Switch Statements

```
switch (variable)
{
	case 1:
		do dis ding;
	break;
	case 2:
		do dis ding;
	break;
	default:
		last option;
}
```

For Loops

```
for (initialization; condition; change)
{
	do dis;
	and dis;
	all deez;
}
```

While Loops

```
while (condition)
{
	put your left foot in;
	put your left foot out;
}
```

Do-While Loops

```
do {
	put your right foot in;
	put your right foot out;
	put your right foot in;
	shake it all about;
} while (condition);
```

If-Else Statements

```
if (condition)
{
	soulja boy off in dis hoe;
}
else if (condition)
{
	watch me crank it;
	watch me roll;
}
else if (condition)
{
	watch me crank dat soulja boy;
	den superman dat hoe;
}
else
	now watch me uuuuuuuu;
```

Boolean (&& (and)/ ||(or))

```
if (iggy && azalea)
{
	im so fancy;
	you already know;
}

if (elsa || ice queen)
{
	let it go;
	let it go;
}
```

Operators	
%	Modulas. Gives remainder. (14 % 4 = 2) (3 % 10 = 3)
==	Equal to. Compares two things. (8 == 16/2) → True
!=	Not equal to. Compares. (18 != 17) → True
<=	Less than or equal to. Compares. (4 <= 10/2) → True
>=	Greater than or equal to. Compares. (6 >= 10/2) → True
x++/x--	Increment/Decrement. If x = 5: x++ = 6; x-- = 4

Variable types		
Type	Meaning	Values held
int	integer	6, 18, -25, 0
double	decimal numbers	4.5, -3.2, 12
char	single character	'$', 'a', 'Z'
string	multiple characters	"phat","booty"

Declaring Variables

```
type name;
int eight;
double yNotEight = 7.6; // initialization
char cashMoney = '$';
string femaleGymnast = "Kaitlyn";
```

Notes on Variables

- The int type truncates decimals, i.e. does not round up. int x = 7.6 → x = 7; int x = 9.2 → x = 9;
- An int divided by a double returns a double, a vice-versa.

Programming Errors		
Compiler	An error in code that prevents compiling.	Missing semicolon, typo
Logic	The program doesn't perform as intended.	Math error, loop error
Runtime	The program ends unexpectedly.	Infinite loop, uninitialized variable

"Strings"	
s[k]	Gives the character at that position in the string. s[0] is the first character.
s.size()	Gives the length of the string.
getline(cin, s)	Gets the entire input line, ignoring spaces.
cin.ignore(1000, '\n');	When transitioning from integer input to strings.
cout.setf(ios::fixed); cout.precision(2);	Sets precision to two decimal points.

Arrays

Declaration

```
int ebolaDeathsByState[50];
```

Initialization

```
string dakotasBaes[1] = {"Fernando Pacheco"};
int myIncomeByWeekday[7] =
{0, 0, 0, 0, 0, 0, 0};
```

Notes on Arrays

- Is a string of continuous data. Like this:

0	1	2	3	4	5

- Position starts at 0, i.e. a[0] is the first element of the array.
- Multidimensional arrays exist. Like this:

yoDawg[2][2]

	0	1
0	yoDawg[0][0]	yoDawg[0][1]
1	yoDawg[1][0]	yoDawg[1][1]

C-Strings

- An array of characters, with a null char at the end.

```
char swift[11] = {"shakeitoff"};
```

s	h	a	k	e	i	t	o	f	f	\0

- This means that the array size is one bigger than the word length.

Initialization

```
char anaconda[] = {"Nicki Minaj"};
```

(This is an array of size 17, counting all characters, the spaces, and the null at the end.)

You can also over-declare the size:

```
char holidayQuestions[900] =
	{"How's college?"};
```

C-String Functions #include <cstring>	
strlen(s)	Returns the number of interesting characters in the string, i.e. everything but the null.
strcpy(s, t)	Copies the string from t and puts it in s. (*Careful! If t is bigger than s, undefined behavior can occur!)
strcat(s, " in bed")	Concatenates. Adds second part to first.
(*) #define _CRT_SECURE_NO_WARNINGS	Needed in Visual Studio to use strcpy, or else it will limit you to 'strcpy_s' to prevent copying in c-strings that are too large.

