Data Types

What is a data type?

NOTE: This guide is based on C++ but should be similar in many programming languages.

Data types refer to **the type of data that a particular variable can store**. A variable is a "bucket" in which data can be stored.

Example 1:

- An *int* variable can only store integers - not numbers with decimals or strings. Data types are needed because **the compiler needs to know how much memory to allocate for each variable. Every single data type has its own size and memory needs.**

Example 2:

- A variable with the data type *int* needs four bytes of memory, while a variable with the data type *double* needs eight bytes of memory. Knowing what datatypes you have at your disposal is important: **memory management**. A programmer should be able to discern when to use each data type to their advantage; if you do not need decimal numbers, then there is no need to declare a *double* variable and spend those extra four bytes.

What data types are there?

There are three groups of data types: **Primitive Data Types, Derived Data Types,** and **Abstract/User-Defined Data Types**[1]. For this handout, we will only focus on the former two.

- **Primitive Data Types:** Refer to the built-in data types in C++. These include *int, char, bool, float, double.*

  NOTE: Void is the only data type on this list that **cannot be used for variables.** Instead, it is used for functions to signify that the function does not return anything.

  - *Integer (int)*: Stores integers or put simply, **numbers with no decimal values**. They can be any integer from –2,147,483,648 to 2,127,483,647. The memory required for a regular *int* type is four bytes.[2]

- *Character (char)*: **Stores a single character** (letter, number, special characters like "@"). A character variable is a single byte.
- *Boolean (bool)*: **Stores true or false values**. They are particularly useful when working with conditional statements since they allow us to **evaluate a condition and introduce control flow to our program**. For example, if we have an if-statement that says if value == true, then that block of code will only run if the value is true at that point in the program. This allows the programmer to control the program even after compilation.
- *Single Precision Floating Point Number (float)*: **Stores numbers with seven decimal places**. While float used to be the most used data type when implementing numbers with decimals, it is not used as frequently due to the greater amount of memory found in modern systems. This data type needs four bytes to be declared.[3]
- *Double Precision Floating Point Number (double)*: This data type is a **more accurate version of float** since it allows for fifteen decimal places to be recorded. The name "double" comes from the decimal places recorded and the amount of memory needed is doubled from a float. As such, doubles need eight bytes to be used in a program.[3]

- **Derived Data Types:** Refer to types that are built from the Primitive DTs. These include *functions*, *arrays*, *pointers*, and *references*. **These data types are used to expand the capabilities of primitive data types or avoid code repetition**.
  - *Functions*: **Reduces the repetition of code**, hides code, and compartmentalizes the many methods that a program can have. Fundamentally, functions compute calculations and then return a value. The programmer can define this value to suit their needs – using PDTs, functions can return values that help the programmer complete their needs.

- *Arrays*: **Allows access to many variables of a particular data type within a single variable.** For example, if we declared "int arr[10]", we would have access to ten sequentially stored integer variables with a single line of code! Arrays can be immensely powerful when we need multiple variables. To access the variables inside, simply change the number between the square brackets – please note that arrays start at 0, so using the example above, we can access arr[0] to arr[9].
- *Pointers*: **Allows us to store a memory address that holds a value of the declared type**. For example, if we declare the following:

  ```
  int myVar = 20;
  int* myPointer = &myVar;
  cout << "The address of myVar is: "<< myPointer << endl;
  cout << "The value of myVar is:" << *myPointer << endl;
  ```

  The first *cout* will output a memory address that looks like "0x313EF02A", which is the address where myVar is stored in main memory (RAM). On the other hand, the second *cout* will output the actual value of myVar, using the pointer variable we declared once again. In short, **pointer variables point to addresses that store data of the data type we choose.**[1]
- *References*: Creates aliases for our variables. Let us look at this example code:

  ```
  int myVar = 20;
  int& alsoMyVar = myVar;
  alsoMyVar = 30;
  cout << myVar << endl;
  ```

  When we output myVar at the end, its value will be thirty, since myVar and alsoMyVar are referring to the same variable; **any changes made to one will reflect on the other** since they are the same. [1]

References:

1. GeeksforGeeks. (2020, February 26). *Derived data types in C++*. GeeksforGeeks. https://www.geeksforgeeks.org/derived-data-types-in-c/
2. GeeksforGeeks. (2023, March 18). *C++ data types*. GeeksforGeeks. https://www.geeksforgeeks.org/cpp-data-types/#
3. W3Schools. (n.d.). *C++ data types*. W3Schools. https://www.w3schools.com/cpp/cpp_data_types.asp

Disclaimer: We did not include all of the resources conferred to formulate this handout. We encourage students to conduct further research to find additional resources. The format of this list is not commensurate with a standard format.