Headers:
Basic I/O: #include <iostream>
Strings: #include<string>
Char Funcs: #include<cctype>
CString #include<cstring>
Standard: using namespace std;

Variable Types

short	Integer -32,768 to 32767
int	Integer -2.14M to 2.14M
long	Integer -2.14M to 2.14M
float	Number -10^{38} to 10^{38}
double	Number -10^{308} to 10^{308}
char	Single char in '': 'a', '\n'
bool	true, false
string	Char sequence in "": "ab"

Variable names can only contain letters, numbers, underscores, and cannot begin with numbers.
Example: a, text, count, _i, count_i

Input:
Chars / Numbers: cin >> var;
String: getline(cin, var);
CString: cin.getline(var, len);
Clearing: cin.ignore(10000, '\n');

Output:
Print: cout << var;
Decimal: cout.precision(x);
Fixed: cout.setf(ios::fixed)

If Statements:
```
if (condition)
    …
else if (condition)
    …
else
    …
```

Attribution evaluates the value being attributed:
```
int a;
int b = 0;
if (a = b)
    …
else
    …
```
Will evaluate to else since 0 is bool for false.

Switch Statement:
```
switch (param) {
    case 1: …
        break;
    case 2:
    case 3: …
        break;
    default: …
}
```

Parameter can be int or char.

For Loop:
```
for (int k = 0; k < num; k++)
    …
```

Iterates k at the end of the loop, even if next case fails.
```
for (int k = 0; k == 0; k++) {}
```
k is iterated to 1, then loop breaks.

While Loop:
```
while (k < num) {
    …
    k++
}
```
Iterate at the end of the loop.

Operators:
Math: +, -, *, /, %
Comparative: ==, !=, <, <=, >, >=
Logic: &&, ||

&& takes priority over ||.
Condition of if statement reads L to R.
! also applies to bool functions.
Example: !isalpha

int m = 5.6: will store 5.6 in m.
int m = 11/5: will store 2 in m.
int m = 2.6/0.5: will store 5 in m.
Calculates before storing.

Characters:

\n	New line
\t	Tab
\\	Backslash
\'	Single quote
\"	Double quote

cctype functions include:
isdigit(char): checks if number.
isalpha(char): checks if letter.
isupper(char): checks if uppercase & letter.
islower(char): checks if lowercase & letter.
toupper(char): changes to uppercase.
tolower(char): changes to lowercase.

For transform functions, if char is not a letter, do nothing.

Common Errors:
- **Missing Semicolon**
- Unmatched semicolons
- Unmatched quotes
- = instead of ==
- Division of integers (e.g. 5/2 = 2)
- Declaring var inside loop
- Using undeclared variable
- , instead of ; in for parameters
- Forgetting to break in switch
- Infinite loop (runtime error)
- Empty char (compilation error)
- Type mismatch (int x = 2.99)

Strings
string.size(): gives the length of the string.
string[k]: gives the character at index k.

Loop character by character using:
```
for (int k = 0; k != s.size(); k++)
    …
```

+ operator can append two strings.

Arrays
Parameter in function is defined by var[]:
int function(string a[]);
Calling a function with array parameter:
function(a);

Arrays can be declared like this:
int foo[] = { 10, 20, 30 };
Assumes size of 3.
int bar [5] = { 10, 20, 30 };
Empty elements are set to default values (normally 0).
Position values start from 0, for example:
cout << bar[1] prints out 20.

2D arrays: foo[row][column]
When declaring the array parameter:
- Leave the first pair of square brackets empty.
- Supply the actual declared size for the remaining dimensions.

Bubble sort:
```
for (int i = 0; i < (n-1); i++) // Bubble sort to sort in ascending order.
{
   string store = "";
   for (int k = 0; k < (n-i-1); k++)
      if (a[k] > a[k+1]) // Switch neighbours if current is larger then next.
      {
         store = a[k];
         a[k] = a[k+1];
         a[k+1] = store;
      }
}
```

CStrings
Declaring strings as a sequence of characters.
"Hello" is a string literal, but can also be expressed by a sequence of 5 characters + the null terminator (\0) (6 elements for 5 letter word).

Declarations:
```
char myword[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char myword[] = "Hello";
```

Arrays cannot be assigned values, following is invalid:
myword[] = "Bye";

User input:
cin.getline(s,50)

strcat(dest, source)
Appends source to dest.

strcpy(dest, source)
Copies source to dest (replaces).

strcmp(str1, str2)
Compares str1 to str2.
If output is 0, str1 == str2.
If output is >0, str1 > str2.
If output is <0, str1 < str2.

strlen(str)
Outputs number of characters between beginning and null terminator.

An array of cstrings is declared by:
char wordArray[100][10]

Declaring array of cstrings as a parameter:
int function(char words[][MAXWORDLEN+1])

Examples:
strcpy:
```
char s1[20];
char s2[20] = "Another new string";
strcpy(s1, ""); // Contents of s1 changed to null string
strcpy(s1, "new string"); // Contents of s1 changed to "new string"
strcpy(s1, s2); // Contents of s1 changed to "Another new string"
```

strcat:
```
char s1[20] = "Hello";
char s2[20] = "friend";
strcat(s1, ", my ");     // s1 now contains "Hello, my "
strcat(s1, s2);          // s1 now contains "Hello, my friend"
```

Loop character by character using:
```
for (int k = 0; string[k] != '\0'; k++)
    …
```

Reference Functions:
```
void addOne(int &y)
{
   y = y + 1;
}
```

Only used to pass integer / double arguments, not needed for strings / arrays.

