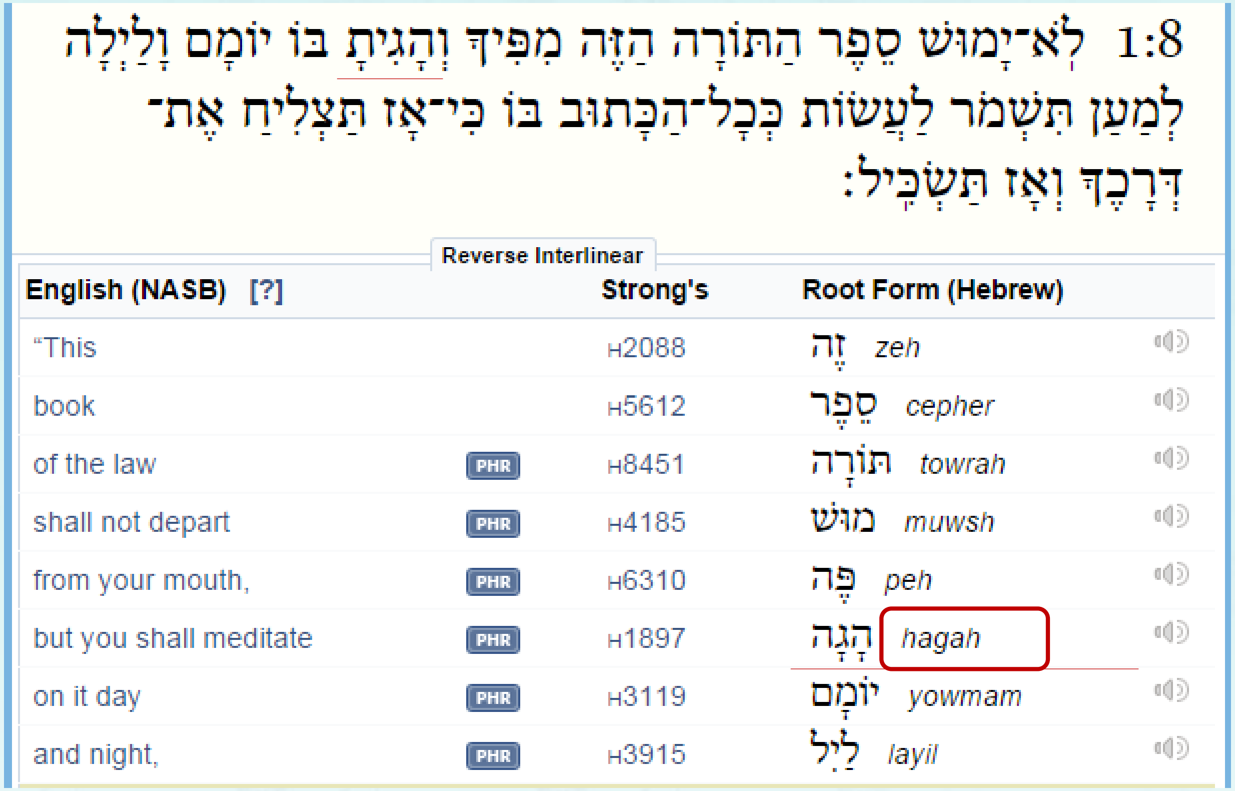
Reverse Interlinear

English (NASB) [?]		Strong's	Root Form (Hebrew)	
But his delight	PHR	H2656	חֵפֶץ	*chephets*
is in the law	PHR	H8451	תּוֹרָה	*towrah*
of the LORD,	PHR	H3068	יְהֹוָה	*Yĕhovah*
And in His law	PHR	H8451	תּוֹרָה	*towrah*
he meditates	PHR	H1897	הָגָה	*hagah*
day		H3119	יוֹמָם	*yowmam*
and night.	PHR	H3915	לַיִל	*layil*

[시1:1-2] **복 있는 사람**은 악인들의 꾀를 따르지 아니하며 죄인들의 길에 서지 아니하며 오만한 자들의 자리에 앉지 아니하고, 오직 여호와의 율법을 즐거워하여 그의 율법을 주야로 **묵상하는도다**

(Psalm1:1-2) **Blessed is the one** who does not walk in step with the wicked or stand in the way that sinners takes or sit in the company of mockers, but whose delight is in the law of the LORD, and who **meditates** on his law day and night.

묵상
하가



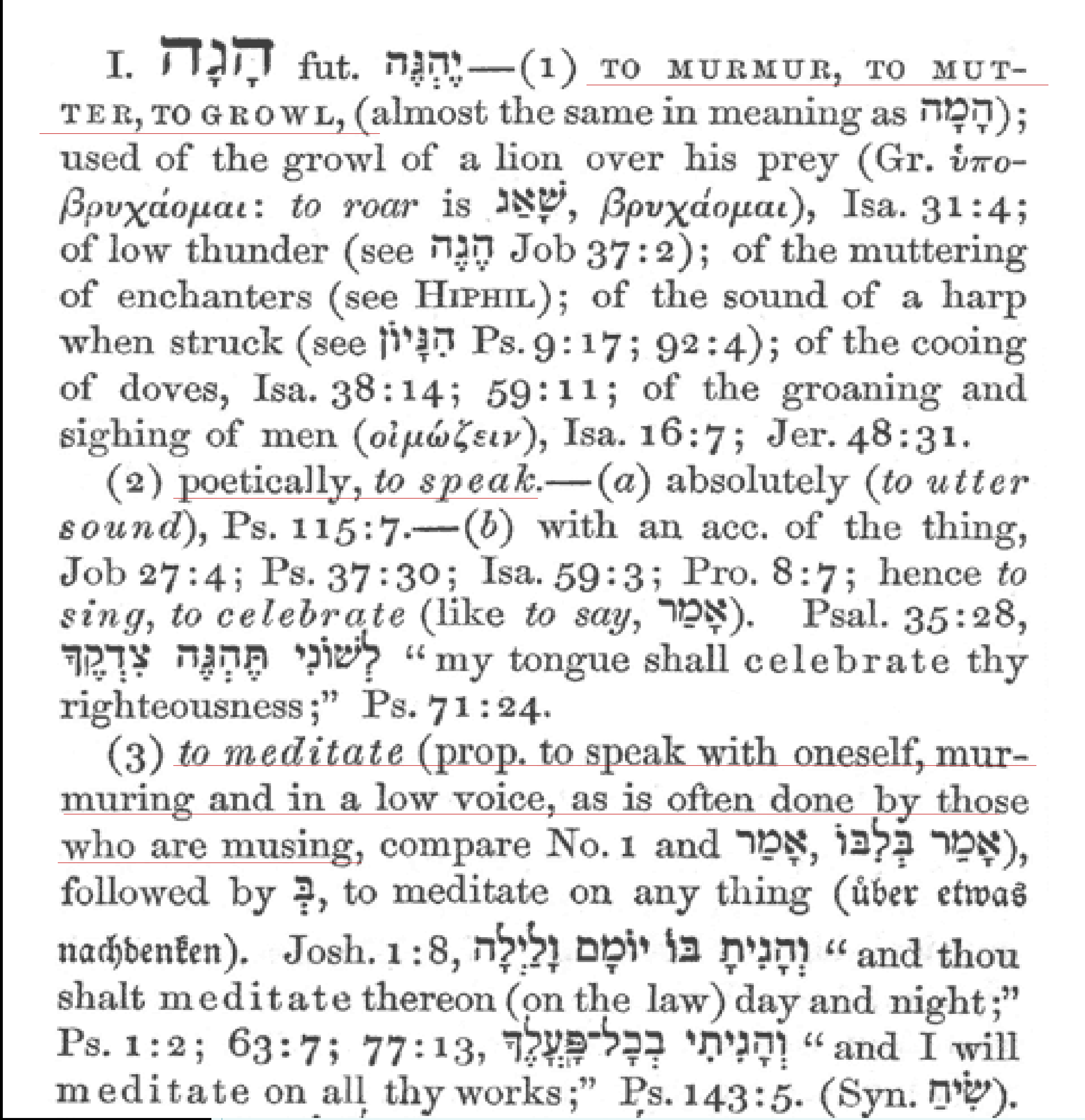
1:8 לֹֽא־יָמוּשׁ סֵפֶר הַתּוֹרָה הַזֶּה מִפִּיךָ וְהָגִיתָ בּוֹ יוֹמָם וָלַיְלָה
לְמַעַן תִּשְׁמֹר לַעֲשׂוֹת כְּכָל־הַכָּתוּב בּוֹ כִּי־אָז תַּצְלִיחַ אֶת־
דְּרָכֶךָ וְאָז תַּשְׂכִּיל׃

