Information about TA

Information about TA

- Asem Alaa

Information about TA

- Asem Alaa
- e-mail: asem.a.abdelaziz@gmail.com

Information about TA

- Asem Alaa
- e-mail: asem.a.abdelaziz@gmail.com
- Office hours and course materials are available on the course page:

{sbme-tutorials.github.io/2020/data-structures}

Information about TA

- Asem Alaa
- e-mail: asem.a.abdelaziz@gmail.com
- Office hours and course materials are available on the course page:

  {sbme-tutorials.github.io/2020/data-structures}

- Main research interests: Bioinformatics Algorithms and Machine Learning

Information about our course

Information about our course

- Aims to understanding various data structures by implementation from scratch.

Information about our course

- Aims to understanding various data structures by implementation from scratch.
- Understanding algorithms by implementation from scratch.

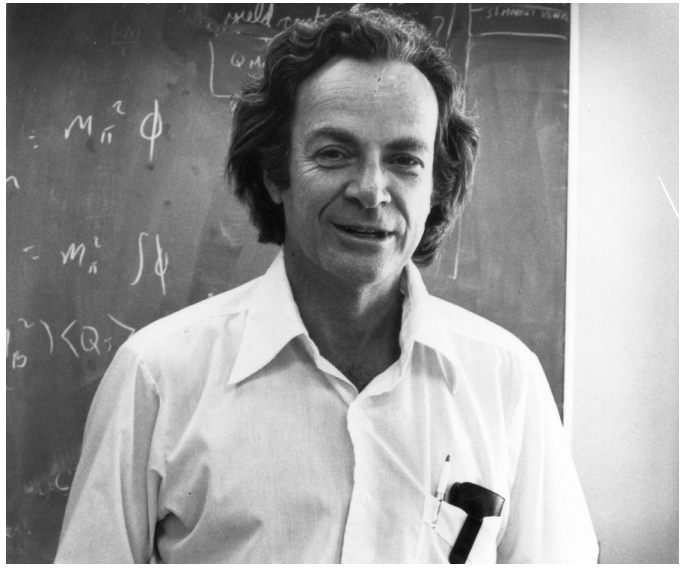
Information about our course

- Aims to understanding various data structures by implementation from scratch.
- Understanding algorithms by implementation from scratch.
- Modern C++ is used to build our data structures and algorithms.
  —*"What I cannot create I don't understand. R.F"*—

Information about our course

- Aims to understanding various data structures by implementation from scratch.
- Understanding algorithms by implementation from scratch.
- Modern C++ is used to build our data structures and algorithms.

—*"What I cannot create I don't understand. R.F"*—



Information about our course (cont'd)

Information about our course (cont'd)

- This course doesn't aim to teach OOP nor design patterns. (Though, I recommend learning these topics after this course).

Information about our course (cont'd)

- This course doesn't aim to teach OOP nor design patterns. (Though, I recommend learning these topics after this course).
- We still aim to write a very clean and simple C++ code.

Information about our course (cont'd)

- This course doesn't aim to teach OOP nor design patterns. (Though, I recommend learning these topics after this course).
- We still aim to write a very clean and simple C++ code.
- We will also learn and practice on version control systems like git.

Information about our course (cont'd)

- This course doesn't aim to teach OOP nor design patterns. (Though, I recommend learning these topics after this course).
- We still aim to write a very clean and simple C++ code.
- We will also learn and practice on version control systems like git.
- We will learn about different topics and tools in the development ecosystem.

Information about our course (cont'd)

- This course doesn't aim to teach OOP nor design patterns. (Though, I recommend learning these topics after this course).
- We still aim to write a very clean and simple C++ code.
- We will also learn and practice on version control systems like git.
- We will learn about different topics and tools in the development ecosystem.
- Implementation assignment each week.

Attendance

- Attendance is a requirement to pass the courses.

Attendance

- Attendance is a requirement to pass the courses.
- Not showing in more than 25% of lectures or tutorials is penalized by failing in the course.

Cheating and Academic Dishonesty

Cheating and Academic Dishonesty

Be it in exams or assignments

- Violating other rights and affects honest students as well.

Cheating and Academic Dishonesty

Be it in exams or assignments

- Violating other rights and affects honest students as well.
- Usually correlated with other corrupted personal values.

Cheating and Academic Dishonesty

Be it in exams or assignments

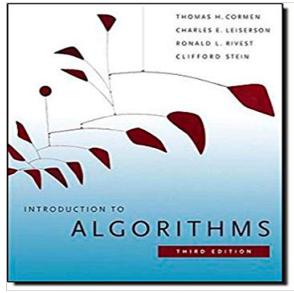
- Violating other rights and affects honest students as well.
- Usually correlated with other corrupted personal values.
- Forbidden by the religions' laws.

Recommended Resources

Data structure and Algorithms

Recommended Resources

Data structure and Algorithms

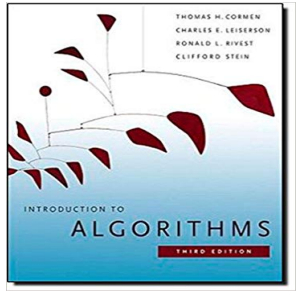


Introduction to Algorithms

by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.

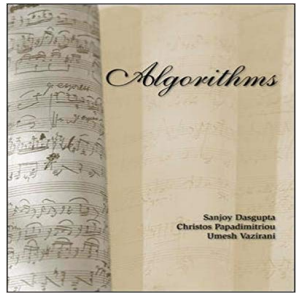
Recommended Resources

Data structure and Algorithms



Introduction to Algorithms

by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.

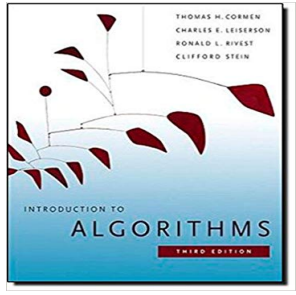


Algorithms

by Sanjoy Dasgupta, Christos H. Papadimitriou, Umesh Vazirani.

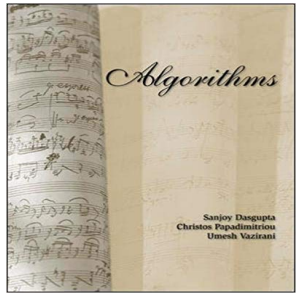
Recommended Resources

Data structure and Algorithms



Introduction to Algorithms

by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.



Algorithms

by Sanjoy Dasgupta, Christos H. Papadimitriou, Umesh Vazirani.

Online course: Data Structures

by Offered By University of California San Diego and National Research University Higher School of Economics.

Recommended Resources

C++ Programming

Recommended Resources

C++ Programming

Online course: C++ Fundamentals Including C++ 17

5h 48m long course, *by Kate Gregory.*

Recommended Resources

C++ Programming

Online course: C++ Fundamentals Including C++ 17

5h 48m long course, *by Kate Gregory.*

Documentation: C++ Standard Documentation

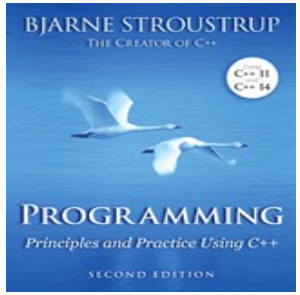
by C++ committee.

Recommended Resources

C++ Programming (cont'd)

Recommended Resources

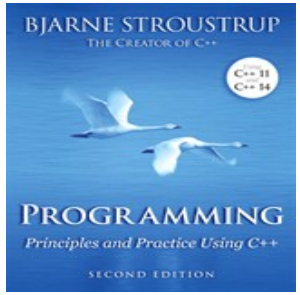
C++ Programming (cont'd)



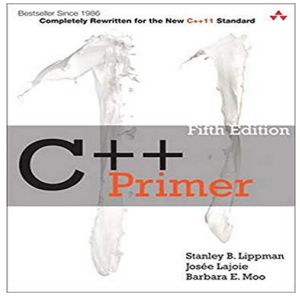
Textbook: Principles and Practice Using C++ including more than 100 pages of exercises, *by Bjarne Stroustrup*.

Recommended Resources

C++ Programming (cont'd)



Textbook: Principles and Practice Using C++ including more than 100 pages of exercises, *by Bjarne Stroustrup*.



Textbook: C++ Primer

by Stanley B. Lippman, Josée Lajoie, Barbara E. Moo.

The very C++ basics

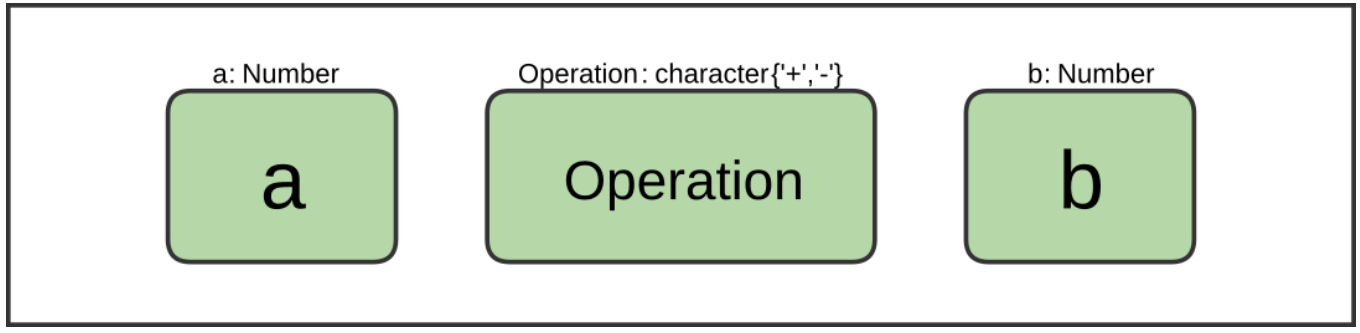
- Creating variables.
- Arithmetic Operations.
- If, else if, else.
- while-for-switch-case.
- functions
- namespace