Swap:
```
void swap(int& x, int& y)
{
   int t = x;
   x = y;
   y = t;
}
```

Pointers
int: the integer type, stores integer.
int&: reference to integer (address).
int*: pointer to integer (location).

To create pointer that points to a:
int* ptr = &a

&a means generate pointer to a.
ptr is a pointer pointing to the location of a.

& is the reference function, and * is the deference function.
*ptr is the value of a, can be modified.

An array is just a pointer to the first element of the array.
a[2] is equivalent to *(a+2).

To pass a pointer by reference in a function do: int*& var

Reversing a string and returning pointer to dynamically allocated array:
```
char* reverseString(char* first, char* last)
{
   int len = (last - first); // length excluding null pointer
   char *newArray = new char[len+1];
   for (int k = 0; k < len; k++)
      newArray[len-1-k] = first[k];
   newArray[len] = '\0';
   return newArray;
}
```

Structures / Classes
(An object of some class type).(the name of a member of that type)
e.g. person.salary;

(A pointer to an object of some class type)->(the name of a **public** member of that type)
e.g. ptr->getSalary()

Defaults: Struct – public, class – private

Example of class:
```
class Soda{
    public:
        Soda();
        void setName(string name);
        string getName() const;
    private:
        string m_name;
};
```

If pointer to object is constant, cannot use non-const functions:
const Zurt* zp = &z

Dynamic Allocation
Non-dynamically allocated values are deleted when the program moves out of scope.

```
int * foo;
foo = new int [5];
```

To delete dynamically allocated data:
Value: delete pointer;
Array: delete[] pointer;
If a function wants to return the pointer to a new array, it must be dynamically allocated:
```
char *newArray = new char[LEN];
return newArray;
```

Example of class:
```
class Aquarium {
    public:
        Aquarium();
        bool addFish (int capacity);
        Goldfish *getFish (int n);
        void Oracle();
        ~Aquarium();
    private:
        Goldfish *m_fish[MAX_FISH];
    // pointers to fish
        int m_nFish; // number of fish
};

Aquarium:: Aquarium() {
    m_nFish = 0;
}

bool Aquarium:: addFish (int capacity) {
    m_fish[m_nFish] = new Goldfish (capacity)
    m_nFish++;
    return true;
}

Aquarium:: ~Aquarium(){
    for (int k = 0; k < m_nFish; k++)
        delete m_fish[k];
}

void Aquarium:: oracle() {
    for (int k = 0; k < m_nFish; k++)
    {
        m_fish[k] -> printMemory();
        m_fish[k] -> forget(); // use -> if different class
    }
}
```

Example of class:
```
class Toy
{
...
};

class Pet
{
    public:
        Pet (string nm, int initialHealth);
        ~Pet(); // destructor, called whenever a function goes away
        int health() const;
        void addToy();
        ...
    private:
        string m_name;
        int m_health;
        Toy* m_favoriteToy;
};

Pet :: Pet(string nm, int initialHealth) //constructor
{
    m_name = nm;
    m_health = initialHealth;
    m_favoriteToy = nullptr;
}

Pet :: ~ Pet ()
{
    delete m_favoriteToy;
}

void Pet :: addToy()
{
    delete m_favoriteToy;
    m_favoriteToy = new Toy;
}
```

Example of structure:

```
struct Employee
{
    string name; // name, salary, age are data-
members/fields/attributes of employee
    double salary;
    int age;
}; // need semicolon when using self declared function type

void printPaycheck(const Employee& e)
{
    cout << "Pay to the order of " << e.name << " the amt $" <<
e.salary/12 << endl;
}

void celebrateBirthday (Employee& e)
{
    e.age++;
}

void celebrateBirthday2 (Employee* ep)
{
    *ep.age++; // Error! Will not compile as . takes higher
precedence
    ep -> age++; // same as (*ep).age++;
}

double totalPayroll (const Employee emps[], int n)
{
    double total  = 0;
    for (int  k = 0; k < n; k++)
        total += emps[k].salary;
    return total;
}
```

Example of changing value with pointer:

```
void f(int *a)
{
    *a = 420;
    a++;
}

int main()
{
    int a[2] = {120,220};
    f(a);
    cout << a[0]; // Prints 420;
}
```

Example of class:

```
Arena::~Arena()
{
  cerr << "Entering Arena destructor" << endl;

  delete m_player;
  for (int k = 0; k < m_nRobots; k++)
    delete m_robots[k];

  cerr << "Leaving Arena destructor" << endl;
}

int Arena::rows() const
{
  return m_rows;
}

int Arena::cols() const
{
  return m_cols;
}

int Arena::nRobotsAt(int r, int c) const
{
  int count = 0;
  for (int k = 0; k < m_nRobots; k++)
    if (m_robots[k]->row() == r && m_robots[k]->col() == c)
      count++;

  return count;
}

bool Arena::determineNewPosition(int& r, int& c, int dir) const
{
  switch (dir)
  {
    case UP:
      if (r <= 1)
        return false;
      r--;
      break;
    case DOWN:
      if (r >= rows())
        return false;
      r++;
      break;
    case LEFT:
      if (c <= 1)
        return false;
      c--;
      break;
    case RIGHT:
      if (c >= cols())
        return false;
      c++;
      break;
    default:
      return false;
  }
  return true;
}
```