**General Program**

*#include <iostream>*

*using namespace std;*

*int main (){*

*string characterName = “John”;*

*int characterAge = 35;*

*cout << “Hello ” << characterName << endl;*

*return 0*

*}*

**Datatypes**

A type of data/information we can use to use in our programs

*char grade = ‘A’ -* Single character

*string phrase = “Hello World” -* Instead of one character, it is a

string of characters

*int age = 50 –* Just a number

*float*

*double gpa = 2.3* -can store more decimal points than float

*bool isMale = true;*

**Strings**

*string phrase = “hello world”;*

*phrase.length();* 🡪 11

*phrase[0]* 🡪 h

*phrase[0] = ‘a’* 🡪 aello world

*phrase.find(“world”, 0); 🡪* look if hello exists starting at index 0,

outputs 6

*phrase.substr(6, 3);* 🡪 parameters starting index, length 🡪 wor

**Numbers [#include <cmath>]**

*10 % 3* 🡪 10 mod 3, gives remainder 🡪 1

*int wnum = 5;*

*double dnum = 5.5*

*wnum++;* 🡪 6 | wnum--; -> 5

*wnum += 20;* 🡪 25

*pow(2,5)* -> 2^5 = 32 | *sqrt(36)* 🡪 6

*round(4.1)* 🡪4*; round(4.6)* 🡪5

*ceil(4.1) ->* 5*; floor(4.8);* 🡪4

*fmax(3,10);* 🡪10 | *fmin(3,10)* 🡪 3

**User Input**

*int/float/char age;*

*cout << “Enter your age: ”;*

*cin >> age;*

*cout << “You are ” << age << “ years old”;*

string name;

getline(cin, name);

int num1, num2, num3;

string name; int age;

cin>>age;

cin.ignore();

getline(cin, name);

**Array**

*int luckyNums[20] = {4, 8, 15, 16, 23, 42};*

*luckyNums[10] = 100;*

*luckyNums[2]* 🡪 *15*

**Functions** **C++ Quick Review**

*void sayHi(){*

*cout << “Hello user”;*

*}*

**If Statements**

*bool isMale = true;*

*if(isMale){*

*cout << “You are male”;*

*}*

*else{*

*cout << “You are not male”;*

*}*

**Switch Statements**

*switch(dayNum){*

*case 0:*

*dayName = “Sunday”;*

*break;*

*case 1:*

*dayName = “Monday”;*

*break;*

*case 2:*

*dayName = “Tuesday”;*

*break*

*default:*

*dayName = “Invalid day Number”;*

*}*

**Loops**

while(){

}

do{

}while();

for(int i = 0; i < num; i++){

}

**2d Array & Nested Loops**

*int numberGrid[3][2] = {{1, 2},*

*{3, 4}.*

*{5,6}};*

*for(int i=0;i<3;i++){*

*for(int j = 0; j<2;j++){*

*cout << numberGrid[i][j];*

*}*

*}*

**Comment**

//Comment

/\* \*/

**Pointers**

*int age = 19;*

*int \*pAge = &age; 🡨* storing memory address of age

same for double, strings

*pAge* 🡪getting address, *\*pAge ->* references so gives 19

*&age* 🡪will give memory address

**Classes & Objects**

*class Book{*

*public:*

*string title;*

*string author;*

*int pages;*

*};*

*Book book1; book1.author = “JK Rowling”*

*book1.title = “Harry potter”; book.pages = 500;*

**Constructor Functions**

*class Book{*

*public:*

*string title;*

*string author;*

*int pages;*

*Book(){*

*title = “no title”;*

*author = “no author”;*

*pages = 0;*

*}*

*Book(string aTitle, string aAuthor, int aPages){*

*title = aTitle;*

*author = aAuthor;*

*pages = aPages;*

*}*

*};*

*Book book1(“Harry Potter”, “JK Rowling”, 500);*

*Book book3;*

**Object(Instance) Functions**

*class Student{*

*public:*

*string name;*

*string major;*

*double gpa;*

*Student(string aName, string aMajor, double aGpa) { name = aName;*

*major = aMajor;*

*gpa = aGpa;*

*}*

*bool hasHonors(){*

*if(gpa >= 3.5){*

*return true;*

*}*

*else{*

*return false;*

*}*

*}*

*};*

*Student student1(“Jim”, “Business”, 2.4);*

*Student student2(“Pam”, “Art”, 3.6);*

*student1.hasHonors();* 🡪 *false or 0*

*student2.hasHonors();* 🡪 *true or 1*

**Getters & Setters**

*class movie{*

*private:*

*string rating;*

*public:*

*string title;*

*string director;*

*Movie(string aTitle; string aDirector, string aRating){*

*title=aTitle;*

*director=aDirector;*

*setRating(aRating);*

*}*

*void setRating(aRating){*

*if(aRating == “G” || aRating == “PG” || aRating == “PG-13” || aRating == “R” ){*