Introduction

Introduction

A Simple Calculator Program and Memory Model

- Programs are all about playing with variables and groups of variables (structures)



C++

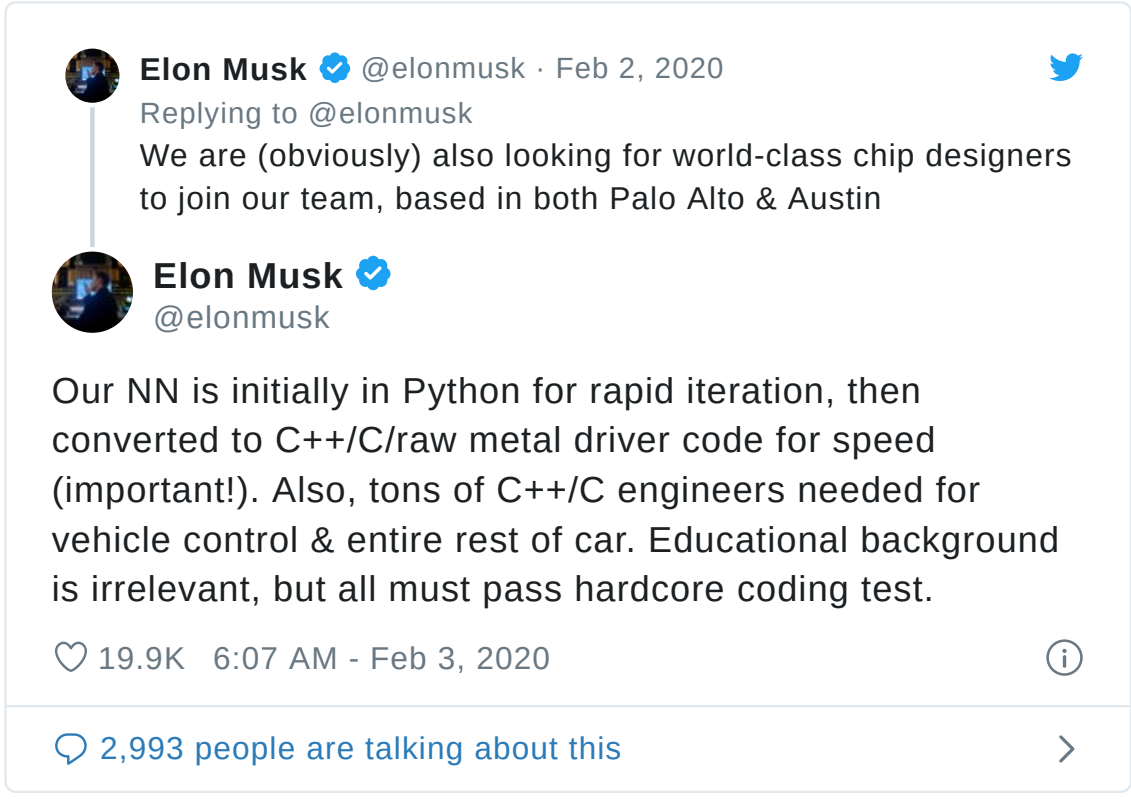
C++

- Bjarne Stroustrup created C++.
- C++ first appeared in 1985 (35 years ago).

What we can build using C++

- Self-driving cars
- Games
- PDE solvers
- Banking software
- Animation software
- Financial software
- Search engines
- Navigation software
- Social networking

What we can build using C++



Elon Musk @elonmusk · Feb 2, 2020

Replying to @elonmusk

We are (obviously) also looking for world-class chip designers to join our team, based in both Palo Alto & Austin

Elon Musk
@elonmusk

Our NN is initially in Python for rapid iteration, then converted to C++/C/raw metal driver code for speed (important!). Also, tons of C++/C engineers needed for vehicle control & entire rest of car. Educational background is irrelevant, but all must pass hardcore coding test.

19.9K 6:07 AM - Feb 3, 2020

2,993 people are talking about this

Variables in C++

Primitive Data Types (PDT) in C++

- `bool`: holds logical value (i.e `true` or `false`), occupies **1 byte** of memory.

Variables in C++

Primitive Data Types (PDT) in C++

- `bool`: holds logical value (i.e `true` or `false`), occupies **1 byte** of memory.
- `char`: a character (e.g `'a'`,`'b'`,..), occupies **1 byte** of memory.

Variables in C++

Primitive Data Types (PDT) in C++

- `bool`: holds logical value (i.e `true` or `false`), occupies **1 byte** of memory.
- `char`: a character (e.g `'a'`,`'b'`,..), occupies **1 byte** of memory.
- `int`: an integer (e.g ...,-1,0,1,2,..), occupies **4 bytes** of memory.

Variables in C++ (cont'd)

- `std::string`: a text (e.g `"Mostafa"`, `"ACCTTG"`, etc.), occupies variable size im memory.

Variables in C++ (cont'd)

- `std::string`: a text (e.g `"Mostafa"`, `"ACCTTG"`, etc.), occupies variable size im memory.
- `float`: a real-number-like (e.g 0.5, 3.141, 9.81), occupies **4 bytes** of memory.

Variables in C++ (cont'd)

- `std::string`: a text (e.g `"Mostafa"`, `"ACCTTG"`, etc.), occupies variable size im memory.
- `float`: a real-number-like (e.g 0.5, 3.141, 9.81), occupies **4 bytes** of memory.
- `double`: like float, but higher precision, occupies **8 bytes** of memory.

Variables in C++ (cont'd)

- `std::string`: a text (e.g `"Mostafa"`, `"ACCTTG"`, etc.), occupies variable size im memory.
- `float`: a real-number-like (e.g 0.5, 3.141, 9.81), occupies **4 bytes** of memory.
- `double`: like float, but higher precision, occupies **8 bytes** of memory.

Double vs float

- π equals:
  `3.14159265358979323846264338327950288419716939937510582097494459230781640628620899`

Variables in C++ (cont'd)

- `std::string`: a text (e.g `"Mostafa"`, `"ACCTTG"`, etc.), occupies variable size im memory.
- `float`: a real-number-like (e.g 0.5, 3.141, 9.81), occupies **4 bytes** of memory.
- `double`: like float, but higher precision, occupies **8 bytes** of memory.

Double vs float

- π equals: `3.14159265358979323846264338327950288419716939937510582097494459230781640628620899`
- π in `float` variable: `3.1415927`.

Variables in C++ (cont'd)

- `std::string`: a text (e.g `"Mostafa"`, `"ACCTTG"`, etc.), occupies variable size im memory.
- `float`: a real-number-like (e.g 0.5, 3.141, 9.81), occupies **4 bytes** of memory.
- `double`: like float, but higher precision, occupies **8 bytes** of memory.

Double vs float

- π equals: `3.14159265358979323846264338327950288419716939937510582097494459230781640628620899`
- π in `float` variable: `3.1415927`.
- π in `double` variable: `3.1415926535897931`.

Variables in C++

Variables in C++

- `std::vector`: collections.

Variables in C++

- `std::vector`: collections.
- `enum class`: finite sets.

Variables in C++

- `std::vector`: collections.
- `enum class`: finite sets.
- `pointer`: next week.

Variables in C++

- `std::vector`: collections.
- `enum class`: finite sets.
- `pointer`: next week.
- `reference`: next week.

Construction of Variables

Construction of Variables

A variable basically has:

Construction of Variables

A variable basically has:

1. **Data Type**: `int`, `char`, `bool`, ..., etc.

Construction of Variables

A variable basically has:

1. **Data Type**: `int`, `char`, `bool`, ..., etc.
2. **Name**: name of the variable to be used throughout your code.

Construction of Variables

A variable basically has:

1. **Data Type**: `int`, `char`, `bool`, ..., etc.
2. **Name**: name of the variable to be used throughout your code.
3. **Value**: the content of the variable.

Construction of Variables

A variable basically has:

1. **Data Type**: `int`, `char`, `bool`, ..., etc.
2. **Name**: name of the variable to be used throughout your code.
3. **Value**: the content of the variable.

Don't mix between them!

So, to construct a variable you need to:

So, to construct a variable you need to:

1. Declare a variable (Compiler Requirement).
    - Indicate your variable **type**.
    - Indicate your variable **name** that your are going to refer later.

So, to construct a variable you need to:

1. Declare a variable (Compiler Requirement).
    - Indicate your variable **type**.
    - Indicate your variable **name** that your are going to refer later.
2. Initialize that variable (to survive undefined behaviour).
    - Give it an initial **value**.

Example: constructing variables

Example: constructing variables

First of all:

- Comments in C++ code.

Example: constructing variables

First of all:

- Comments in C++ code.

```
// What comes after double forward-slash (//) in a line is a comment.
// Compiler Ignores comments.
// Comments are not contributing to your application logic.
// Comments are message to the readers of your code.
```

Cont'd

Cont'd

```
// Declare a character variable.
// Variable names are not the actual value!
char x;
// What is the value of x?!
// When not initialized, x will hold a value from garbage.

// Please, always initialize your variables.

// Declaration of character with initializng to 's'.
char x = 's';

// Declaration of float whith initializing to 3.1415.
float pi = 3.1415;
```

Cont'd

Cont'd

```
// If no interesting value to initialize
// your variable with, initialize with 0.
int k = 0;

// You can initialize a variable with the value of
// another variable.
int j = k;

// Another way to initialize a variable is
// using braces, it is up to you.
double e {2.71828};
```

Cont'd

```
// If no interesting value to initialize
// your variable with, initialize with 0.
int k = 0;

// You can initialize a variable with the value of
// another variable.
int j = k;

// Another way to initialize a variable is
// using braces, it is up to you.
double e {2.71828};
```

One way to avoid bugs (undefined behaviour) is initializing your variables.