Reverse Interlinear

English (NASB) [?]		Strong's	Root Form (Hebrew)
"This		H2088	זֶה *zeh*
book		H5612	סֵפֶר *cepher*
of the law	PHR	H8451	תּוֹרָה *towrah*
shall not depart	PHR	H4185	מוּשׁ *muwsh*
from your mouth,	PHR	H6310	פֶּה *peh*
but you shall meditate	PHR	H1897	הָגָה *hagah*
on it day	PHR	H3119	יוֹמָם *yowmam*
and night,	PHR	H3915	לַיִל *layil*

[수1:8] 이 율법책을 **네 입에서 떠나지 말게 하며 주야로 그것을 묵상하여** 그 안에 기록된 대로 다 지켜 행하라 그리하면 네 길이 평탄하게 될 것이며 네가 형통하리라

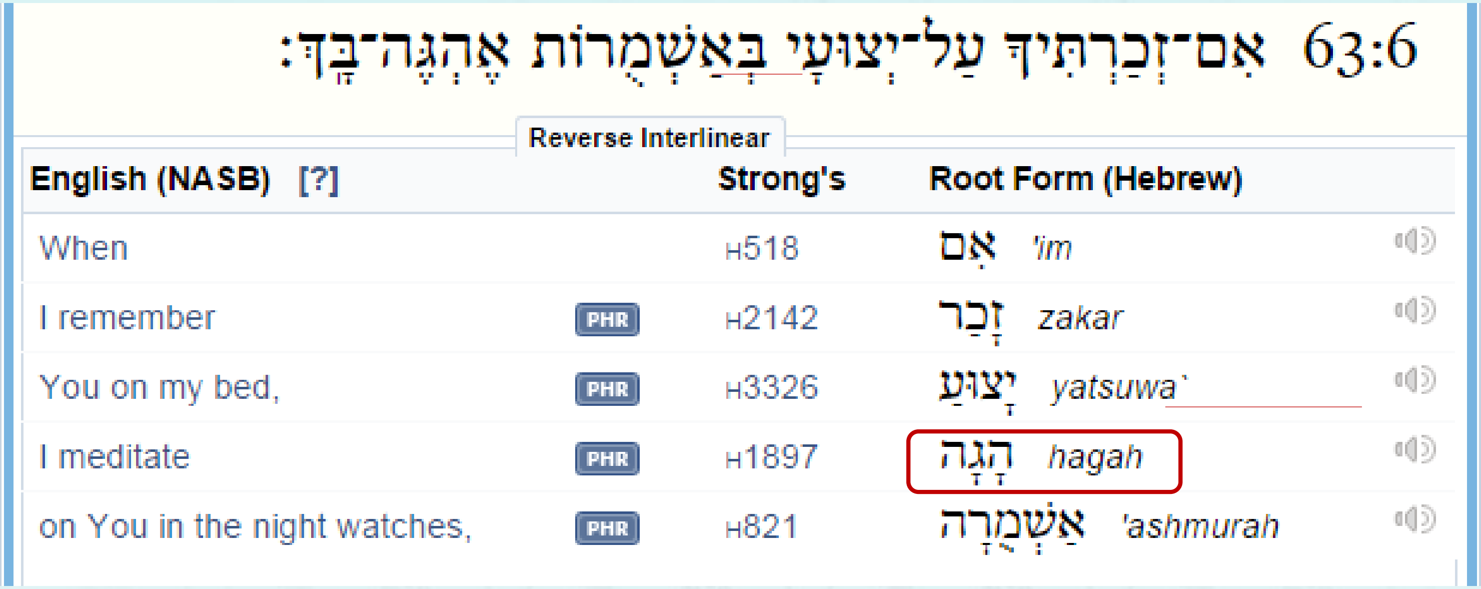
Keep this Book of the Law always on your lips; **meditate on** it day and night, so that you may be careful to do everything written in it. Then you will be prosperous and successful.



I. הָגָה fut. יֶהְגֶּה—(1) TO MURMUR, TO MUTTER, TO GROWL, (almost the same in meaning as הָמָה); used of the growl of a lion over his prey (Gr. *ὑποβρυχάομαι*: *to roar* is שָׁאַג, *βρυχάομαι*), Isa. 31:4; of low thunder (see הֶגֶה Job 37:2); of the muttering of enchanters (see Hiphil); of the sound of a harp when struck (see הִגָּיוֹן Ps. 9:17; 92:4); of the cooing of doves, Isa. 38:14; 59:11; of the groaning and sighing of men (*οἰμώζειν*), Isa. 16:7; Jer. 48:31.

(2) poetically, *to speak*.—(*a*) absolutely (*to utter sound*), Ps. 115:7.—(*b*) with an acc. of the thing, Job 27:4; Ps. 37:30; Isa. 59:3; Pro. 8:7; hence *to sing, to celebrate* (like *to say*, אָמַר). Psal. 35:28, לְשׁוֹנִי תֶּהְגֶּה צִדְקֶךָ "my tongue shall celebrate thy righteousness;" Ps. 71:24.

