Programming Notes 1: Data Abstraction

Always remember, I will provide examples of the notes in the example program.

1. Intro to data abstraction

Data abstraction is essentially how data is held in a program, and how you, the programmer, can use it. The process itself is pretty simple, but there are some complicating factors. In computers, all data is held in a series of 0's and 1's, that is, binary, and humans are not too great at manipulating large quantities of numbers. For example, a computer might hold the data value 19 as 00000000 00010011. Thankfully, most high level programming languages, such as C++, will do this low-level data management for you, and all you have to deal with are variables (that's not to say you can't do low-level memory management – we'll talk about that later). The "covering up" process is called abstraction—the process of using one thing that represents another. Your programming language abstracts the concept of low-level data, allowing you to not worry about the finer details of it, and use much simpler, human-understandable data.

2. Identifiers

Now, to use this data, your programming language allows you to store data in variables, which are essentially containers that hold your values – see section 3 for more detail. Variables are a type of identifier, which is any uninterrupted (i.e., no spaces) combination of characters and digits. Identifiers are used to represent various parts of a program, including variables, functions, and more. It is standard for identifiers to follow the format camelCase, which means that the first word is not capitalized, and each following one is.

Examples:

payRate

userAge

findMaxValue

calcDiscriminant

They must start with either a normal character or an underscore, and can be as long as you want. You can use any identifier you want, as long as it is not reserved by the C++ language (for a list of these, look at the end of this document). In general, you want to use noun identifiers for "things," which would be variables or other objects that contain actual information, and verb identifiers for other functions which do things, rather than hold values.

3. Literals vs. Variables

While a variable is an identifier that holds a value, a literal is the value itself. You won't be using too many literals in your programming, but keep in mind that they are things like a number, a letter, or a sequence of words. For example,

"Hello!"

5

345.7

'A'

are all literals.

4. Fundamental/derived data types

In C++, there are four fundamental data types. What is a fundamental data type, you may ask? Well, fundamental types represent physically different types of data which are stored in computer memory. Everything other than these four types are built out of them, and are called derived data types. The four fundamental types are:

Full Data Name	C++ Identifier	Data Type
Character	char	Technically, a character is just an integer that can only hold values between -128 and 127. The value corresponds to a character (a single letter or something like a bracket) using a standard convention. For example, the letter 'G' is represented by the value 71.
Integer	int	Any positive or negative integer.
Floating point value	float, double	Any positive or negative fractional quantity. These values are held in memory in the form of scientific notation. You will most often use "double," as it can contain a more precise value than "float."
Boolean	bool	Only the values 'true' or 'false,' but how they are actually stored in memory is more complicated.

The only common derived data type we will be using for a while is "string." A string holds any combination of characters. For example, a string can hold "My name is George." We'll be looking at these closer later, but for now just know they hold anything you can type.

5. Using variables

Now, you must be wondering how to actually use these variables that I've talked of for so long. In C++, all variables must be explicitly declared, which is to say you must tell the computer your variables data type when you create it.

To declare a variable, the syntax is simple: simply type the name of the data type you are using, and then your variable identifier, and end the line with a semicolon. Get used to that—almost every line in C++ is ended with a semicolon. Finally, remember it is conventional to declare all variables at the start of a function. Examples:

```
int myAge;
```

Declares a variable named myAge with the type integer.

```
double payRate;
```

Declares a variable named payRate with the type double.

To use your new variables, you use the assignment operator ('=') to assign a value to them. The assignment operator always takes what is on the right and assigns it to the identifier on the left. This can also be combined with the declaration. When assigning an integer or double value, you can simply type your number. However, when assigning a character value, wrap your character in single quotes, and when assigning a string, wrap your phrase in double quotes. Examples:

myAge = 24;

The variable myAge now holds the value 24.

double payRate = 23.50;

This creates the variable payRate, and makes it hold the value 23.5.

char letter = 'U';

The variable letter now holds the letter 'U'.

string username = "xxx_billy_xxx";

The variable username now holds the phrase "xxx_billy_xxx".

Now, whenever you use your variable, your program will use the value held within your variable.

6. Scope

Finally, there is the topic of scope. The scope of a variable is the area in which it can be used, or to put it simply, the area between the opening and closing curly braces in which your variable was first declared. Keep this in mind when declaring variables, as they will not be useable except in the scope you declared them in. For example:

{ – Begins the scope

int myVar;

Here, myVar can be used.

} – Ends the scope

Here, it can't.

Now, you may wonder what happens if the variable is not declared in any scope. This is called a global variable, as it can be accessed from anywhere in the program. Do not use these (except in very rare situations), as global variables can lead to very hard-to-find bugs in your program.

7. C++ reserved keywords:

You can't use these as identifiers.

alignas	decltype	new	struct
alignof	default	noexcept	switch
and	delete	not	template

and_eq	do	not_eq	this
asm	double	nullptr	thread_local
auto	dynamic_cast	operator	throw
bitand	else	or	true
bitor	enum	or_eq	try
bool	explicit	private	typedef
break	export	protected	typeid
case	extern	public	typename
catch	false	register	union
char	float	reinterpret_cast	unsigned
char16_t	for	requires	using
char32_t	friend	return	virtual
class	goto[1]	short	void
compl	if	signed	volatile
concept	inline	sizeof	wchar_t
const	int	static	while
const_cast	long	static_assert	xor
continue	mutable	static_cast	xor_eq

[1] Never use "goto." You will be shamed.