Cont'd

```
// If no interesting value to initialize
// your variable with, initialize with 0.
int k = 0;

// You can initialize a variable with the value of
// another variable.
int j = k;

// Another way to initialize a variable is
// using braces, it is up to you.
double e {2.71828};
```

One way to avoid bugs (undefined behaviour) is initializing your variables.

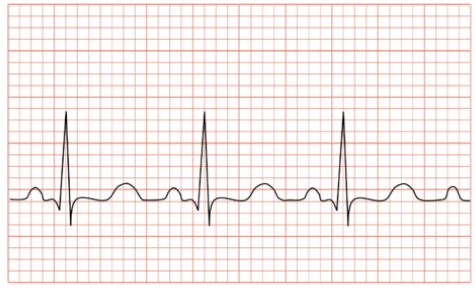
Any Questions?

Overview on data structures

Collections of Variables (Data Structures)

A data structure is a particular way of organizing data so they can be used efficiently by some task.

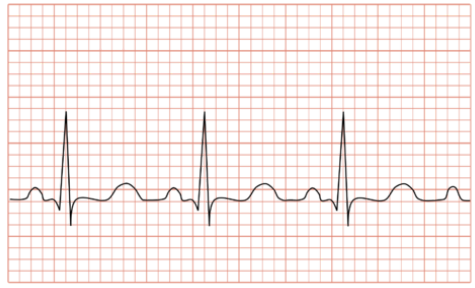
Example 1: Data Structures in Biomedical Informatics



Collections of Variables (Data Structures)

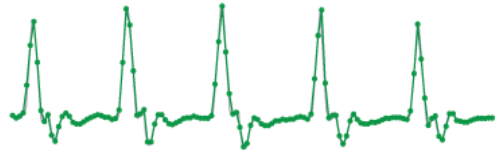
A data structure is a particular way of organizing data so they can be used efficiently by some task.

Example 1: Data Structures in Biomedical Informatics

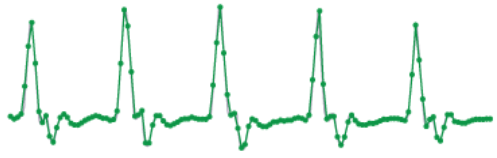


- Application: analysis of ECG of the heart.

Example 1: Data Structures in Biomedical Informatics (cont'd)

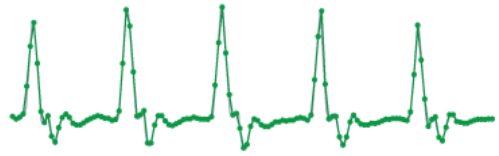


Example 1: Data Structures in Biomedical Informatics (cont'd)



$$\text{Sampled Signal} = [12.3, 12.7, 14.5, 18.0, 16.2, 10.1, 8.6, \ldots]$$

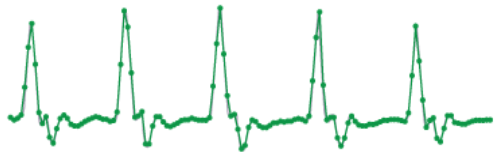
Example 1: Data Structures in Biomedical Informatics (cont'd)



$$\text{Sampled Signal} = [12.3, 12.7, 14.5, 18.0, 16.2, 10.1, 8.6, \ldots]$$

- It is pointless to construct a variable for each sample!

Example 1: Data Structures in Biomedical Informatics (cont'd)

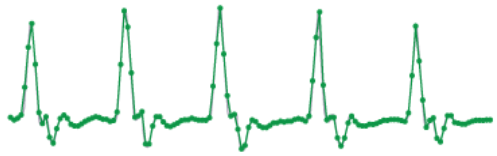


$$\text{Sampled Signal} = [12.3, 12.7, 14.5, 18.0, 16.2, 10.1, 8.6, \ldots]$$

- It is pointless to construct a variable for each sample!

```
double s1 = 12.3;
double s2 = 12.7;
.
.
double s256 = -0.5;
```

Example 1: Data Structures in Biomedical Informatics (cont'd)



$$\text{Sampled Signal} = [12.3, 12.7, 14.5, 18.0, 16.2, 10.1, 8.6, \ldots]$$

- It is pointless to construct a variable for each sample!

```
double s1 = 12.3;
double s2 = 12.7;
.
.
double s256 = -0.5;
```

Instead we need to store all values and use a single name for them.

Example 2: Data Structures in Biomedical Informatics

Input:

txt	**= "AACAAGAATAACAACA"**
pattern	= "AACA"

Example 2: Data Structures in Biomedical Informatics

Input:

txt	**= "AACAAGAATAACAACA"**
pattern	= "AACA"

- Output: "AACAAGAATAACAACA"
- Pattern found at index 0,9,12

Example 2: Data Structures in Biomedical Informatics

Input:

txt	**= "AACAAGAATAACAACA"**
pattern	= "AACA"

- Output: "AACAAGAATAACAACA"
- Pattern found at index 0,9,12

Assume that we have:

- `txt` of size $n = 10,000,000$
- `pattern` of size $m = 12$

Example 2: Data Structures in Biomedical Informatics

Input:

txt	**= "AACAAGAATAACAACA"**
pattern	= "AACA"

- Output: "AACAAGAATAACAACA"
- Pattern found at index 0,9,12

Assume that we have:

- `txt` of size $n = 10,000,000$
- `pattern` of size $m = 12$
- No. of comparisons/steps $\approx m \times n = 120,000,000$

Example 2: Data Structures in Biomedical Informatics

Input:

txt	**= "AACAAGAATAACAACA"**
pattern	= "AACA"

- Output: "AACAAGAATAACAACA"
- Pattern found at index 0,9,12

Assume that we have:

- `txt` of size $n = 10,000,000$
- `pattern` of size $m = 12$
- No. of comparisons/steps $\approx m \times n = 120,000,000$
- Can we do it in only $\approx$ 12 step!!

Example 2: Data Structures in Biomedical Informatics

Input:

txt	**= "AACAAGAATAACAACA"**
pattern	= "AACA"

- Output: "AACAAGAATAACAACA"
- Pattern found at index 0,9,12

Assume that we have:

- `txt` of size $n = 10,000,000$
- `pattern` of size $m = 12$
- No. of comparisons/steps $\approx m \times n = 120,000,000$
- Can we do it in only $\approx$ 12 step!!
- Yes! but using special data structure like suffix trees.

Data structures and basic algorithms on them

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

- Different data structures (i.e collections of elements): Array, Linked List, Stack, Queue, Tree.

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

- Different data structures (i.e collections of elements): Array, Linked List, Stack, Queue, Tree.
- How to **construct** collections.

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

- Different data structures (i.e collections of elements): Array, Linked List, Stack, Queue, Tree.
- How to **construct** collections.
- How to **insert** elements to our collection.

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

- Different data structures (i.e collections of elements): Array, Linked List, Stack, Queue, Tree.
- How to **construct** collections.
- How to **insert** elements to our collection.
- How to **modify** element in our collection.

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

- Different data structures (i.e collections of elements): Array, Linked List, Stack, Queue, Tree.
- How to **construct** collections.
- How to **insert** elements to our collection.
- How to **modify** element in our collection.
- How to **delete** an element.

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

- Different data structures (i.e collections of elements): Array, Linked List, Stack, Queue, Tree.
- How to **construct** collections.
- How to **insert** elements to our collection.
- How to **modify** element in our collection.
- How to **delete** an element.
- How to **traverse** our collection (i.e print all its elements).

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

- Different data structures (i.e collections of elements): Array, Linked List, Stack, Queue, Tree.
- How to **construct** collections.
- How to **insert** elements to our collection.
- How to **modify** element in our collection.
- How to **delete** an element.
- How to **traverse** our collection (i.e print all its elements).
- Applying **algorithms** on our collection.

Data structures and basic algorithms on them

Mainly these what we are going to study through this course:

- Different data structures (i.e collections of elements): Array, Linked List, Stack, Queue, Tree.
- How to **construct** collections.
- How to **insert** elements to our collection.
- How to **modify** element in our collection.
- How to **delete** an element.
- How to **traverse** our collection (i.e print all its elements).
- Applying **algorithms** on our collection.
- **Searching** for an element in our collection.

Back to C++

Basic Operations on Primitive Data Types (PDT)

Basic Operations on Primitive Data Types (PDT)

- A) Arithmetic Operations.

Basic Operations on Primitive Data Types (PDT)

- A) Arithmetic Operations.

```
int x = 12;
int y = 5;

x + y; // 17
x - y; // 7
x * y; // 60
x / y; // 2
x % y; // 2
```

Cont'd

Cont'd

```
// x = x+y
x += y; // x is now 17.

// increment: x = x+1
++x; // x is now 18.

// x = x-y
x -= y; // x is now 13.

// decrement: x = x-1
--x; // x is now 12.

double u = 12.5;
double v { 3 };

u / v;  // 4.166667
```

Cont'd

Cont'd

- B) Logical Operations

Cont'd

- B) Logical Operations

```
int x = 3;
int y = 4;

// equal
x==y; // False

// not equal
x!=y; // True

// less than
x<y; // True

// greater than
x>y; // False
```

Cont'd

Cont'd

```
// less than or equal
x<=y; // True

// greater than or equal
x>=y; // False

// logical and
x == 3 && y > x; // True
x != 3 && y > x; // False
true && true; // True
5 < 10 && 13 >= 11; // True
5 % 2 == 1 && 3 / 2 > 1; // False
5 % 2 == 1 && 3 / 2.0 > 1; // True
```

Cont'd

```
// less than or equal
x<=y; // True

// greater than or equal
x>=y; // False

// logical and
x == 3 && y > x; // True
x != 3 && y > x; // False
true && true; // True
5 < 10 && 13 >= 11; // True
5 % 2 == 1 && 3 / 2 > 1; // False
5 % 2 == 1 && 3 / 2.0 > 1; // True
```

- Note 1: expressions are more generic unit than variables.