(3) *to meditate* (prop. to speak with oneself, murmuring and in a low voice, as is often done by those who are musing, compare No. 1 and אָמַר, אָמַר בְּלִבּוֹ), followed by בְּ, to meditate on any thing (über etwas nachdenken). Josh. 1:8, וְהָגִיתָ בּוֹ יוֹמָם וָלַיְלָה "and thou shalt meditate thereon (on the law) day and night;" Ps. 1:2; 63:7; 77:13, וְהָגִיתִי בְכָל־פָּעֳלֶךָ "and I will meditate on all thy works;" Ps. 143:5. (Syn. שִׂיחַ).



63:6 אִם־זְכַרְתִּיךָ עַל־יְצוּעָי בְּאַשְׁמֻרוֹת אֶהְגֶּה־בָּךְ׃

Reverse Interlinear

English (NASB) [?]		Strong's	Root Form (Hebrew)	
When		H518	אִם	'im
I remember	PHR	H2142	זָכַר	zakar
You on my bed,	PHR	H3326	יָצוּעַ	yatsuwa`
I meditate	PHR	H1897	הָגָה	hagah
on You in the night watches,	PHR	H821	אַשְׁמֻרָה	'ashmurah

[시63:6] 내가 나의 침상에서 주를 기억하며 새벽에 주의 말씀을 작은 소리로 읊조릴 때에 하오리니

(Psalm63:6) When I remember thee upon my bed, *and* **meditate on** thee in the *night* watches.

Course overview

What does the data structure mean?

- **Data structures**:
  - **methods to store and organize** [redacted] in a computer so that it can be used efficiently.
  - A key to designing efficient [redacted]

Course overview

What does the data structure mean?

- **Data structures**:
  - **methods to store and organize data** in a computer so that it can be used efficiently.
  - A key to designing efficient **algorithms**.
- **Algorithms**:
  - methods for solving a problem

Course overview

What does the data structure mean?

- **Data structures**:
  - **methods to store and organize data** in a computer so that it can be used efficiently.
  - A key to designing efficient **algorithms**.
- **Algorithms**:
  - methods for solving a problem
- **Data structures &algorithms** are the fundamentals of programming.
  - To become a good computer scientist or engineering it is essential to master the **data structures and algorithms** and learn to apply them to the real world problems.

which is complicated or complex.

Course overview

What is this course?

- Intermediate-level course.
- Programming **after** programming for problem solving with applications.

Course overview

What is this course?

- Intermediate-level course.
- Programming **after** programming for problem solving with applications.

topic	data structures and algorithms
concepts	algorithms, time-complexity, array and structure

Course overview

What is this course?

- Intermediate-level course.
- Programming **after** programming for problem solving with applications.

topic	data structures and algorithms
concepts	algorithms, time-complexity, array and structure
data types	linked list, array, stack, queue, trees, union-find, bag, priority queues

Course overview

What is this course?

- Intermediate-level course.
- Programming **after** programming for problem solving with applications.

topic	data structures and algorithms
concepts	algorithms, time-complexity, array and structure
data types	linked list, array, stack, queue, trees, union-find, bag, priority queues
sorting	selection sort, quick sort, merge sort, heap sort

Course overview

What is this course?

- Intermediate-level course.
- Programming **after** programming for problem solving with applications.

topic	data structures and algorithms
concepts	algorithms, time-complexity, array and structure
data types	linked list, array, stack, queue, trees, union-find, bag, priority queues
sorting	selection sort, quick sort, merge sort, heap sort
searching	binary search tree, hashing

Course overview

What is this course?

- Intermediate-level course.
- Programming **after** programming for problem solving with applications.

topic	data structures and algorithms
concepts	algorithms, time-complexity, array and structure
data types	linked list, array, stack, queue, trees, union-find, bag, priority queues
sorting	selection sort, quick sort, merge sort, heap sort
searching	binary search tree, hashing
graph	BFS, DFS

Why study data structures?

Their impact is broad and far-reaching

Why study data structures?

Their impact is broad and far-reaching

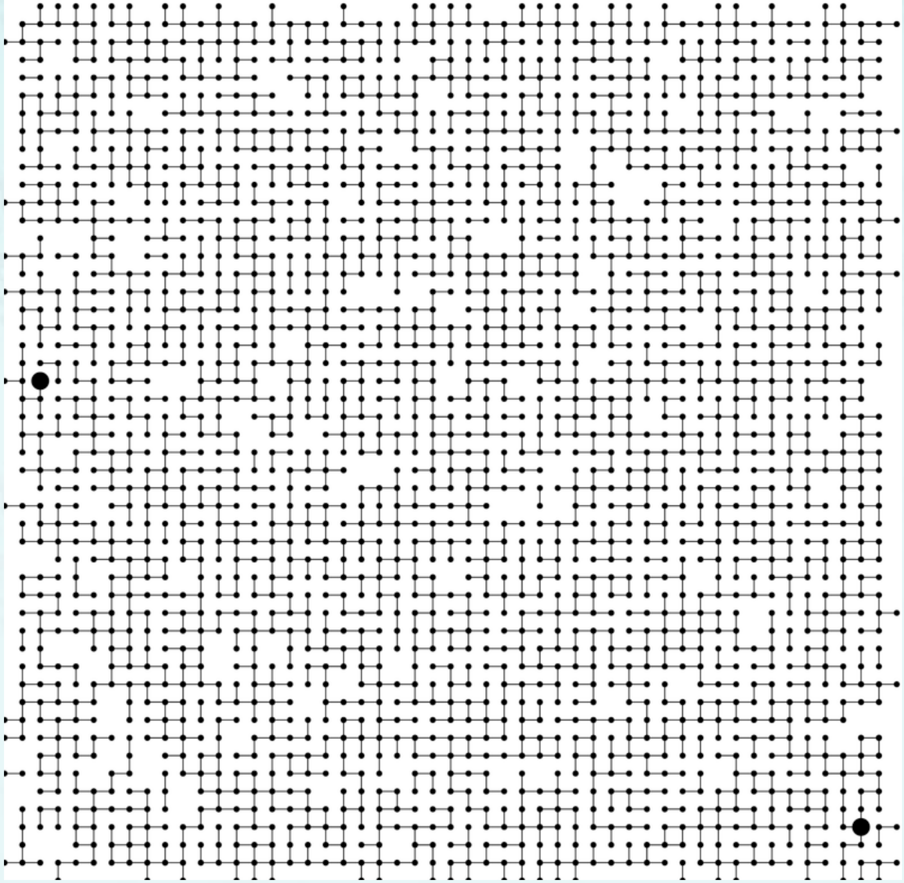
- **Internet** Web search, packet routing, distributed file sharing, …
- **Social networks** News feeds, advertisements, …
- **Computers** Circuit layout, file system, compilers, …
- **Computer graphics** Movies, video games, virtual reality, …
- **Multimedia** MP3, JPG, DivX, HDTV, face recognition, …

- **Security** Cell phones, e-commerce, voting machines, …
- **Biology** Human genome project, protein folding, …
- **Physics** N-body simulation, particle collision simulation, …

Why study data structures?

