**#include <iostream>**

**#include <string>**

**#include <cctype>**

**#include <cmath>**

**#include <cstring>**

**using namespace std;**

**data types:**

int: whole numbers from -2bil to +2bil

double: numbers w/ <=16 digits after decimal point

char: one character (letter, number, special, etc)

bool: true (any non-0 number) or false (0)

string: a sequence of chars

**rounding doubles:**

cout.setf(ios::fixed);

cout.precision(*#*);

**inputs:**

cin >> *variable\_name*;

cin.ignore(10000, ‘\n’);

getline(cin, *string\_name*);

**casting & orders of magnitude:**

double (*operation*) int 🡪 double

int / other\_int 🡪 truncated answer

static\_cast<*desired\_type*>(*variable*);

(% = / = \*) > (+ = -)

! > && > ||

**while loops:**

int i = 0;

while (*condition*) {

*do something*;

i++;

}

**do while loops:**

int i = 0;

do {

*something*;

i++;

}

while (*condition*);

**for loops:**

for (int i = 0; *condition*; i++) {

*do something*;

}

**if statements:**

if (*boolean expression*) {

*do something*;

}

else

*do something else*;

**switch statement:**

int num = *some\_value*;

switch (num) {

case 1:

case 2:

*do\_a\_thing*;

break;

default:

*do\_another\_thing*;

break;

}

**i++ vs ++i:**

i++: returns i (1), then adds 1.

++i: adds 1, then returns i (2).

**strings & chars:**

string s = “*something*”;

string t = “*something\_else*”;

char c = ‘*some\_char*’;

useful functions:

*// returns the size of s*

s.size();

*// returns char at position k*

s.at(k);

*// returns char at position k*

s[k];

*// adds c to the end of s*

s += c;

*// adds t to the end of s*

s.append(t);

*// changes char to lowercase*

tolower(s.at(k));

*// changes char to uppercase*

toupper(s.at(k));

*// checks if char is a digit*

isdigit(s.at(k));

*// checks if char is alphabet*

isalpha(s.at(k));

*// checks if char is uppercase*

isupper(s.at(k));

*// checks if char is lowercase*

islower(s.at(k));

chars have int values:

‘0’ - ‘9’ are consecutive increasing vales

‘a’ - ‘z’ are increasing values

‘A’ - ‘Z’ are increasing values

string to int:

int i = s.at(k) - ‘0’; *// if s.at(k) is a digit*

substrings:

s.substr(*starting\_index*, *substring\_size*)

**functions:**

*return\_type* functionName(*parameters*) {

*do\_a\_thing*;

return *something*;

}

void return type: nothing is returned

**cmath:**

sqrt(double *number*);

pow(double *base*, double *exponent*);

sin(double *radians*);

cos(double *radians*);

**reference parameters:**

Allows a function to modify a variable.

*// example (varName is passed by reference):*

*return\_type* functionName(*data type***&**varName) {}

**arrays**:

*data\_type* array[*size*] =

{*element 1, element 2, …* };

^index 0^ ^index 1^

*// necessary to iterate through array*

int n = *size*;

*//* *returns element at index i*

array[i];

arrays as parameters:

void function(*data\_type* array[], int n) {

for (int i = 0; i < n; i++)

array[i] = *something*;

}

**2D arrays:**

*data\_type* 2dArray[*#rows*][*#cols*] =

{ {*row 1*}, {*row 2*}, … };

int row = *size*; *// keeps track of #rows*

int col = *size*; *// keeps track of #cols*

2dArray[r][c]; */\* returns element at row index r & column index c \*/*

**c strings:**

must be terminated with ‘\0’ or the “null” byte

char[6] = “hello”;

char[6]={‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};

cin.getline(*var*, *number\_of\_chars*);

*// length of str*

strlen(str)

*// note: all of the “n” functions look at up to count characters in src/dest*

*// compare strings*

strcmp(lhs, rhs) < or == or > 0

*// compare strings up to count chars*

strncmp(lhs, rhs, count)

*// copy src to dest*

strcpy(dest, src)

*// copy count chars of src into dest*

strncpy(dest, src, count)

*// append src to the end of dest*

strcat(dest, src)

*// append count chars of src to dest*

strncat(dest, src, count)

*// note: the following functions return pointers*

*// find first or last char in str*

strchr(str, char)

strrchr(str, char)

*// find first needle in haystack*

strstr(haystack, needle)

**Pointers:**

*// declaration:*

*data type*\* ptr = &*var\_name*;

*// follow a pointer to change the var:*

\*ptr = *something*;

*// pointer rules (a=array, p=pointer):*

1. \*&x == x
2. &a[i] +/- j == &a[i + j];
3. &a[i] < &a[j] <==> i < j
4. a == &a[0]
5. p[i] == \*(p + i)
6. &a[i] - &a[j] == i - j

*// iterating using pointers:  
// (a + n) is the same as &a[n]*

for (int\* p = a; p < a + n; p++) {

*do something;*

}

*// const ptr vs const var:*

*// can modify ptr but not var:*

const int\* ptr = &var;

*// can modify var but not ptr:*

int\* const ptr = &var;

**Classes & Structs:**

*// creation of class & members:*

class C

{

*// user accessible functions & variables:*

public:

*// constructor:*

C(int i, string b);

*// overloaded constructor allows user to set as many private member variables as they want:*

C(int a, string b, double d);

*// default constructor:*

C();

*// destructor (needed for dynamic allocation):*

~C();

*// accessors:*

*// should be declared const*

int getA() const;

string getB() const;

double getD() const;

*// mutators:*

void setA();

void setB();

void setD();

*// members only accessible within class:*

private:

int a;

string b;

double d;

};

*// constructor sets initial values:*

C::C(int i, string b) {

a = i;

this->b = b;

}

*// sample member function:*

int C::getA() const {  
 return a;

}

Struct creation is the same as class creation except structs have public variables first by default whereas classes have private ones.

This is irrelevant if you use public: & private:

*// creating instances of a class:*

C var1; *// default constructor*

C var2(5, “Hi”);

// DON’T DO: this declares a function

C var3(); // type: C (\*var3)(void)

*// new creates an instance of the class & returns a pointer to that instance:*

C\* ptrToC = new C(10, “Hello”);

**Dynamic Allocation:**

A useful way to store large classes in a large array storing pointers instead of instances of the class. This way you don’t have to initialize them all at once (more memory is free & faster program)

*// initialization:*

int n = 0;

int maxSize = *massive\_number*;

C\* array[maxSize];

*// adding new instances of the class:*

array[n] = new C;

n++;

*// deleting instances of the class (delete frees up memory used by that variable):*

delete array[i];

array[i] = array[n];

n--;

**Destructors:**

Gets rid of all the dynamically allocated instances of the class to prevent memory leaks.

*// creation of destructor:*

~C::C() {

for (int i = 0; i < n; i++)

delete array[i];

}

**Overloading:**

Basically, where you have multiple constructors or have multiple sets of parameters for a function. You’ve already seen overloading the constructor in the example of a class. Overloading functions works in a similar way.

*// overloading a function:*

int f() {  
 return 5;

}

int f(int a, int b) {

return a\*b;

}

THINGS TO ADD:

* the first part of for loop only runs once