Cont'd

```
// less than or equal
x<=y; // True

// greater than or equal
x>=y; // False

// logical and
x == 3 && y > x; // True
x != 3 && y > x; // False
true && true; // True
5 < 10 && 13 >= 11; // True
5 % 2 == 1 && 3 / 2 > 1; // False
5 % 2 == 1 && 3 / 2.0 > 1; // True
```

- Note 1: expressions are more generic unit than variables.
- Note 2: (`expression % 2 == 1`) is a way to test if that expression is even or odd.

Cont'd

```
// logical or
true || true; // True
true || false; // True
false || true; // True
false || false; // False
5 % 2 == 1 || 3 / 2 > 1; // True
```

Basic Control Statements

Basic Control Statements

- Conditions: `if`, `else if`, `else`, `switch`-`case`

```
bool myCondition = 5 % 2 == 1 || 3 / 2 > 1;

if( myCondition )
{
  // Some operations here.
}
else
{
  // Other operations here.
}
```

Cont'd

```
char base = 'A'; char complementary = 'T';
std::cin >> base;
if( base == 'A' )
{
  complementary = 'T';
}
else if( base == 'C' )
{ complementary = 'G'; }
else if( base == 'G' )
  complementary = 'C';
else
  complementary = 'A';

std::cout << complementary << std::endl;
```

Cont'd

```
char base = 'A'; char complementary = 'T';
std::cin >> base;
switch (base)
{
  case 'A':
    complementary = 'T'; break;
  case 'C':
    complementary = 'G'; break;
  case 'G':
    complementary = 'C'; break;
  default:
    complementary = 'A'; break;
}
std::cout << complementary << std::endl;
```

Cont'd

```
char base = 'A'; char complementary = 'T';
std::cin >> base;
switch (base)
{
  case 'A':
    complementary = 'T'; break;
  case 'C':
    complementary = 'G'; break;
  case 'G':
    complementary = 'C'; break;
  default:
    complementary = 'A'; break;
}
std::cout << complementary << std::endl;
```

- `std::cout` is used to print out object values to the terminal.

Cont'd

```
char base = 'A'; char complementary = 'T';
std::cin >> base;
switch (base)
{
  case 'A':
    complementary = 'T'; break;
  case 'C':
    complementary = 'G'; break;
  case 'G':
    complementary = 'C'; break;
  default:
    complementary = 'A'; break;
}
std::cout << complementary << std::endl;
```

- `std::cout` is used to print out object values to the terminal.
- What is `std::` and what is `cout`?

Loops: `for`, `while`

```
for( int i = 0; i < 10; ++i )
{
  std::cout << i << " ";
}
// prints:0 1 2 3 4 5 6 7 8 9

int i = 0;
while( i < 10 )
{
  std::cout << i << " ";
}
// prints:0 1 2 3 4 5 6 7 8 9
```

Loops: `for`, `while`

```
for( int i = 0; i < 10; ++i )
{
  std::cout << i << " ";
}
// prints:0 1 2 3 4 5 6 7 8 9

int i = 0;
while( i < 10 )
{
  std::cout << i << " ";
}
// prints:0 1 2 3 4 5 6 7 8 9
```

Any bug?

Loops: `for`, `while`

```
for( int i = 0; i < 10; ++i )
{
  std::cout << i << " ";
}
// prints:0 1 2 3 4 5 6 7 8 9

int i = 0;
while( i < 10 )
{
  std::cout << i << " ";
  ++i;
}
// prints:0 1 2 3 4 5 6 7 8 9
```

Functions

Functions

A function is a unit that you write some logic in it. So we can use that logic many times through that function.

Functions

A function is a unit that you write some logic in it. So we can use that logic many times through that function. A function basically has:

Functions

A function is a unit that you write some logic in it. So we can use that logic many times through that function. A function basically has:

- **Name** to be used when calling this function.

Functions

A function is a unit that you write some logic in it. So we can use that logic many times through that function. A function basically has:

- **Name** to be used when calling this function.
- **Return Type**: a function may return `int`, `double`, `char`, ... etc. Also, it may not return, so its return type is `void`.

Functions

A function is a unit that you write some logic in it. So we can use that logic many times through that function. A function basically has:

- **Name** to be used when calling this function.
- **Return Type**: a function may return `int`, `double`, `char`, ... etc. Also, it may not return, so its return type is `void`.
- **Arguments**: the variables given to your function so it makes some operations on.

Declaration and Definition of Functions

Declaration and Definition of Functions

Like variables, functions must be declared before you implement your logic in this function.

Declaration and Definition of Functions

Like variables, functions must be declared before you implement your logic in this function.

- **Declaration** is a function header that indicates the function **name**, **return type**, and **arguments**.

Declaration and Definition of Functions

Like variables, functions must be declared before you implement your logic in this function.

- **Declaration** is a function header that indicates the function **name**, **return type**, and **arguments**.
- **Definition** is the function logic.

Example

```
double average( double a , double b ) // function header (Declaration)
{ // function definition (logic) goes here
  return ( a + b ) / 2;
}

double max( double a , double b ) // declaration
{ // definition
  if( a > b )
    return a;
  else return b;
}

int main()
{
  // Define 'x' as double. Realize the type consistency.
  double x = average( 13.5 , 21.0 );
  bool y = average( 11.5 , 15.0 ); // Compiler Error, type mismatch!
  std::cout << max( 15.0 , 9.0 ) << std::endl; // prints: 15.0
}
```

Scopes and Lifetime

Scopes and Lifetime

- Variables are bound to scopes where they are declared. Scopes types:

Scopes and Lifetime

- Variables are bound to scopes where they are declared. Scopes types:

1. Local scope: any variable declared in a function is not accessible outside that function.

Scopes and Lifetime

- Variables are bound to scopes where they are declared. Scopes types:

1. Local scope: any variable declared in a function is not accessible outside that function.
2. Block: any variable declared inside braces `{}`, like the blocks of the `for`, `while`, `if`, `else if`, `else`, and `switch`-`case`.

Scopes and Lifetime

- Variables are bound to scopes where they are declared. Scopes types:

1. Local scope: any variable declared in a function is not accessible outside that function.
2. Block: any variable declared inside braces `{}`, like the blocks of the `for`, `while`, `if`, `else if`, `else`, and `switch`-`case`.
3. `Namespace` scope.

Scopes and Lifetime

- Variables are bound to scopes where they are declared. Scopes types:

1. Local scope: any variable declared in a function is not accessible outside that function.
2. Block: any variable declared inside braces `{}`, like the blocks of the `for`, `while`, `if`, `else if`, `else`, and `switch`-`case`.
3. `Namespace` scope.

- Once the scope is terminated, all variables in that scope are destructed.

Scopes and Lifetime

- Variables are bound to scopes where they are declared. Scopes types:

1. Local scope: any variable declared in a function is not accessible outside that function.
2. Block: any variable declared inside braces `{}`, like the blocks of the `for`, `while`, `if`, `else if`, `else`, and `switch`-`case`.
3. `Namespace` scope.

- Once the scope is terminated, all variables in that scope are destructed.
- Otherwise, if variable is declared outside all of the mentioned scopes, then it is a global variable.

Scopes and Lifetime

- Variables are bound to scopes where they are declared. Scopes types:

1. Local scope: any variable declared in a function is not accessible outside that function.
2. Block: any variable declared inside braces `{}`, like the blocks of the `for`, `while`, `if`, `else if`, `else`, and `switch`-`case`.
3. `Namespace` scope.

- Once the scope is terminated, all variables in that scope are destructed.
- Otherwise, if variable is declared outside all of the mentioned scopes, then it is a global variable.
- Global variables are accessible anywhere in the source file.

Example of a local scope and a block scope

```
double rectangleArea( double width , double height )
{
  // The arguments width and height are local variables to this function.
  // width, height, area are not accessible outside.

  double area = width * height;
  return area; // return by value
}

int main()
{
  // area here is completely different than area in the rectangleArea fun
  // They have the same value. But they are not same the variables.
  double area = rectangleArea( 12.9 , 2.5 );
}
```

Example of namespace scope

Example of namespace scope

Consider a situation when you need to implement a function that computes the area of rectangle and the area of right triangle. Using the same function name `area`!