To solve problems that could not otherwise be addressed

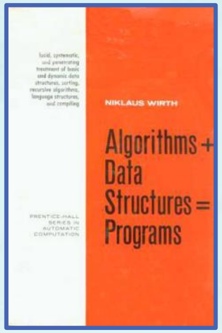
- To work with **algorithms** to solve problems
- Ex. Network connectivity, navigation



Why study data structures?

To become a proficient programmer.

*" Algorithms + **Data Structures** = Programs. " — Niklaus Wirth*



lucid, systematic,
and penetrating
treatment of basic
and dynamic data
structures, sorting,
recursive algorithms,
language structures,
and compiling

NIKLAUS WIRTH

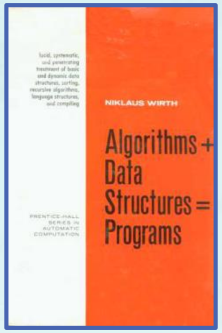
Algorithms +
Data
Structures =
Programs

PRENTICE-HALL
SERIES IN
AUTOMATIC
COMPUTATION

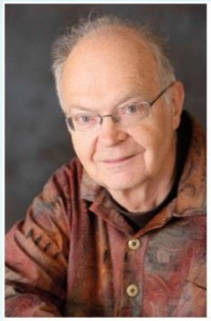
Why study data structures?

To become a proficient programmer.

*" Algorithms + **Data Structures** = Programs. " — Niklaus Wirth*


lucid, systematic, and penetrating treatment of basic and dynamic data structures, sorting, recursive algorithms, language structures, and compiling
NIKLAUS WIRTH
Algorithms + Data Structures = Programs
PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION

*" An **algorithm** must be seen to be believed. " — Donald Knuth*

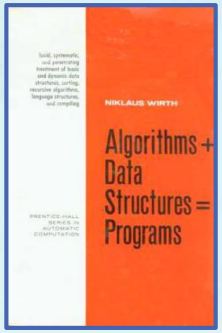


Donald E. Knuth, winner of the Katayanagi Prize for Research Excellence.

Why study data structures?

To become a proficient programmer.

*" Algorithms + **Data Structures** = Programs. "* — Niklaus Wirth



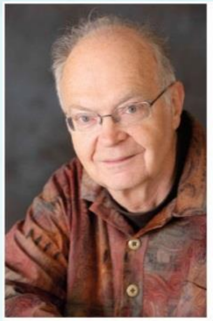
lucid, systematic,
and penetrating
treatment of basic
and dynamic data
structures, sorting,
recursive algorithms,
language structures,
and compiling

PRENTICE-HALL
SERIES IN
AUTOMATIC
COMPUTATION

NIKLAUS WIRTH

Algorithms +
Data
Structures =
Programs

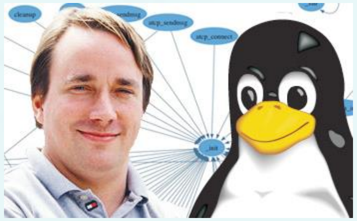
*" An **algorithm** must be seen to be believed. "* — Donald Knuth



Donald E. Knuth, winner of the Katayanagi Prize for Research Excellence.

*" I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about **data structures** and their **relationships**. "*

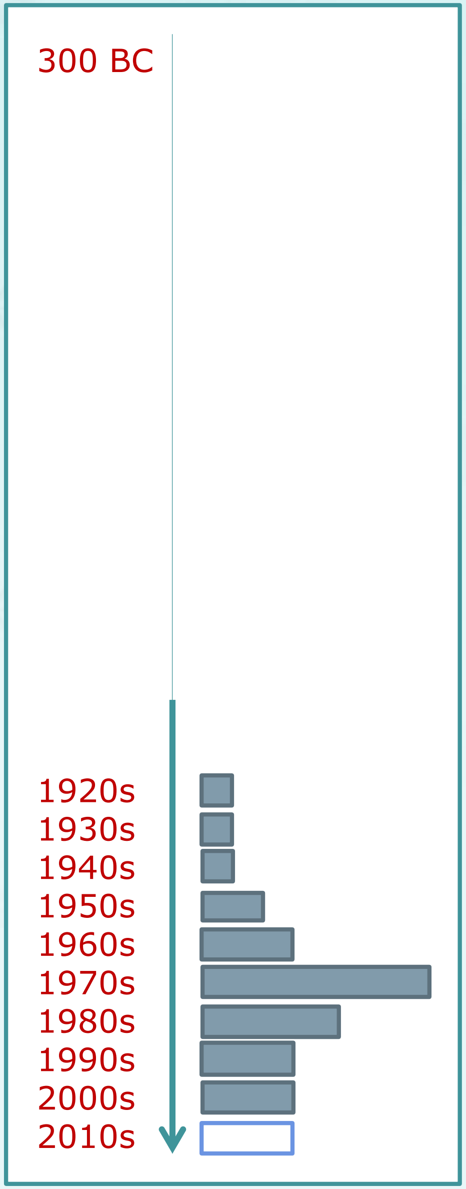
— *Linus Torvalds (creator of Linux)*



Why study data structures?

Algorithms – Old roots, new opportunities.

- Study of **algorithms** dates at least to Euclid.
- Formalized by Church and Turing in 1930s.
- Some important **algorithms** were discovered by undergraduates in a course like this.
- Then, why **data structures**?
  It always comes with algorithms like its shadow.



Ex. Fast Fourier Transform(FFT) Algorithm

Joseph Fourier(1768-1830) used for heat-transfer computation.
1805 – invented by Carl Friedrich Gauss.
1965 – popularized by James Cooley(IBM) and John Tukey(Princeton).

1986 – JPEG(Joint Photographic Experts Group) was formed.
1992 – issued the first standard of JPEG using DCT
Discrete cosine transform – another form of FFT.

Why study data structures?

They may unlock the secrets of life and of the universe.

Why study data structures?

They may unlock the secrets of life and of the universe.

Computational models are replacing math models in scientific inquiry.

Ex. Fourier Transform →

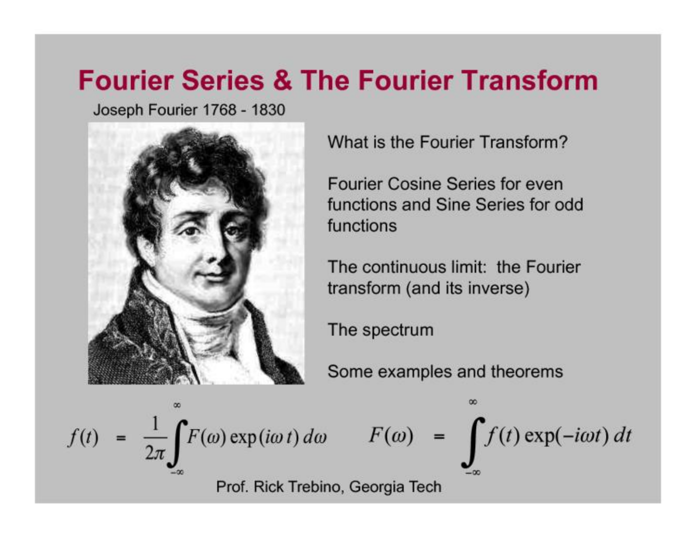
Why study data structures?