*rating = aRating;*

*}*

*else{*

*rating = “NR”;*

*}*

*}*

*string getRating(){*

*return rating;*

*}*

*};*

*Movie avengers(“The Avengers”, Joss Whedon, “PG-13”);*

*avengers.setRating(“Dog”);*

*avengers.getRating();*

**Inheritance**

*class Chef{*

*public:*

*void makeChicken(){*

*cout << “Chef makes chicken” << endl;*

*}*

*void makeSalad(){*

*cout << “Chef makes salad” << endl;*

*void makeSpecialDish(){*

*cout << “Chef makes bbq ribs” << endl;*

*}*

*};*

*Class ItalianChef : public Chef{*

*public:*

*void MakePasta(){*

*cout << “Chef makes pasta” << endl;*

*}*

*void makeSpecialDish(){*

*//overriding the function*

*cout << “Chef makes spaghetti” << endl;*

*}*

*};*

*ItalianChef italianChef;*

*italianChef.makeChicken();*

*italianChef.makePasta();*

**Struct**

(used for smaller, classes used for larger generally)

Structs, members are by default public

Classes, members are by default private

**Array & Vectors**

Arrays were available in C (statically sized)

Vectors were introduced in C++ (dynamically sized)

**Array:**

*int values[4];*

*values[0] = 2;*

*int src[] = {2,4,6,8};*

*int dest[****10****];*

*copy(src, src+****10****,dest); -> dest = {2,4,6,8, \_, \_, \_, …}*

*int s\_size = sizeof(src) / sizeof(\*src);*

*int d\_size = sizeof(dest) / sizeof(\*dest);*

*int \*values = new int[4]; 🡨 manually have to disallocate from*

*memory later*

*values[0] = 2; values[1] = 4; values[2] = 6; values[4] = 8;*

*delete[values];* //Memory not freed immediately

**Vector:**

*vector<int> values;*

*values.push\_back(2); values.push\_back(4); values.push\_back(6); values.push\_back(8);*

