**CS 37 Notes Part 2**

The next programs are used to illustrate the function processing that has been added to C++ from C.

**inline.cpp**

This is an inline function:

**inline** **double** cube(**const** **double** s) {return s \* s \* s;}

*When inline functions are called they literally replaces the call function with whatever was coded in the inline prototype*

cout << cube(side) <<endl; Original

cout << side\*side\*side <<endl; After replacement (what the computer will do when you use it). The program now runs as if no function was called at all.

* Because they replace the actual call function, inline functions are usually short and frequently used

The const in the parameter list means that the parameter s cannot change value in the function.

**inline** **double** cube(**const** **double** s) {return s \* s \* s;}

* A regular function requires the following overhead (time and memory needed to call and return from a regular function)

On a call, the return address of the next instruction is put on the stack along with the local variables of the calling function. A stack is a LIFO data structure which means last in first out.

On a return from a function, the local variables are pulled off the stack along with the return address. The computer must store the return address of the next instruction to do so the program knows where to go when returning from a function.

As you know, the main can call function a which can call function b which can call function c. When c is done, the program returns to b, then a, and then to the main. Note how the function returns match the stack processing of LIFO.

|  |
| --- |
| #include<iostream>  using **namespace** std;  *// inline functions reduce function call overhead*  *// compiler generates a copy of the function code in place where called*  *// used for small, frequently used functions*  *// can put const in parameter list to state parameter does not change*  **inline** **double** cube(**const** **double** s) {return s \* s \* s;}  **int** main()  {    cout <<"enter the side length of a cube: ";  **double** side;    cin >> side;    cout << "volume of cube with side " << side << " is " << cube(side) <<endl;    return 0;  }  */\**  *enter the side length of a cube: 3*  *volume of cube with side 3 is 27*  *\*/* |

**scoperes.cpp**

What is ::?

* A unary scope resolution operator
* Allows access to variables outside the current scope

We demonstrate that we can access variables outside the current scope in this program.

\*Note that global variables are strictly outlawed in this class

|  |
| --- |
| *//the unary scope resolution operator ::*  *//allows access to variables outside of the current scope*  *//will be used later with objects to call methods in other classes*  #include<iostream>  using **namespace** std;  **float** num = 2.345;  **int** main()  {  **int** num = 5;  *// can use single quotes for the characters*      cout <<"local value = " << num <<endl;      cout <<"global value = " <<::num << endl;      return 0;  }  */\**  *local value = 5*  *global value = 2.345*  *\*/* |

*Later in the class, the scope resolution operator will be used to access variables and methods outside the current object. For example, suppose we had two objects called Teacher and Student. For a teacher object, the teacher methods are called automatically and for a student object, the student methods are automatically. However, suppose that a teacher object wanted to call a student method calc. We would have to use the scope resolution operator to call the student method calc from the teacher object. That is, student::calc(). Same deal to call a teacher method from a student object, teacher:: calc().*

**overload.cpp**

What are overloaded functions?

* A collection of functions with the same name but different type (int void etc) or parameters
* *This is an example of polymorphism or “many shapes” as these “many shapes” are handled automatically by the compiler.*
* Any variation in the function call will lead the program to call only that particular function
  + *For example, suppose we had 4 functions called calc. The first calc has 1 parameter, the second calc has 2 parameters, the third calc has 3 parameters, and the last calc has 4 parameters. In the main, if the calc function is called with 3 parameters, then the calc function with 3 parameters is automatically called. If the calc function is called with 1 parameter, then the calc function with 1 parameter is automatically called.*

Based on the call in the main, the appropriate function is automatically called.

\*Overloaded functions are convenient to use as the programmer does not need to remember different function names. No need to ask yourself “What did I call the function with 3 parameters?”

|  |
| --- |
| *// overloaded functions*  *// a collection of functions with the same name but different signatures*  *// or parameter lists.  The appropriate function is automatically called.*  *// this is an example of polymorphism - "many shapes"*  #include <iostream>  using **namespace** std;  **int** twice(**int** x)   { return x + x;}  **double** twice(**double** x)  { return x + x;}  **int** twice (**int** x, **int** y)  { return x + y;}  **int** twice (**int** x, **int** y, **int** z)  { return x + y + z;}  **int** main()  {      cout << twice(5) << endl;      cout << twice(2.5)<< endl;      cout << twice (10, 20) << endl;      cout << twice (10, 20, 40) << endl;     return 0;  }    */\**  *The output is:*  *10*  *5*  *30*  *70*  *\*/* |