They may unlock the secrets of life and of the universe.

Computational models are replacing math models in scientific inquiry.

Ex. Fourier Transform → Fast FT algorithm

1805



Fourier Series & The Fourier Transform

Joseph Fourier 1768 - 1830

What is the Fourier Transform?

Fourier Cosine Series for even functions and Sine Series for odd functions

The continuous limit: the Fourier transform (and its inverse)

The spectrum

Some examples and theorems

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\exp(i\omega t)\,d\omega \qquad F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(-i\omega t)\,dt$$

Prof. Rick Trebino, Georgia Tech

~ old century science
(formula based)

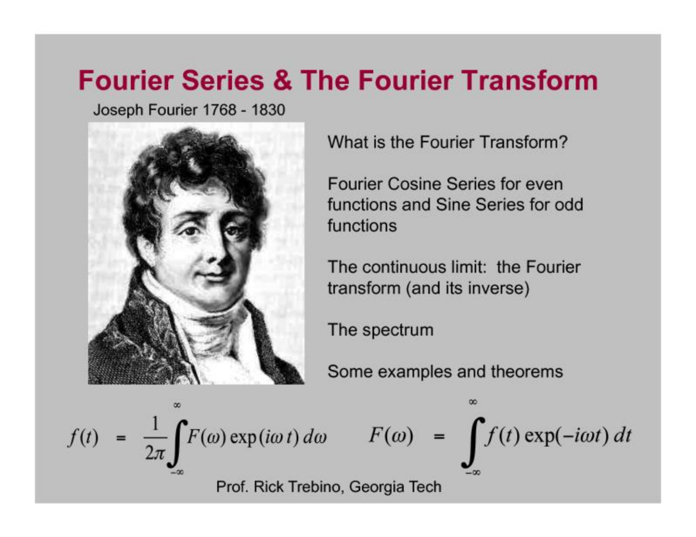
Why study data structures?

They may unlock the secrets of life and of the universe.

Computational models are replacing math models in scientific inquiry.

Ex. Fourier Transform → Fast FT algorithm → Image Processing →

1805



Fourier Series & The Fourier Transform

Joseph Fourier 1768 - 1830

What is the Fourier Transform?

Fourier Cosine Series for even functions and Sine Series for odd functions

The continuous limit: the Fourier transform (and its inverse)

The spectrum

Some examples and theorems

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\exp(i\omega t)\,d\omega \qquad F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(-i\omega t)\,dt$$

Prof. Rick Trebino, Georgia Tech

~ old century science
(formula based)

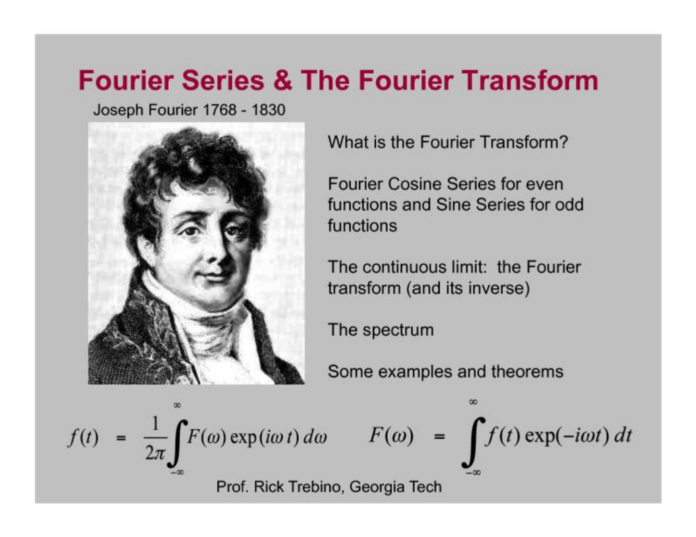
Why study data structures?

They may unlock the secrets of life and of the universe.

Computational models are replacing math models in scientific inquiry.

Ex. Fourier Transform → Fast FT algorithm → Image Processing →

1805 1965



Fourier Series & The Fourier Transform

Joseph Fourier 1768 - 1830

What is the Fourier Transform?

Fourier Cosine Series for even functions and Sine Series for odd functions

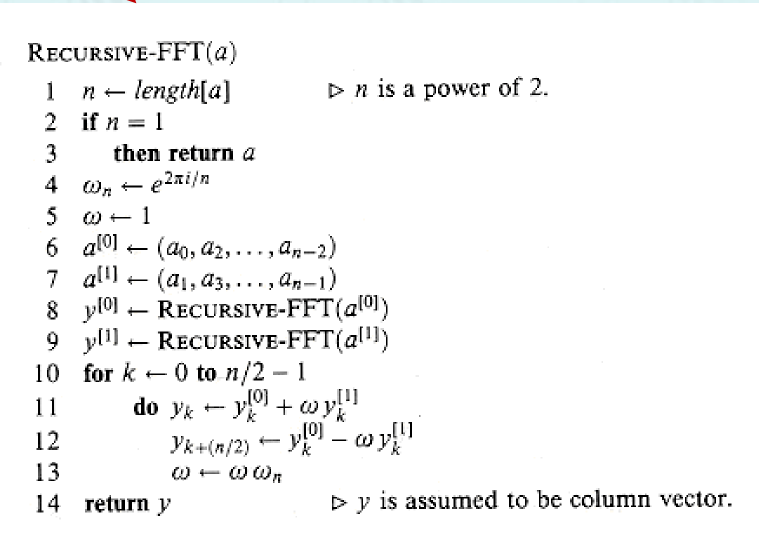
The continuous limit: the Fourier transform (and its inverse)

The spectrum

Some examples and theorems

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\exp(i\omega t)\,d\omega \qquad F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(-i\omega t)\,dt$$

Prof. Rick Trebino, Georgia Tech



```
RECURSIVE-FFT($a$)
$n \leftarrow length[a]$                ▷ $n$ is a power of 2.
if $n = 1$
    then return $a$
$\omega_n \leftarrow e^{2\pi i/n}$
$\omega \leftarrow 1$
$a^{[0]} \leftarrow (a_0, a_2, \ldots, a_{n-2})$
$a^{[1]} \leftarrow (a_1, a_3, \ldots, a_{n-1})$
$y^{[0]} \leftarrow$ RECURSIVE-FFT($a^{[0]}$)
$y^{[1]} \leftarrow$ RECURSIVE-FFT($a^{[1]}$)
for $k \leftarrow 0$ to $n/2 - 1$
    do $y_k \leftarrow y_k^{[0]} + \omega\, y_k^{[1]}$
       $y_{k+(n/2)} \leftarrow y_k^{[0]} - \omega\, y_k^{[1]}$
       $\omega \leftarrow \omega\, \omega_n$
return $y$                              ▷ $y$ is assumed to be column vector.
```

~ old century science
(formula based)

21th century science
(algorithm based)