Example of namespace scope

```
namespace rectangle
{
  double area( double width , double height )
  {
    return width * height;
  }
}
namespace triangle
{
  double area( double base , double height )
  {
    return ( base * height ) / 2;
  }
}
int main()
{
  double rectangleArea = rectangle::area( 12.9 , 2.5 );
  double triangleArea = triangle::area( 4.0 , 3.0 );
  std::cout << rectangleArea << std::endl << triangleArea << std::endl;
}
```

Example of namespace scope

```
int main()
{
   double rectangleArea = rectangle::area( 12.9 , 2.5 );
   double triangleArea = triangle::area( 4.0 , 3.0 );
  std::cout << rectangleArea << std::endl << triangleArea << std::endl;
}
```

Example of namespace scope

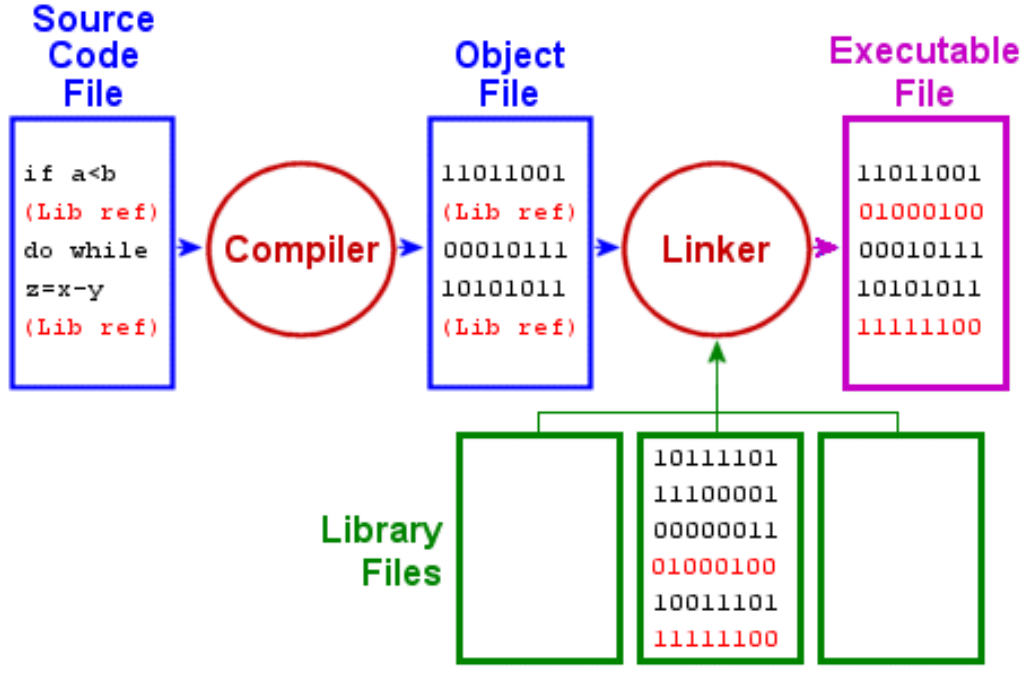
```
int main()
{
   double rectangleArea = rectangle::area( 12.9 , 2.5 );
   double triangleArea = triangle::area( 4.0 , 3.0 );
  std::cout << rectangleArea << std::endl << triangleArea << std::endl;
}
```

- Now you have a little sense about `std::cout` and `std` Namespace.

C++ Programs

C++ is a compiled language which means you need to install a compiler in order to generate executable files for your application.

A typical process of executable file generation is shown in this image:



Writing C++ codes

- To write a C++ source code we will rely on Integrated Development Environment (IDE).

Writing C++ codes

- To write a C++ source code we will rely on Integrated Development Environment (IDE).

Qt Creator for SBE201

{Installing and running Qt Creator IDE}

You will find in the link above instructions on:

1. Downloading the Qt project packages.
2. Installation.
3. Starting and writing your first program.

Lightweight Alternative: Microsoft VSCode

- A light IDE.
- You can use to write Markdown files and simple C++ codes.
- Download from: {Visual Studio Code}

Lightweight Alternative: Microsoft VSCode

- A light IDE.
- You can use to write Markdown files and simple C++ codes.
- Download from: {Visual Studio Code}
- After downloading the `.deb` package file, open a terminal at the directory where you downloaded the package file, then:

Lightweight Alternative: Microsoft VSCode

- A light IDE.
- You can use to write Markdown files and simple C++ codes.
- Download from: {Visual Studio Code}
- After downloading the `.deb` package file, open a terminal at the directory where you downloaded the package file, then:

```
sudo dpkg -i ./<package file="">
code
```

Writing your first C++ application

Let's write our first source file. Copy the following code to your VS Code editor. Save the file as `firstApp4SBME.cpp`.

```
#include <iostream>
namespace rectangle
{
  double area( double width , double height )
  {
    return width * height;
  }
}
namespace triangle
{
  double area( double base , double height )
  {
    return ( base * height ) / 2;
  }
}
int main()
{
  double rectangleArea = rectangle::area( 12.9 , 2.5 );
```

Compiling your code

Compiling your code

```
g++ -o firstAppSBME firstAppSBME.cpp
```

Compiling your code

```
g++ -o firstAppSBME firstAppSBME.cpp
```

CONGRATULATIONS! you have built your first application.

Execute the application

Execute the application

```
./firstAppSBME
```

Execute the application

```
./firstAppSBME
```

you should see:

```
32.25
6
```

Git

Git

Problem Definition

Imagine the case when *Emad* and *Ahmed* need to collaborate on this project. Such that:

Problem Definition (cont'd)

1. *Emad* generates the biolerplate/skeleton (i.e the files and the main function) of the project.

```
#include <iostream>
namespace rectangle
{
    // No implementation yet!
}
namespace triangle
{
    // No implmenetation yet!
}
int main()
{
  double rectangleArea = rectangle::area( 12.9 , 2.5 );
  double triangleArea = triangle::area( 4.0 , 3.0 );
  std::cout << "Rectangle area: " << rectangleArea << std::endl
            << "Triangle area: " << triangleArea << std::endl;
}
```

Problem Definition (cont'd)

Problem Definition (cont'd)

1. *Ahmed* has to implement the rectangle area function

$$A = wh$$

Problem Definition (cont'd)

1. *Ahmed* has to implement the rectangle area function

$$A = wh$$

2. *Emad* has to implement the triangle area function

$$A = \frac{bh}{2}$$

Problem Definition (cont'd)

~~Possible~~ Awful Solutions:

- Ahmed finishes the whole project alone.

Problem Definition (cont'd)

~~Possible~~ Awful Solutions:

- Ahmed finishes the whole project alone.
- Emad finishes the whole project alone.

Problem Definition (cont'd)

~~Possible~~ Awful Solutions:

- Ahmed finishes the whole project alone.
- Emad finishes the whole project alone.
- They share intermediate codes through messenger, e-mail, or dropbox!

Problem Definition (cont'd)

~~Possible~~ Awful Solutions:

- Ahmed finishes the whole project alone.
- Emad finishes the whole project alone.
- They share intermediate codes through messenger, e-mail, or dropbox!
- They pass USB disk back and forth!

Problem Definition (cont'd)

~~Possible~~ Awful Solutions:

- Ahmed finishes the whole project alone.
- Emad finishes the whole project alone.
- They share intermediate codes through messenger, e-mail, or dropbox!
- They pass USB disk back and forth!
- They sit together to finish the project!

Problem Definition (cont'd)

What if?!

Problem Definition (cont'd)

What if?!

- What if we have a team of 8 members.

Problem Definition (cont'd)

What if?!

- What if we have a team of 8 members.
- What if your application was as big as 20000 lines of code across tens of files.

Version Control Systems

Version Control Systems

- Keep track of all the changes that happened (No lost work).

Version Control Systems

- Keep track of all the changes that happened (No lost work).
- Many Developers can work on the same file at the same time.

Version Control Systems

- Keep track of all the changes that happened (No lost work).
- Many Developers can work on the same file at the same time.
- The Version Control System will handle conflicts if possible, if not, it will ask the developers to check it.

Version Control Systems

- Keep track of all the changes that happened (No lost work).
- Many Developers can work on the same file at the same time.
- The Version Control System will handle conflicts if possible, if not, it will ask the developers to check it.

Popular Version Control Systems

Version Control Systems

- Keep track of all the changes that happened (No lost work).
- Many Developers can work on the same file at the same time.
- The Version Control System will handle conflicts if possible, if not, it will ask the developers to check it.

Popular Version Control Systems

- Git (we will use this)
- Mercurial
- Subversion (SVN)

Git

Git

- {Linus Torvaldos} developed Linux Kernel in 1991.

Git

- {Linus Torvaldos} developed Linux Kernel in 1991.
- Torvalds and others developed Git for management of Linux Kernel source in 2005.

Git

- {Linus Torvaldos} developed Linux Kernel in 1991.
- Torvalds and others developed Git for management of Linux Kernel source in 2005.
- Git is Free and Open Source.

Git

- {Linus Torvaldos} developed Linux Kernel in 1991.
- Torvalds and others developed Git for management of Linux Kernel source in 2005.
- Git is Free and Open Source.
- Great community support. You can always search in {Quora} and {Stackoverflow} for problems you face.

Typical Git Cycle

For your first experience with git, refer to this workflow.

Typical Git Cycle

For your first experience with git, refer to this workflow.

1. [First Time Only] Create/Clone Repository to your disk, so you have a local copy.

Typical Git Cycle

For your first experience with git, refer to this workflow.

1. [First Time Only] Create/Clone Repository to your disk, so you have a local copy.
2. Make changes to your source (edit/add new file).

Typical Git Cycle

For your first experience with git, refer to this workflow.

1. [First Time Only] Create/Clone Repository to your disk, so you have a local copy.
2. Make changes to your source (edit/add new file).
3. Add new files to your **repository system**. (You already created the files physically, but you need to ask the git repository to take control of your new file).

Typical Git Cycle

For your first experience with git, refer to this workflow.

1. [First Time Only] Create/Clone Repository to your disk, so you have a local copy.
2. Make changes to your source (edit/add new file).
3. Add new files to your **repository system**. (You already created the files physically, but you need to ask the git repository to take control of your new file).
4. Commit your changes.

Typical Git Cycle

For your first experience with git, refer to this workflow.

1. [First Time Only] Create/Clone Repository to your disk, so you have a local copy.
2. Make changes to your source (edit/add new file).
3. Add new files to your **repository system**. (You already created the files physically, but you need to ask the git repository to take control of your new file).
4. Commit your changes.
5. Get latest updates.

Typical Git Cycle

For your first experience with git, refer to this workflow.

1. [First Time Only] Create/Clone Repository to your disk, so you have a local copy.
2. Make changes to your source (edit/add new file).
3. Add new files to your **repository system**. (You already created the files physically, but you need to ask the git repository to take control of your new file).
4. Commit your changes.
5. Get latest updates.
6. Resolve any conflict (if any).

Typical Git Cycle

For your first experience with git, refer to this workflow.