**maxtemp.cpp** *Applications of overloaded functions*

\*NOTE:

**template** <**class** T>

This line of code right here signifies that the parameters:

T max(T one, T two, T three)

Are going to be able to take in any sort type (like int or float) without overloading the code

**THEREFORE TYPE T REPLACES OVERLOADING**

We could have the following function to return the largest of three int parameters:

int max(int one, int two, int three)

{

int big = one;

if (two > big)

big = two;

if(three > big)

big = three;

return big;

}

We could then add another function to return the largest of three doubles:

double max(double one, double two, double three)

{

double big = one;

if (two > big)

big = two;

if(three > big)

big = three;

return big;

}

We could then add another function to return the largest of three chars:

char max(char one, char two, char three)

{

int big = one;

if (two > big)

big = two;

if(three > big)

big = three;

return big;

}

Repeating the function for each data type is not a good idea. It is a much better idea to use a template or function template. A function template is a function written only once that will work for multiple data types. The above functions could be written as one template:

template <class T> // or template<typename T>

T max(T one, T two, T three)

{

T big = one;

if (two > big)

big = two;

if(three > big)

big = three;

return big;

}

For each call, the datatype of T is determined by the call. For example, if three ints are passed, then T becomes int for this call only. If three floats are passed, then T become float for this call only. If the datatypes do not match in the call, an error is generated.

This program finds the largest of 3 values for ints, chars, and doubles.

For chars and strings, the later in the alphabet, the greater the char or string and lower case letters are larger than upper case letters.

eg ‘G’ > ‘C’ ‘h’ > ‘H’

* **The template T in this program cannot be used to compare strings as strings require the strcmp function to compare strings.**

|  |
| --- |
| #include<iostream>  using **namespace** std;  *//example of a function template*  *//works for multiple data types*  **template** <**class** T>  *// or can do: template <typename T>*  T max(T one, T two, T three)  {      T big = one;      if (two > big)          big = two;      if (three > big)          big = three;      return big;  }  **int** main()  {  **int** a,b,c;      cout << "enter three integers ";      cin >> a >>b >>c;      cout << "the biggest integer is " << max(a,b,c) << endl << endl;  **char** c1,c2,c3;      cout << "enter three characters ";      cin >> c1 >> c2 >> c3;      cout << "the biggest character is " << max(c1,c2,c3) << endl << endl;  **double** d1,d2,d3;      cout << "enter three doubles ";      cin >> d1 >> d2 >> d3;      cout << "the biggest double is " << max(d1,d2,d3) << endl << endl;        return 0;  }  */\**  *enter three integers 2 6 4*  *the biggest integer is 6*  *enter three characters f h U*  *the biggest character is h*  *enter three doubles 2.3456 1.23435 7.6543*  *the biggest double is 7.6543*  *\*/* |

**arrtemp.cpp**

This program proves that you can use different types of arrays in order to pass through the functions, namely exemplifying int float and char

When passing arrays with T classes, the parameter (in the prototype) can be either T\*a or T a[] (the second option is how they were passed in C)

THUS The heading line of the template is:

void printarray (T \*a, const int n)

*C++ contains the STL – Standard Template Library which are header files containing templates for many data structures and algorithms. Using STL takes advantage of using reusable code for software development. The term RAD means rapid application development and the STL and reusable code support RAD. To use the STL, you need to include the appropriate header file that contains the templates needed for the software that you are developing. STL templates include templates for vectors, lists, stacks, queues, priority queues, mathematical algorithms, and searching and sorting algorithms plus others.*

|  |
| --- |
| #include <iostream>  using **namespace** std;  *//function template to print an array*  *//works for multiple data types*  **template** <**class** T>  **void** printarray (T **\***a, **const** **int** n)  {      for (**int** i = 0; i < n; i++)          cout << a[i] << "  ";      cout << endl;  }  **int** main()  {  **const** **int** n1 = 5, n2 = 7, n3 = 6;  **int** a[n1] = {2, 4, 6, 8, 10};  **float** b[n2] = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7};  **char** c[n3] = "HELLO";      cout <<"the integer array" << endl;      printarray(a, n1);      cout <<"the float array" << endl;      printarray(b,n2);      cout <<"the string is" << endl;      printarray(c,n3);      return 0;  }  */\**  *the integer array*  *2  4  6  8  10*  *the float array*  *1.1  2.2  3.3  4.4  5.5  6.6  7.7*  *the string is*  *H  E  L  L  O*  *\*/* |
|  |

