Quick Review

Data Types

- These are int, double, float, char, bool, string.
- Used to store data in the form of variables for your program.
- Always remember to initialize the variables, if you don’t we can’t predict the value of it.
- Know when to use which.
  - Use **int** for storing numbers (and not decimals) positive and negative numbers
  - Use **char** to store a single character in a single quote (‘a’, ‘x’ etc.)
  - Use **double** or **float** to store decimal numbers (positive or negative)
  - Use **bool** to store a true or false, mostly used in programs to make decisions (inside if)
  - Use **string** to store sequence of characters in a double quote (“abcd”, “hello” etc.)
- Remember the rules for naming (snake_case)
  - Names always start with an underscore or an alphabet, followed by alphabets, underscores and numbers

Data Types - An Example

- What data type do you use to store the value of
  - Number of pages in a book
  - Author of a book
  - Price of a book
  - Publication Year
  - Whether the book is a fiction or not
- Practice naming the variables for each of the above case and initialize them correctly. (Put in your own hypothetical values)

Data Types - Another Example (Take home)

- What data type do you use to store the value of
  - Name of a city you visited during summer
  - The temperature of the city
  - The population of the city
  - The postal code of the city
  - If the city was above 5000 feet in altitude
- Practice naming the variables for each of the above case and initialize them correctly. (Put in your own hypothetical values)

Header Files

- iostream - For using cin, cout, endl statements
  - cout - Prints to the screen
  - cin - Takes an input from the user into a variable
- iomanip - For fixed and setprecision manipulators
  - fixed - sets the default notation to decimals for floating point numbers
  - setprecision (number) - sets the number of decimal places to print
  - **Use all the above only in cout statements directly**
- You can cascade (use multiple) cin and cout operators
  - cin >> x >> y >> z; -> Takes in 3 values
  - cout << “Hello : “ << username << endl;

Arithmetic

- Basic operators: +, -, *, /
- Know the order of operators precedence
  - Ex: 9 + 5 / 3 - 6 * 4 + 12
  - Hint: To evaluate such expressions, group them.
  - 9 + (5 / 3) - (6 * 4) + 12 -> 9 + (1) - (24) + 12 -> **-2**
- Modulus Operator - Used with two operators, as (a % b). It returns the **remainder** when a is divided by b.
  - Ex: 73 % 5 = 3, int y = 12 % 7;
  - Simple application we saw: Check if a number is even or odd.
- To do square root, and exponents -> cmath library has functions you can use.
  - sqrt(x) -> Gives the square root of x
  - pow(x, y) -> Gives the value of x raised to the power of y.

Type Casting

- What does the below yield?
  - 7 / 4
  - 5 / 9
- The decimal value is truncated, because an integer divided by an integer is treated as an integer in C++.
- Even if you do : `double result = 7 / 4;` It's still not a decimal number.
- Type Casting - Converts a variable from one type to another (Commonly used between ints and doubles, vice versa too).
  - Do `double result = 7.0 / 4`
  - Double result = static_cast<double> (7) / 4;

Relational, Boolean & Decisions

- Know the syntax and write it down in the cheat sheet you're allowed.
- Know how relational and boolean operators are used
  - **Relational:** < (less than), >, ==, !=, <= and >=.
  - Used for comparing variables of similar data types.
  - Ex: **x <= y**, **7 >= 2.5**, **city == "Boulder", value != 3.14.**
  - **Boolean:** &&, || and !
  - Used for combining multiple relational operators
  - Ex: **x <= y && 7 >= 2.5, city == "Boulder" || value != 3.14.** Revisit more examples discussed in lecture slides
- If/else and switch statements - Used to implement decision logic - `Should the program execute this or that`

Points to remember for If statements

- If statement needs an expression (something) that evaluates to a true or false
- If required, combine as many relational and logical expressions you need inside an if statement condition.
- Don’t put semicolon after if (<condition>)
- Don’t mistake ==, with a single =. First is a comparison, and second is an assignment (like we’ve seen in variables being assigned values)
- Understand the flow of if statements.