1. [First Time Only] Create/Clone Repository to your disk, so you have a local copy.
2. Make changes to your source (edit/add new file).
3. Add new files to your **repository system**. (You already created the files physically, but you need to ask the git repository to take control of your new file).
4. Commit your changes.
5. Get latest updates.
6. Resolve any conflict (if any).
7. Push to the remote repository.

Create/Clone Repo

Create/Clone Repo

- Case 1: New Repository.

```
$ git init
$ git remote add [name] [URL]
```

Create/Clone Repo

- Case 1: New Repository.

```
$ git init
$ git remote add [name] [URL]
```

- Case 2: Existing Repository.

```
$ git clone [URL]
```

Track files

Track files

It is recommended to add file by file, so apply this command to all your application **souce** files, **exclude** any executable files or files generated by the compiler.

```
$ git add [file name]
```

Track files

It is recommended to add file by file, so apply this command to all your application **souce** files, **exclude** any executable files or files generated by the compiler.

```
$ git add [file name]
```

Or, alternatively, do it once for all files (not recommended, but it is up to you anyway).

```
$ git add *
```

Track files

It is recommended to add file by file, so apply this command to all your application **souce** files, **exclude** any executable files or files generated by the compiler.

```
$ git add [file name]
```

Or, alternatively, do it once for all files (not recommended, but it is up to you anyway).

```
$ git add *
```

`add` here means you are asking the repository to watch your files that already exists on disk.

Commit changes

Commit changes

- After making changes, you need to your repository to **confirm** these changes and documenting that change.
- Write a message that you can understand (e.g briefly, indicate your changes in the repository e.g "implementing square::area function").

Commit changes

- After making changes, you need to your repository to **confirm** these changes and documenting that change.
- Write a message that you can understand (e.g briefly, indicate your changes in the repository e.g "implementing square::area function").

```
$ git commit -a -m “I implemented square::area function”
```

Get latest source code updates

Before you publish your changes to the remote repository, update your repository in case some member of your team has made changes before you.

Get latest source code updates

Before you publish your changes to the remote repository, update your repository in case some member of your team has made changes before you.

```
$ git pull [remote name] [branch name]
```

Get latest source code updates

Before you publish your changes to the remote repository, update your repository in case some member of your team has made changes before you.

```
$ git pull [remote name] [branch name]
```

By default, *remote name* is `origin` and *branch name* is `master`, unless you made a new branch you are working on with your teammates.

Push source code changes

Publish your changes to your teammates on the remote repository:

Push source code changes

Publish your changes to your teammates on the remote repository:

```
$ git push [remote name] [branch name]
```

Push source code changes

Publish your changes to your teammates on the remote repository:

```
$ git push [remote name] [branch name]
```

Similarly, by default, *remote name* is `origin` and *branch name* is `master`, unless you made a new branch your are working on with your teammates.

Push source code changes

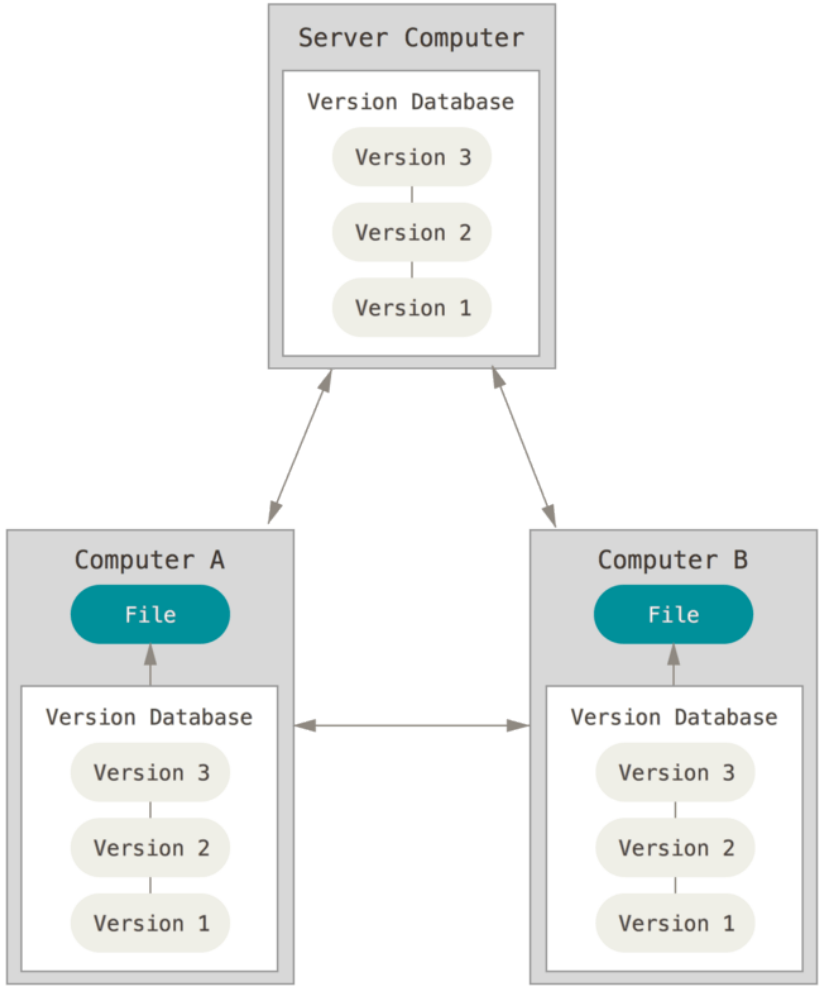
Publish your changes to your teammates on the remote repository:

```
$ git push [remote name] [branch name]
```

Similarly, by default, *remote name* is `origin` and *branch name* is `master`, unless you made a new branch your are working on with your teammates.

But what is **Remote Repository**, What do you mean

Local repository and remote repository



This photo is from {official git website}.

Git on the cloud

Popular servers offering free remote repository hosting:

Git on the cloud

Popular servers offering free remote repository hosting:

+ Bitbucket

Git on the cloud

Popular servers offering free remote repository hosting:

+ Bitbucket

- Github is offering you unlimited public and private repositories, your teammates per repository are limited to 5 members (Otherwise, pay). Unless you are a student. Everything is free!

Git on the cloud

Popular servers offering free remote repository hosting:

+ Bitbucket

- Github is offering you unlimited public and private repositories, your teammates per repository are limited to 5 members (Otherwise, pay). Unless you are a student. Everything is free!
- Bitbucket is offering you unlimited public and private repository, but your teammates for all repositories are limited to 5 members (Otherwise, pay).

What would you gain from keeping your projects on the cloud?

- Never lose your work

What would you gain from keeping your projects on the cloud?

- Never lose your work

Why Git on the cloud

Why Git on the cloud

- If you messed with your project, you can review your repository timeline and recover to a good state.

Why Git on the cloud

- If you messed with your project, you can review your repository timeline and recover to a good state.
- It is always safe to keep your projects on the cloud in one place.

Why Git on the cloud

- If you messed with your project, you can review your repository timeline and recover to a good state.
- It is always safe to keep your projects on the cloud in one place.
- Build a portfolio: always an elegant reference to your projects when you apply for a job. Include GitHub profile on your CV.

Very efficient way to demonstrate your skills

Example

{UK VISA: Tier 1 Exceptional Talent}

2. HOW DO I SHOW THAT I HAVE BEEN RECOGNISED FOR MY WORK OUTSIDE MY IMMEDIATE OCCUPATION?

You can demonstrate this by providing evidence that you have gone beyond your day-to-day profession to engage in an activity that contributes to the advancement of the sector. Examples may include mentoring, advising, organising interest groups, leading on policy, teaching at a university, or participating in clubs or societies for the furthering of the field.

Examples of relevant evidence include:

- Evidence of contributions to an Open Source project
- Your GitHub profile demonstrating active participation in a collaborative project
- Your StackOverflow profile showing significant contribution to discussions around code
- A link to one or more videos of talks or conferences that have had a significant viewership

A Special Gift for Bio2020 Class

Amr Mahmoud
@Amr_A_A_Mahmoud

#ThanksGitHub for this gift 😄 ❤️ from all Systems and Biomedical Engineering students class 2020 at Cairo University.