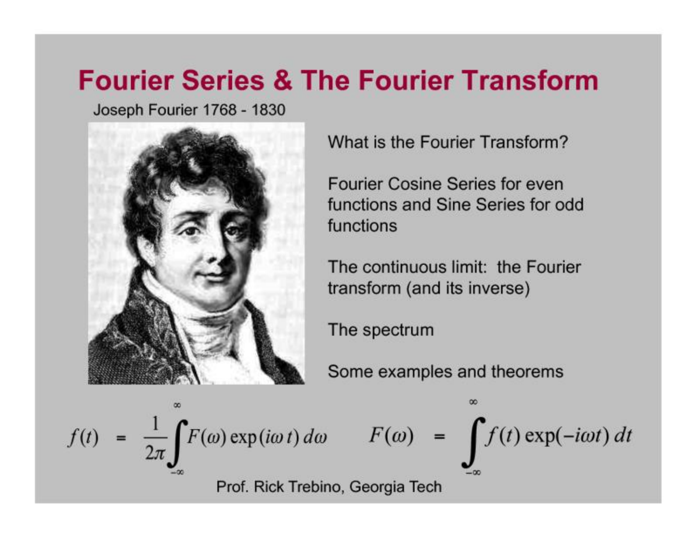
Why study data structures?

They may unlock the secrets of life and of the universe.

Computational models are replacing math models in scientific inquiry.

Ex. Fourier Transform → Fast FT algorithm → Image Processing → JPEG/MPEG

1805 1965 1992



Fourier Series & The Fourier Transform

Joseph Fourier 1768 - 1830

What is the Fourier Transform?

Fourier Cosine Series for even functions and Sine Series for odd functions

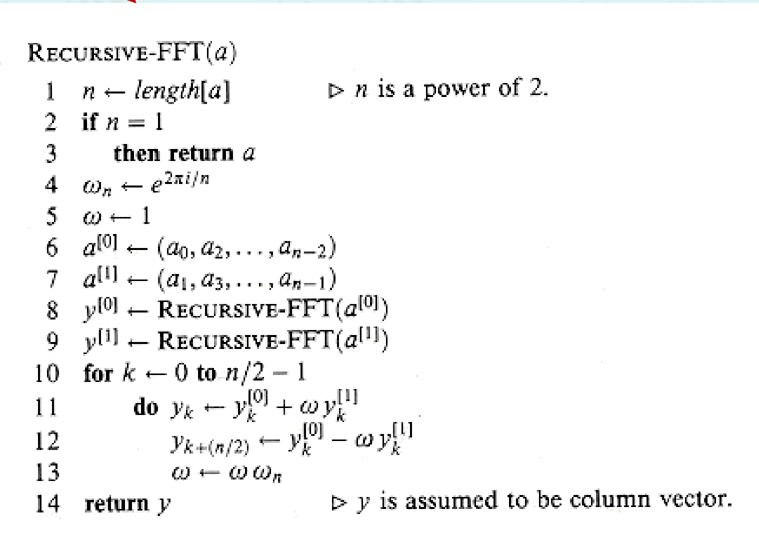
The continuous limit: the Fourier transform (and its inverse)

The spectrum

Some examples and theorems

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\exp(i\omega t)\,d\omega \qquad F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(-i\omega t)\,dt$$

Prof. Rick Trebino, Georgia Tech



```
RECURSIVE-FFT($a$)
  $n \leftarrow length[a]$                      ▷ $n$ is a power of 2.
  if $n = 1$
     then return $a$
  $\omega_n \leftarrow e^{2\pi i/n}$
  $\omega \leftarrow 1$
  $a^{[0]} \leftarrow (a_0, a_2, \ldots, a_{n-2})$
  $a^{[1]} \leftarrow (a_1, a_3, \ldots, a_{n-1})$
  $y^{[0]} \leftarrow$ RECURSIVE-FFT($a^{[0]}$)
  $y^{[1]} \leftarrow$ RECURSIVE-FFT($a^{[1]}$)
  for $k \leftarrow 0$ to $n/2 - 1$
      do $y_k \leftarrow y_k^{[0]} + \omega y_k^{[1]}$
         $y_{k+(n/2)} \leftarrow y_k^{[0]} - \omega y_k^{[1]}$
         $\omega \leftarrow \omega \omega_n$
  return $y$                                    ▷ $y$ is assumed to be column vector.
```

~ old century science
(formula based)

21th century science
(algorithm based)

Why study data structures?

For fun and profit.

Why study data structures?

- Their impact is broad and far-reaching.
- Old roots, new opportunities.
- To solve problems that could not otherwise be addressed.
- For intellectual stimulation.
- To become a proficient programmer.
- They may unlock the secrets of life and of the universe.
- For fun and profit..

Data Structures!
Why study anything else?

Algorithm!
Why study anything else?

Why study data structures?

Textbook & resources – required

- Fundamentals of data structures, 2nd Edition
  by Horwitz, Sahni, Anderson
- MOOC/OCW/EdX/Coursera or Google/YouTube/Wikipedia

Prerequisites

- C Programming: loops, arrays, functions, recursion, **pointer**
- C programming: using it extensively and **seriously.**
- Mathematics: high-school algebra.

Programming environment

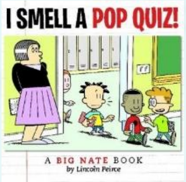
- MinGW/MSYS – GNU GCC C compiler,
- Atom or Sublime Text editor
- MS Visual Studio Community, **Eclipse CDT for C/C++**, Dev-C++,

Required reading and be ready for the Prbolem set 01 and 02

- Course syllabus, some hand-outs

ITP20001/ECE 20010 Data Structures

- Check your attendance – it matters!
- Be ready for a pop quiz



ITP20001/ECE 20010 Data Structures

Data Structures

Chapter 1

- *overview*
  - *pointers and dynamic memory allocation*
- *algorithm specification*
  - *homework set 01, 02*
  - *recursive algorithm*
- *data abstraction*
- ***performance analysis - time complexity***
  - ***discrete math***
  - ***homework set 03 - profiling***

Chapter 1 – Basic concepts

1.1 Overview – system life cycle

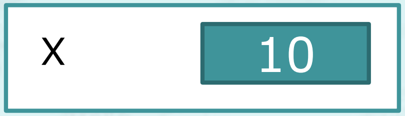
- **Requirements**
  - ✓ begins a set of specification.
- **Analysis**
  - ✓ break the problems down into manageable pieces.
- **Design**
  - ✓ lead to the creation of abstract data type and algorithm specifications – programming language independent.
- **Refinement and coding**
  - ✓ choose representations for data objects and write algorithms for each operation on them.
- **Verification**
  - ✓ correctness proof
  - ✓ testing
  - ✓ error removal

Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

Pointers and addresses

- Every **variable** represents an **address** in memory and a **value**:
  Ex. int x=10;



- We know the **variable** x and its the **value** is 10. How about **address** of x?

Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

Pointers and addresses

- Every **variable** represents an **address** in memory and a **value**:
  Ex. int x=10;

X

10

- We know the **variable** x and its the **value** is 10. How about **address** of x?

- You can get the address of x by using **the address operator &** such as &x; then what? You can save this address to a special data type called **pointer**.

Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

Pointers and addresses

- Every **variable** represents an **address** in memory and a **value**:
  Ex. int x=10;

- We know the **variable** x and its the **value** is 10. How about **address** of x?

- You can get the address of x by using **the address operator &** such as &x; then what? You can save this address to a special data type called **pointer**.
- A pointer is a variable that **only stores** an address of other variable.
  A pointer **only points** to the block of memory that a variable represents.
  & the address operator

Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

Pointers and addresses

- Every **variable** represents an **address** in memory and a **value**:
  Ex. int x=10;

X
10

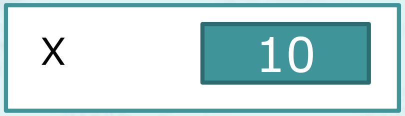
- We know the **variable** x and its the **value** is 10. How about **address** of x?

- You can get the address of x by using **the address operator &** such as &x; then what? You can save this address to a special data type called **pointer**.
- A pointer is a variable that **only stores** an address of other variable.
  A pointer **only points** to the block of memory that a variable represents.
  & the address operator
  * the dereferencing operator

1.2 Pointers and dynamic memory allocation

Pointers and addresses

- Every **variable** represents an **address** in memory and a **value**:
  Ex. int x=10;

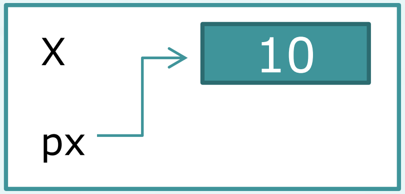


- We know the **variable** x and its the **value** is 10. How about **address** of x?

- You can get the address of x by using **the address operator &** such as &x; then what? You can save this address to a special data type called **pointer**.
- A pointer is a variable that **only stores** an address of other variable.
  A pointer **only points** to the block of memory that a variable represents.
  & the address operator
  * the dereferencing operator

- **Example:**

```
int  x = 10;
int  *px;
px = &x;
```



1.2 Pointers and dynamic memory allocation

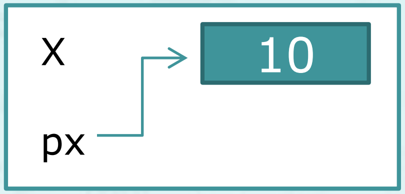
Pointer – cannot have a value but an address of memory

- For any type T in C, there is a corresponding type pointer-to-T
- The actual value of a pointer type is an address of memory

1.2 Pointers and dynamic memory allocation

Pointer – cannot have a value but an address of memory

- For any type T in C, there is a corresponding type pointer-to-T
- The actual value of a pointer type is an address of memory



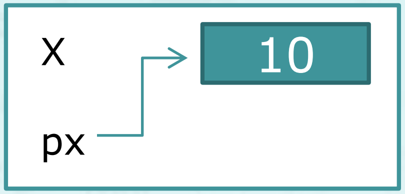
- Ex.

```
int x=10;        // type int
int *px;         // pointer to int type
px = &x;         // The value of px an is address of x
                 // px is pointing the value of x
```

1.2 Pointers and dynamic memory allocation

Pointer – cannot have a value but an address of memory

- For any type T in C, there is a corresponding type pointer-to-T
- The actual value of a pointer type is an address of memory

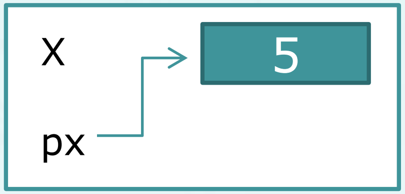


- Ex.

```
int x=10;         // type int
int *px;          // pointer to int type
px = &x;          // The value of px an is address of x
                  // px is pointing the value of x
```

- To store a value in x, you may do

```
*px = 5;
printf("x = %d, *px = %d, px = %x", x, *px, px);
```

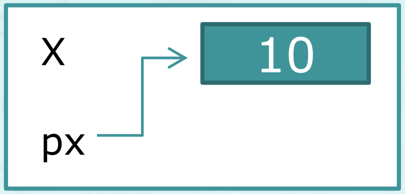


Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

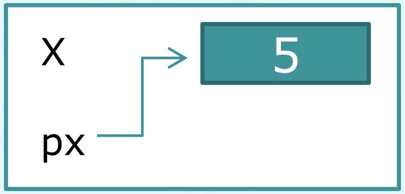
Pointer – cannot have a value but an address of memory

- For any type T in C, there is a corresponding type pointer-to-T
- The actual value of a pointer type is an address of memory



- Ex.
```
int x=10;          // type int
int *px;           // pointer to int type
px = &x;           // The value of px an is address of x
                   // px is pointing the value of x
```

- To store a value in x, you may do
```
*px = 5;
printf("x = %d, *px = %d, px = %x", x, *px, px);
```

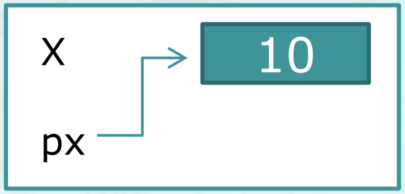


- When a pointer has NULL value, it is called a NULL pointer.
```
if (px == NULL)
```

1.2 Pointers and dynamic memory allocation

Pointer – cannot have a value but an address of memory

- For any type T in C, there is a corresponding type pointer-to-T
- The actual value of a pointer type is an address of memory

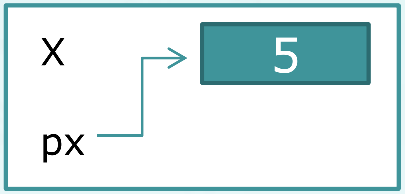


- Ex.

```
int x=10;         // type int
int *px;          // pointer to int type
px = &x;          // The value of px an is address of x
                  // px is pointing the value of x
```

- To store a value in x, you may do

```
*px = 5;
printf("x = %d, *px = %d, px = %x", x, *px, px);
```



- When a pointer has NULL value, it is called a NULL pointer.

```
if (px == NULL)
```

- Pointers are integer variables themselves, so can have **pointer to pointers**:

```
int    **ptr;
```

Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

- **Pointer Arithmetic –** you can do math on pointers:
- **Example:**

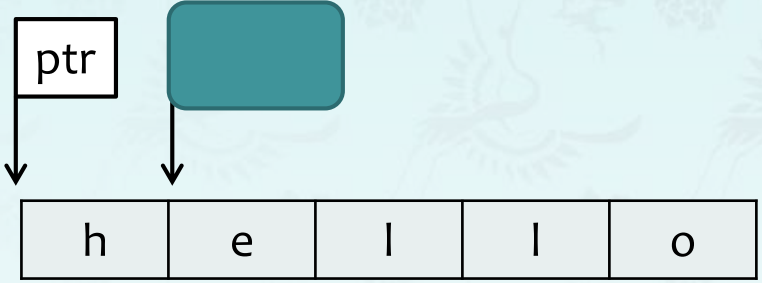
```
char str[10] = "hello";
char *ptr;
        ptr = str;                  // char *ptr = str;
```

Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

- **Pointer Arithmetic –** you can do math on pointers:
- **Example:**

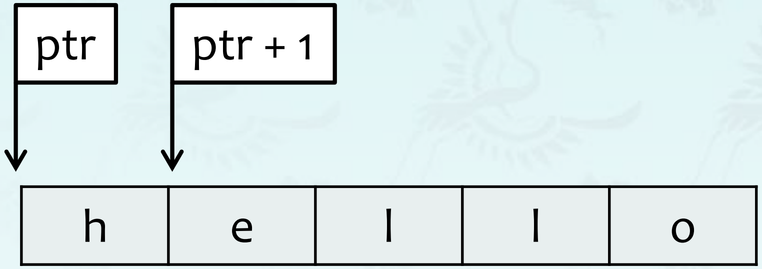
```
char str[10] = "hello";
char *ptr;
        ptr = str;                // char *ptr = str;
```



1.2 Pointers and dynamic memory allocation

- **Pointer Arithmetic –** you can do math on pointers:
- **Example:**

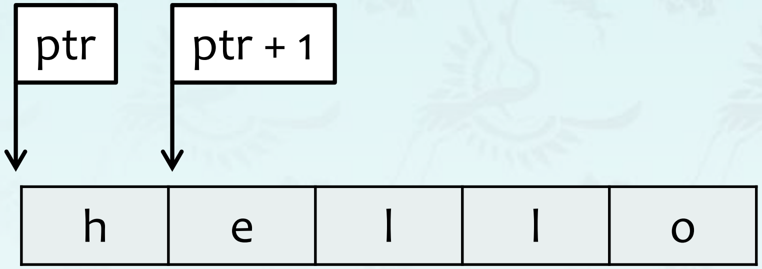
```
char str[10] = "hello";
char *ptr;
     ptr = str;              // char *ptr = str;
```



1.2 Pointers and dynamic memory allocation

- **Pointer Arithmetic –** you can do math on pointers:
- **Example:**

```
char str[10] = "hello";
char *ptr;
     ptr = str;              // char *ptr = str;
```



ptr

ptr + 1

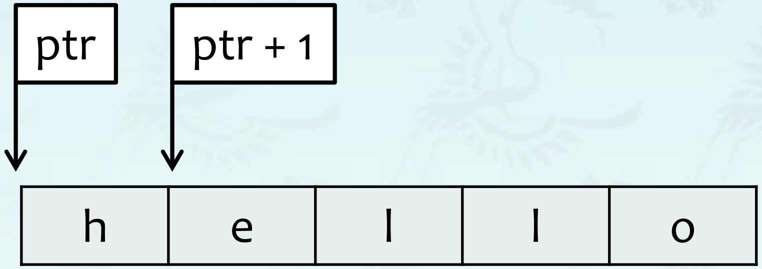
h	e	l	l	o

- **`ptr + i`** has the address: `ptr + i * sizeof(data type of ptr)`

1.2 Pointers and dynamic memory allocation

- **Pointer Arithmetic –** you can do math on pointers:
- **Example:**

```
char str[10] = "hello";
char *ptr;
     ptr = str;            // char *ptr = str;
```



ptr

ptr + 1

h	e	l	l	o

- **`ptr + i`** has the address: `ptr + i * sizeof(data type of ptr)`
- increment/decrement:
  - `ptr++,   ptr--, *ptr++ // equivalent to *(ptr++)`

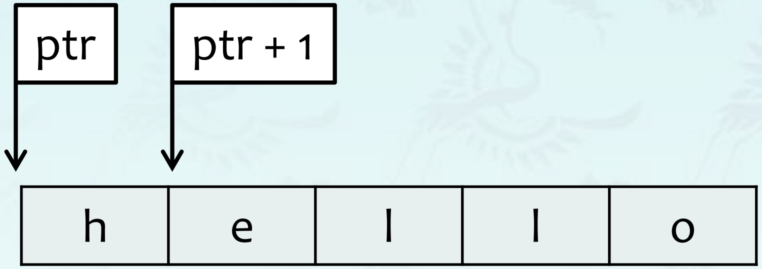
```
                        // different from (*ptr)++
```

Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

- **Pointer Arithmetic –** you can do math on pointers:
- **Example:**

```
char str[10] = "hello";
char *ptr;
        ptr = str;              // char *ptr = str;
```



ptr

ptr + 1

h	e	l	l	o

- **`ptr + i`** has the address: `ptr + i * sizeof(data type of ptr)`
- increment/decrement:
  - `ptr++,   ptr--, *ptr++ // equivalent to *(ptr++)`
    `// different from (*ptr)++`
- addition/subtraction:
  - `ptr + 2,  ptr - 2`

1.2 Pointers and dynamic memory allocation

- **Dynamic allocation:** In C, the memory may be allocated during run time or dynamically, called **heap**.
  - `#include <stdlib.h>`
  - `sizeof()` returns number of bytes of a data type.
  - `malloc()/realloc()` finds a specified amount of free memory and returns a void pointer to it.

1.2 Pointers and dynamic memory allocation

- **Dynamic allocation:** In C, the memory may be allocated during run time or dynamically, called **heap**.
  - `#include <stdlib.h>`
  - `sizeof()` returns number of bytes of a data type.
  - `malloc()/realloc()` finds a specified amount of free memory and returns a void pointer to it.

- Example:

  -
    ```
    char  *str = (char *)malloc(3 * sizeof(char));
    strcpy(str, "hi");

    str = (char *)realloc(str, ■ * sizeof(char));
    strcpy(str, "hello");
    ```

Chapter 1 – Basic concepts

1.2 Pointers and dynamic memory allocation

- **Dynamic allocation:** In C, the memory may be allocated during run time or dynamically, called **heap**.
  - `#include <stdlib.h>`
  - `sizeof()` returns number of bytes of a data type.
  - `malloc()/realloc()` finds a specified amount of free memory and returns a void pointer to it.

- Example:

  -
    ```
    char  *str = (char *)malloc(3 * sizeof(char));
    strcpy(str, "hi");

    str = (char *)realloc(str, 6 * sizeof(char));
    strcpy(str, "hello");
    ```

1.2 Pointers and dynamic memory allocation

- **Dynamic deallocation**
  - #include <stdlib.h>
  - **free( )** declares the memory pointed to by a pointer variable as free for future use:
  - Example:

```
char  *str = (char *)malloc(3 * sizeof(char));
assert(str != NULL);  // simplest checking ever
strcpy(str, "hi");
…
free(str);
```

ECE 20010 Data Structures

Data Structures

Chapter 1

- *overview*
    - *pointers and dynamic memory allocation*
- *algorithm specification*
    - *Problem set 02*
    - *recursive algorithm*
- *data abstraction*
- *performance analysis - time complexity*