Flow of if statements

- If statements can be followed by else ifs or else (it's all optional)
- The else if's and else statements must be part of one if statement.

```
int value = 7;

if (value == 0) {
    cout << "Value is 0" << endl;
}
if (value > 0) {
    cout << "Greater than 0" << endl;
}
else if (value > 10) {
    cout << "Greater than 10" << endl;
}
else if (value < 5) {
    cout << "Less than 5" << endl;
}
if (value < 2) {
    cout << "Less than 2" << endl;
}
else {
    cout << "Inside else" << endl;
}
```

Switch Statements

- Alternative to if statements, but used only when you need to compare for equality. Can’t do inequalities with switch.
- Optimal for a menu based program

```
char x = 'a';

if (x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u') {
    cout << "Vowel" << endl;
}
else {
    cout << "Not a vowel" << endl;
}
```

```
char x = 'b';

switch(x) {
    case 'a':
        cout << "Vowel" << endl;
        break;
    case 'e':
        cout << "Vowel" << endl;
        break;
    case 'i':
        cout << "Vowel" << endl;
        break;
    case 'o':
        cout << "Vowel" << endl;
        break;
    case 'u':
        cout << "Vowel" << endl;
        break;
    default:
        cout << "Not a vowel" << endl;
}
```

Functions - The tricky one!

- What is it - > Reusable block of code
- Why -> Reusable… uggh same answer

  Anyway, know the syntax

- 4 parts
  - *Function name* -> camelCasing and an appropriate name (probably a verb for the first word)
    getArea, **calculateGrade**, **findAverage**, **greetUser**
  - *Function parameters* -> The inputs to the function. Mention the data types and the name of the parameter variables.
    getArea(**float radius**), calculateGrade(**int score**), findAverage(**int first, int second**), greetUser(**string name**)
  - *Function return type* -> The output from the function. Mention the data type only (can be void too! if the function doesn't need to return anything)
    float getArea(float radius), **char** calculateGrade(int score), **float** findAverage(int first, int second), **void** greetUser(string name)
  - *Function body* -> Depending on what you need to do!

Calling a Function

- Well you wrote a function, that does something. Now invoke it (or call it) so the function does what it's intended to do.
- Who calls it?? - Maybe main function.

Things to remember

- Writing a function was the first task, but the second simpler one is calling it.
- "Arguments" are passed from main function as input. Here there is only one argument - radius
- What you provide is what the function gets, so
  - Pass the arguments in the correct order as the function expects
  - Provide the correct data types
  - *But the names of the variables can be whatever you want*

```
5   float getArea(float radius) {
6       return 3.14 * radius * radius;
7   }
8
9   int main() {
10      float radius = 1.0;
11
12      float area = getArea(radius);
13      cout << area << endl;
14
15      return 0;
16  }
```

Regarding Function

- Know and identify parts of the function
- Know how you can write a function given a problem
  - Just need to figure out what the inputs should be, and what the output should be
- Identify if the function is returning anything or not (void)
- Function Signatures (or prototypes)
  - These are very similar looking to function definitions, except they don't include the body.
  - You just tell the compiler, hey.. I'm going to be writing what the function will do at a later point in time, but for now just know that this exists.

```
float getArea(float radius); // Function Prototype
// At this step compiler only knows that this function will be completed sometime later, with the body
// Hence the name prototype!

int main() {
    float radius = 1.0;

    float area = getArea(radius);
    cout << area << endl;

    assert(area == 1);

    return 0;
}

float getArea(float radius) { // Function Definition
    return 3.14 * radius * radius;
}
```

Testing and Asserts

- Asserts are statements in C++ to check for a condition that **must** be true.
- Can be used anywhere in a program, for instance
  - assert(x == 5);
  - assert(city_temperature >= 70.5);
  - assert(getArea(radius) == 3.14);
- If assert fails, or if the condition is false, then the program is terminated