♡ 2 1:49 PM - Feb 8, 2018

See Amr Mahmoud's other Tweets

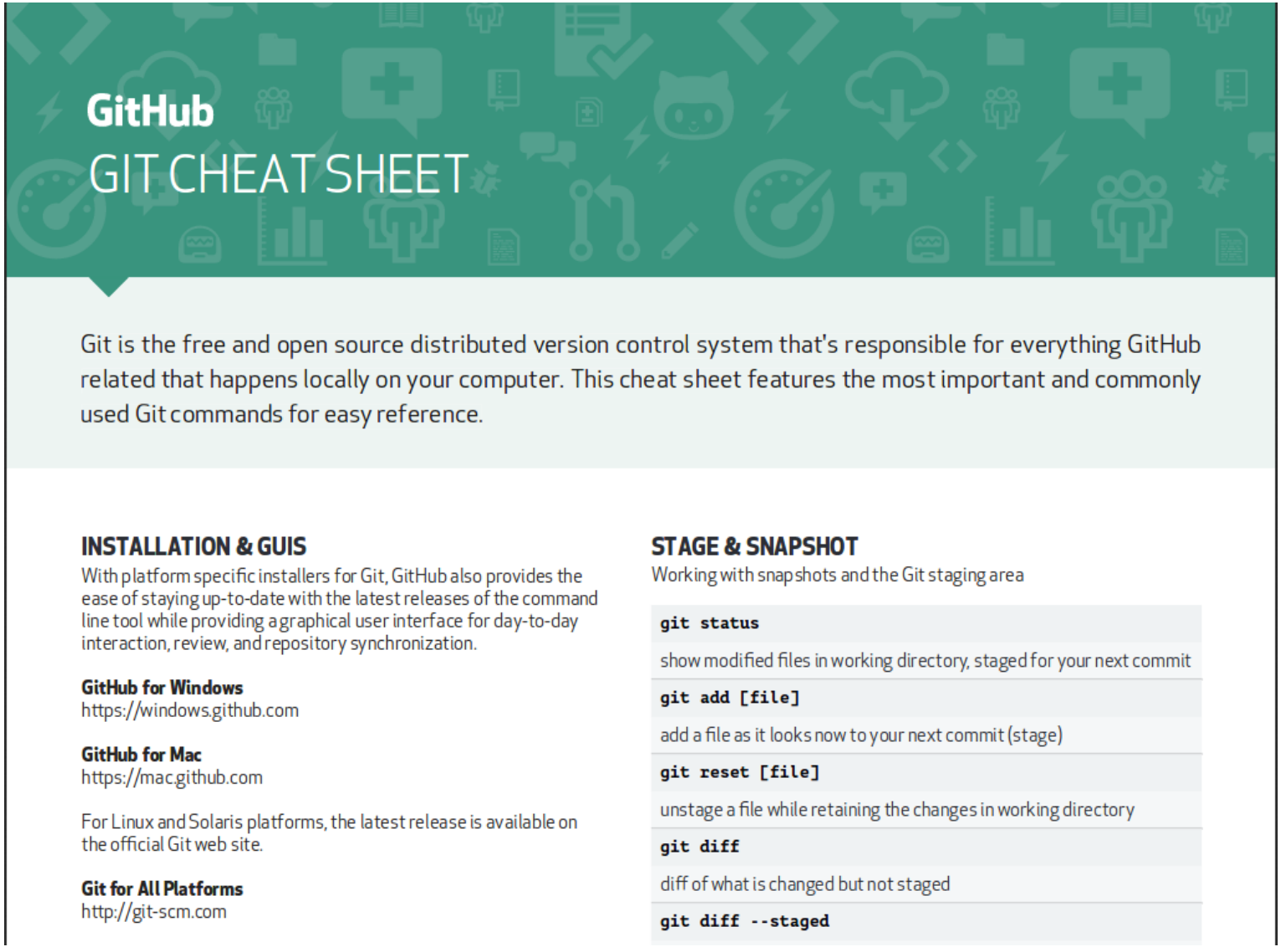
A Special Gift for Bio2021 Class

MouEhab
@_muhammedehab_

Sometimes the smallest things take up the most room in your heart. #ThanksGitHub for supporting us with your dear gift. We improved our studying process through GitHub network.
-From all systems and biomedical engineering students/class 2021/ at cairo university.❤️❤️

♡ 8 9:14 PM - Feb 14, 2019

Git cheat sheets



GitHub

GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

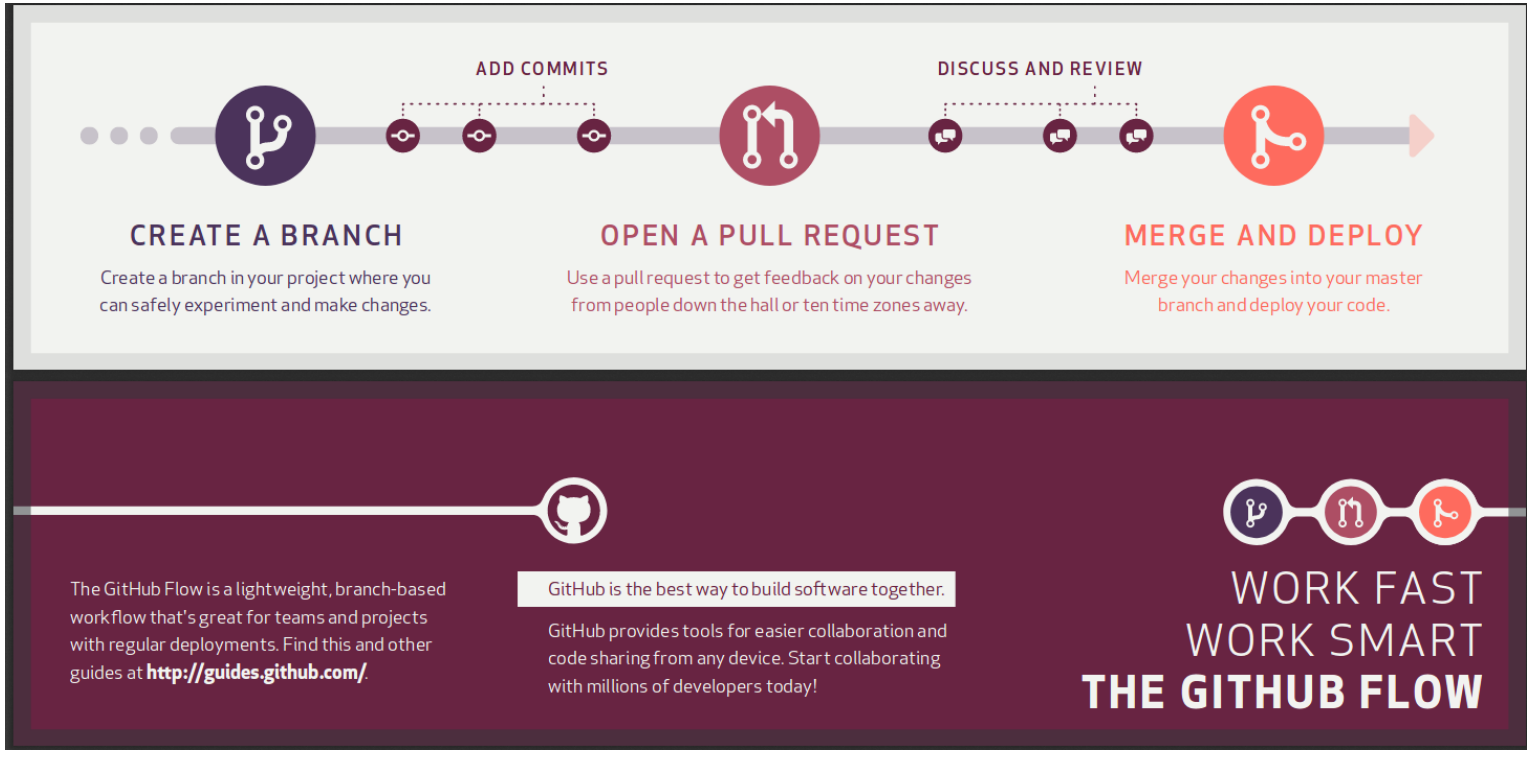
STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`
show modified files in working directory, staged for your next commit
`git add [file]`
add a file as it looks now to your next commit (stage)
`git reset [file]`
unstage a file while retaining the changes in working directory
`git diff`
diff of what is changed but not staged
`git diff --staged`