*values -> {2,4, 6, 8};*

*int v\_size = values.size(); values.pop\_back(); remove last*

*values.empty(); 🡨 returns false values.resize(3);*

*values.clear(); 🡨 clears values values.back(); last element*

*values.at(i) 🡨 alternative way to access value*

*values = {2,4,6};*

*for(vector<int>::iterator it = values.begin(); it != values.end(); it++){*

*cout << \*it << “ ”;*

*}*

* *2, 4, 6*

*for(vector<int>::iterator it = values.begin(); it != values.end(); it++){*

*if(\*it == 4){*

*values.insert(it+1, 7);*

*break;*

*}*

*}*

* *2, 4, 7, 6*

*values = {2,4,6};*

*for(vector<int>::iterator it = values.begin(); it != values.end(); it++){*

*if(\*it == 4){*

*values.erase(it+1);*

*break;*

*}*

*}*

* *2, 4*

*values = {2,4,6};*

*values.pop\_back();*

* *2, 4*

*vector<int> values;*

*values.push\_back(2); values.push\_back(4); values.push\_back(6);*

*values.resize(10);*

*values[5] = 10;*

* *2, 4, 6, 0, 10, 0, 0, 0…*

**Global Variables**

*int x = 2; //global*

*other functions:*

*int foo(){*

*return x;*

*}*

*int foo2(){*

*int x = 25;*

*return x;*

*}*

*void main{*

*int x = 5;*

*x 🡪 5*

*::x 🡪 2*

*foo() 🡪 2*

*foo2() 🡪 25*

*}*

**Typedef Struct**

*typedef struct Person\_structs{*

*int age;*

*int weight;*

*char name[15];*

*char sex;*

*}Person;*

*Person Bob;*

*Bob.age = 70; Bob.weight = 175; strcpy(Bob.name, “Bob”);*

*Bob.sex = ‘M’;*

**1D Array (Returning through function)**

*//Note that you cannot get the size of a dynamically allocated array*

*void main(){*

*int\* primes = getArray(5);*

*}*

*int\* getArray(int size){*

*int\* array = (int\*)malloc((size)\*sizeof(int)); //Declare*

*array = (int\*)realloc(array,(size)\*sizeof(int)); //Resize*

*return array;*

*}*

**1D Array (Passing)**

*int main(){*

*int\* primes = {1,2,3,7};*

*change(primes);*

*}*

*void change(int\* primes){*

*primes[0] = #;*

*}*

**1D Array (Freeing)**

*void deleteArray(int\*\* primes){*

*free(\*primes);*

*}*

*Call in main: deleteArray(&primes);*

**2D Array (Returning)**

*int main(){*

*char\*\* board = makeBoard(4, 5);*

*for(int i = 0; i < 4; i++){*

*for(int j = 0; j < 5; j++){*

*printf("%c ", board[i][j]);*

*}*

*printf("\n");*

*}*

*}*

*char\*\* makeBoard(int rows, int cols){*

*char\*\* board = (char\*\*)malloc(rows\*sizeof(char\*));*

*for(int i = 0; i < rows; i++){*

*board[i] = (char\*)malloc(cols\*sizeof(char));*

*for(int j = 0; j < cols; j++){*

*board[i][j] = '#';*

*}*

*}*

*return board;*

*}*

**2d Array (Passing)**

*void changeBoard(char\*\* board){*

*board[1][2] = 'h';*

*}*

*changeBoard(board);*

**2d Array (Freeing)**

*void deleteBoard(char \*\*board, int m){*

*for (int i = 0; i < m; ++i) {*

*free(board[i]);*

*}*

*free(board);*

*}*

*// free dynamically allocated memory*

*for( int i = 0 ; i < \*row ; i++ ){*

*delete[] matrix[i]; // delete array within matrix*

*}*

*delete[] matrix; // delete actual matrix*

**3d Array**

*int\*\*\* array = (int\*\*\*)malloc(D1\*sizeof(int\*\*));*

*for(int i = 0; i < D1; i++){*

*array[i] = (int\*\*)malloc(D2\*sizeof(int\*));*

*for(j = 0; j < D2; j++){*

*array[i][j] = (int\*)malloc(D3\*sizeof(int));*

*}*

*}*

**3D Array (Freeing)**

*for (int i = 0; i < D1; i++){*

*for (int j = 0; j < D2; j++) {*

*free(array[i][j];*

*}*

*free(array[i]);*

*}*

*free(array);*

**Pointers**

*void main(){ void bar(a){*

*int b = 5; a = 80;*

*bar(b); }*

*//b is still 5*

*}*

*void main(){ void bar(\*a){*

*int b = 5; \*a = 80;*

*bar(&b); }*

*//b becomes 80*

*}*

*void main(){ void foo(int\* p){*

*int\* q = 95; p = NULL;*

*foo(q); }*

*//q is still 95*

*}*

*void main(){ void foo(int\*\* p){*

*int\* q = 95; \*p = NULL;*

*foo(&q); }*

*//q is now NULL*

*}*

**2-D Vector**

*#include <iostream>*

*#include <vector>*

*int main(){*

*std::vector<std::vector<int>> mat;*

*//Create a 4x4 vector filled with zeroes*

*mat.resize(4, std::vector<int>(4,0));*

*//Printing it*

*for ( std::size\_t i = 0; i < mat.size(); i++){*

*for (std::size\_t j = 0; j < mat[i].size(); j++){*

*std::cout << mat[i][j] << " ";*

*}*

*std::cout << std::endl;*

*}*

*}*

*#include <iostream>*

*#include <vector>*

*using namespace std;*

*int main () {*

*vector< vector<int>> stuff;*

*for(int i = 0; i < 3; i++){*

*vector<int> temp;*

*for (int j = 0; j < 3; j++){*

*temp.push\_back(i);*

*}*

*stuff.push\_back(temp);*

*}*

*for(int i = 0; i <stuff.size(); i++){*

*for(int j = 0; j<stuff[i].size(); j++){*

*cout << stuff[i][j];*

*}*

*cout << endl;*

*}*

*return 0;}*

Stack and heap are memory types in an application. Stack memory stores data types like int, double, boolean etc. While heap stores data types like string and objects.