Pass By Value vs Pass By Reference

- Pass by Value makes a copy of the variable and stores that in memory, this newer variable is then deleted at the end of the function's execution
- Pass by Reference alters the value at the memory location of the variable passed, denoted by an ampersand&

```
void polarToCartesian (double rho, double theta, double& xx, double& yy)
{
	xx = rho * cos(theta);
	yy = rho * sin(theta);
}
```

1- Make sure you initialize a C-String before using it, otherwise random characters will be printed out	2 - Always terminates with the zero byte char dog[3] = "cat"; // Not valid, no room for zero byte	3) Don't have to put the zero byte in char s[6] = "hello\0"; //Compiler will try to add another zero byte and result in a runtime error

4) char dog[6] = "c\0a\0t" //
cout << dog << endl;
// Acts more like a string, encounters zero byte and stops printing
cout << dog[2] << endl;
// Acts more like an array, prints out third element of array
	5) char dog[] = { 'c', 'a', 't'};
	cout << dog << endl;
	No zero byte, this causes a compilation error
	
	The zero byte must be explicitly listed when initializing like an array
		6) char dog[] = { 'c', 'a', 't', '\0'};
		cout << dog << endl;
		This is fine

7) char dog[6] = { 'c', 'a', 't', '\0'}; cout << dog << endl; Fine but trying to print out dog[4] or dog[5] results in undefined behavior	8) char dog[6]; dog = { 'c', 'a', 't', '\0'}; cout << dog << endl; // Wrong, cannot do this in separate lines	9) char dog[6]; dog = "cat"; cout << dog << endl; // Wrong for the same reason as #8
10) char dog[6]; dog[0] = 'c'; dog[1] = 'a'; dog[2] = 't'; dog[3] = '\0'; cout << dog << endl; // This is fine because it follows proper syntax rules	11) You can assign a C++ string to a C-String char dog[] = { 'c', 'a', 't', '\0' }; string s = dog; You CANNOT add a C-String to a C-String using + Use strcat for that	12) You can add a C++ string to a C-String using + You CANNOT add a C-String to a C-String using + Use strcat for that

Phone Number Example

```
string digitsOf(string pn)
{
	string digitsOnly = "";
	for (int i = 0; i < pn.size(); i++)
	{
		if (isdigit(pn[i]))
			digitsOnly += pn[i];
	}
	return digitsOnly;
}
```

Character Functions

```
isdigit(char c)
islower(char c)
isupper(char c)
isalpha(char c)

#include <cctype> // defines isdigit, islower, isupper, isalpha, etc.
using namespace std;

string s;
getline (cin, s);

if (s != "" )    // could be undefined behavior if passed an empty string (s.size() > 0 also valid)
	s[0] = toupper(s[0]); // returns the uppercase version of the char passed

string t = "aB9 ?";
char c0 = toupper(t[0]);  // c0 is 'A'
char c1 = toupper(t[1]);  // c1 is 'B'
char c2 = toupper(t[2]);  // c2 is '9'
char c3 = toupper(t[3]);  // c3 is ' '
char c4 = toupper(t[4]);  // c4 is '?'
```

C-Strings

- Essentially an array of char values, ending with the zero byte

char t[10] = “Hello”;
char s[100]; // Note: not initialized to the empty string, it will be set to garbage values

char s[100] = “”; // Actually sets all values to a zero byte

‘\0’ - the zero-byte, the physical representation of a char value set to “nothing” in memory

FUNCTION	DESCRIPTION	CAUTIONS
`strcpy(`*Target_String_Var*`,` *Src_String*`)`	Copies the C-string value *Src_String* into the C-string variable *Target_String_Var*.	Does not check to make sure *Target_String_Var* is large enough to hold the value *Src_String*.
`strncpy(`*Target_String_Var*`,` *Src_String, Limit*`)`	The same as the two-argument `strcpy` except that at most *Limit* characters are copied.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcpy`. Not implemented in all versions of C++.
`strcat(`*Target_String_Var*`,` *Src_String*`)`	Concatenates the C-string value *Src_String* onto the end of the C-string in the C-string variable *Target_String_Var*.	Does not check to see that *Target_String_Var* is large enough to hold the result of the concatenation.
`strncat(`*Target_String_Var*`,` *Src_String, Limit*`)`	The same as the two argument `strcat` except that at most *Limit* characters are appended.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcat`. Not implemented in all versions of C++.

Input
cin.getline(s, 100);

cout << strlen(t); // Gives the length of t

strcopy(s, t); // string copy (destination string, source string)
// Note: the destination string needs enough room for source string + the zero byte

if (t < s) // Compiles but does not check the contents of s and t, rather compares the memory location of s and t

if (strcmp(x, y) < 0) // Given two c-strings, returns a negative value if x comes before y
// Returns a 0 if x == y
// Returns a positive value if x comes after y

FUNCTION	DESCRIPTION	CAUTIONS
`strlen(`*Src_String*`)`	Returns an integer equal to the length of *Src_String*. (The null character, `'\0'`, is not counted in the length.)	
`strcmp(`*String_1*`,` *String_2*`)`	Returns `0` if *String_1* and *String_2* are the same. Returns a value < 0 if *String_1* is less than *String_2*. Returns a value > 0 if *String_1* is greater than *String_2* (that is, returns a nonzero value if *String_1* and *String_2* are different). The order is lexicographic.	If *String_1* equals *String_2*, this function returns `0`, which converts to `false`. Note that this is the reverse of what you might expect it to return when the strings are equal.
`strncmp(`*String_1*`,` *String_2, Limit*`)`	The same as the two-argument `strcat` except that at most *Limit* characters are compared.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcmp`. Not implemented in all versions of C++.