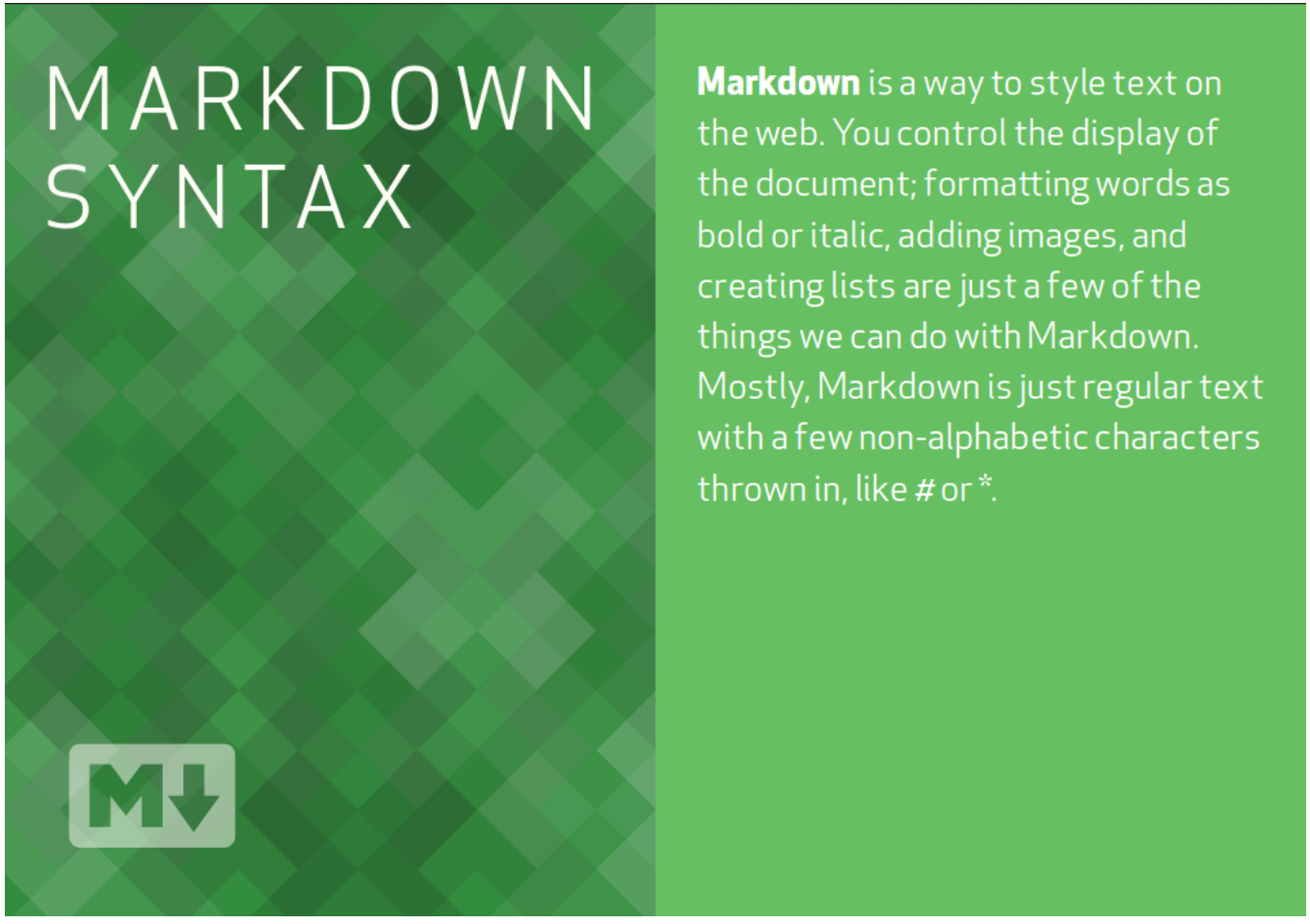
{PDF}

GitHub Flow guide



{PDF}

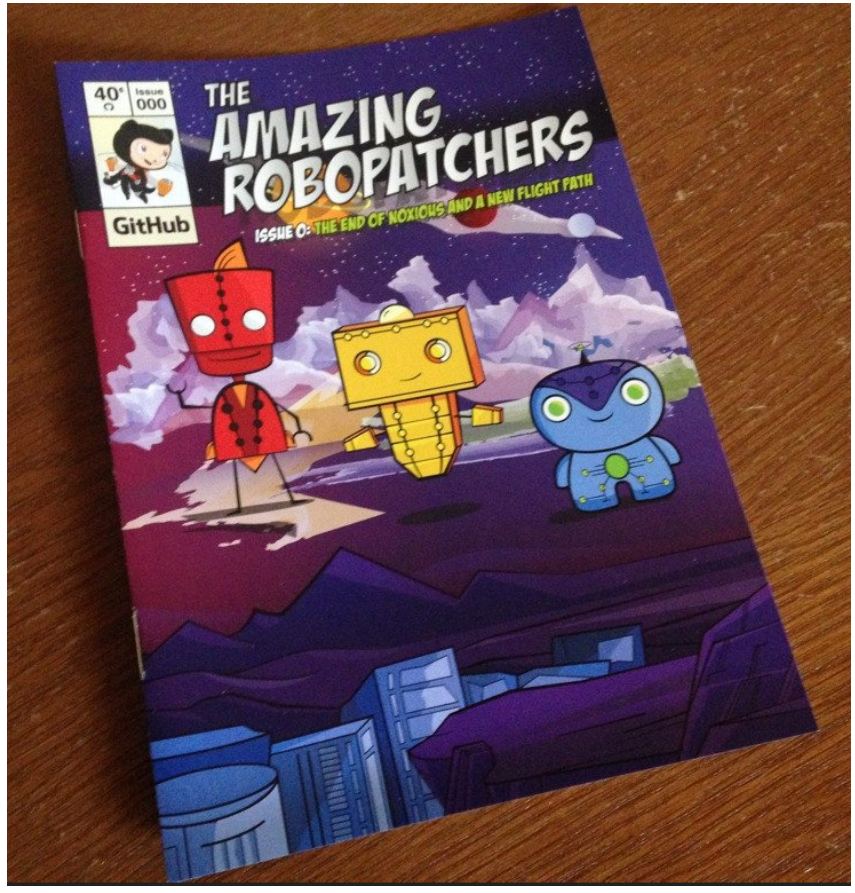
GitHub-Flavored Markdown guide



{PDF}

GitHub for Robotics comic book

explains the basics of using GitHub



{CBR}

Special Gifts for Best Students

Invertocat Hoodie

55.00

Invertocat 2.0 Shirt

25.00

Questocat Tee

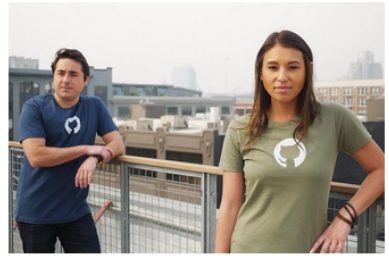
25.00

GitHub Drip Tee

25.00

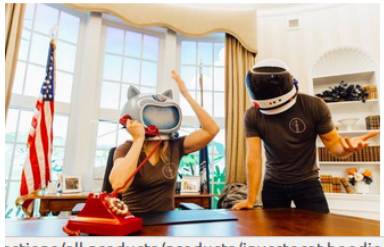
Social Coding Shirt

17.50



GitHub Username Shirt

25.00



ections/all-products/products/invertocat-hoodie

Special Gift from GitHub to SBME 2022 Class

Special Gift from GitHub to SBME 2022 Class

#ThanksGitHub

Installing Git on your machine

Issue the following command in your terminal.

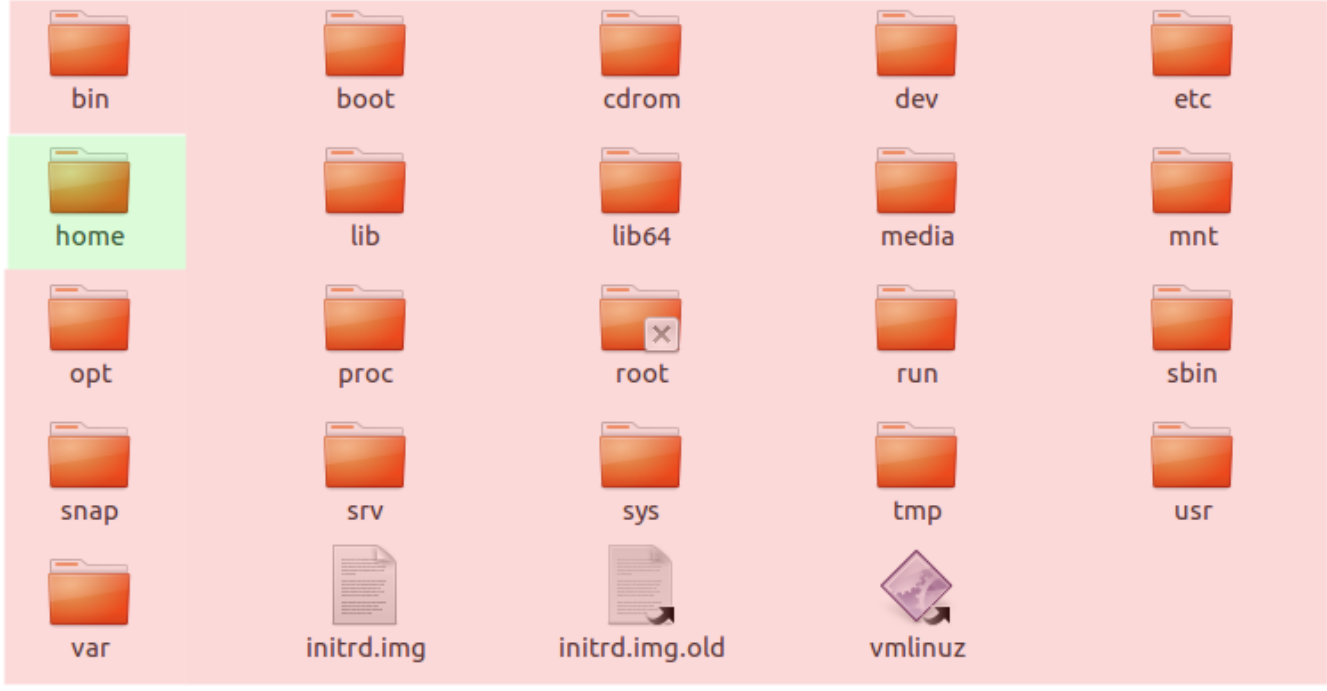
```
$ sudo apt-get install git
```

Homework

- Markdown resumes
- Basic C++
- To be announced soon

Linux Spaces

System-wise space vs. User space



- When working on your projects, you are a **USER**.
- When installing/upgrading system-wise application/library, you are an **ADMIN**.

Jumping between folders (changing directories)

```
$ cd (Relative Path|Absolute Path)
```

- In terminal commands, with `A|B`, I mean "Either A or B".

Listing files in the current directory (folder)

List files/directories inside the current directory of the terminal

```
$ ls
```

List files/directories on from other directory

```
$ ls (Relative Path|Relative Path)
```

Change folder name or moving folder name

```
$ mv (file|directory) (new file|new directory)
```

Copy file

```
$ cp (file) (target path)
```

Copy directory

```
$ cp -r (directory) (target path)
```

Create a new directory (folder)

```
$ mkdir (new folder name)
```

Removing a file

```
$ rm (file)
```

Remove a directory

```
$ rm -r (directory)
```

WARNING: Did you say `rm`?

HOW ABOUT `sudo rm -rf /`

DO NOT DO THIS!

```
$ sudo rm -rf /
```

WARNING: Did you say `rm`?

HOW ABOUT `sudo rm -rf /`

DO NOT DO THIS!

```
$ sudo rm -rf /
```

Updating & Upgrading your Linux

Upgrades are very important. Many hardware drivers issues are being fixed through these updates. Also, security-wise, updates guarantees your system to be safe against hackable vulnerabilities. For example, *Spectre* and *Meltdown* vulnerabilities that exposed all Operating Systems (including Widnows and Linux), for more info.

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Installing packages from the apt store

```
$ sudo apt-get install (package name)
```

Installing local `.deb` packages

```
$ sudo dpkg -i (package path)
```

Interesting Appliactions

Category	package name
Music & Video	vlc, rhythm box (shipped with Ubuntu)
PDFs	Okular, Foxit, PdfShuffler
Screenshots	Shutter
C++ IDEs	Qt Creator, Jet-brains CLion, VSCode
Python IDEs	Pycharm, Anaconda (Spyder)
Web IDEs	VSCode, Jet-brains WebStorm

Thank you