**defargs.cpp**

You can specify default values for the parameters in the parameterization of the function prototype in case no value is sent from the calling function for that parameter. This is a feature in C++ called default arguments.

THUS This program demonstrates several calls from the main to show how the parameters are passed.

**inline** **int** volume(**int** length = 1, **int** width = 1, **int** height = 1)

* This will work even if it’s not an inline function
* Each parameter that is not passed from the main will be assigned 1 in the function.

    cout << "the volume for no arguments is " << volume() << endl;

As no values are passed, length, width, and height are all set to 1 at the start of the function.

    cout << "the volume for length 10 is " << volume(10) << endl;

Will pass 10 to length, the other two as 1 as it’s defaulted

    cout << "the volume for length 10 and width 5 is " << volume(10,5);

Will pass 10 to length, and 5 to width, but only 1 to height because it’s defaulted

    cout << volume(10,5,2) << endl;

Will pass 10 to length, 5 to width, and 2 to height. *You usually use it like this before you’ve learned the lesson.*

*Note, as in the last couple examples, default arguments must be the rightmost arguments in the parameter list.* You cannot do the following: cout << volume(,2,); in an effort to set length and height to 1 and pass 2 to the second parameter width.

*We will use default arguments with constructors later which are used to create objects.*

|  |
| --- |
| *// default arguments program*  #include <iostream>  using **namespace** std;  *// program works whether the volume function is inline or not*  **inline** **int** volume(**int** length = 1, **int** width = 1, **int** height = 1)      {return length \* width \* height;}  **int** main ()  {      cout << "the volume for no arguments is " << volume() << endl;      cout << "the volume for length 10 is " << volume(10) << endl;      cout << "the volume for length 10 and width 5 is " << volume(10,5);      cout << endl << "the volume for all three parameters is ";      cout << volume(10,5,2) << endl;  }  *// default arguments must be the rightmost arguments in the*  *// parameter list*  *// inline functions advise compiler to generate a copy of the function's*  *// code.  Therefore, there is no function call.  Used for small functions.*  *// Reduces the function call overhead.*  */\**  *the volume for no arguments is 1*  *the volume for length 10 is 10*  *the volume for length 10 and width 5 is 50*  *the volume for all three parameters is 100*  *\*/* |

**funcptr1.cpp and funcptr2.cpp**

This is a function pointer, it points to the function the program will do whenever it’s set equal to something. Notice that the function pointer and the function needs to have the same format.

**void** (\*f)(**int**); // function pointer

**void** one(**int** a); // function to be called (in prototype format)

      f = one;  // function call

**Function pointers are used in C++ because it helps menu-handling (if statements/switch cases) much easier.**

        switch(choice)

        {

            case 1: f = one; break;

            case 2: f = two; break;

            case 3: f = three; break;

            default: f = four; break;

         }

*Function pointers will make the menu-handling more efficient as no if or switch statements will be needed.*

WHILE funcptr1.cpp USES A SWITCH STATEMENT, funcptr2.cpp IS THE RIGHT WAY TO DO THINGS.

COMPARE THE TWO:

Funcptr1.cpp

**void** (\*f)(**int**);

**int** choice;

    cout<< "enter a number from 1 to 4, 5 to quit ";

    cin>>choice;

    while (choice >=1 && choice < 5)

    {

        switch(choice)

        {

            case 1: f = one; break;

            case 2: f = two; break;

            case 3: f = three; break;

            default: f = four; break;

         }

         f (choice);

         cout<< "enter a number from 1 to 4, 5 to quit ";

         cin>>choice;

    }

Funcptr2.cpp

**void** (\*f[4])(**int**) = {one, two, three, four};

**int** choice;

    cout<< "enter a number from 1 to 4, 5 to quit ";

    cin>>choice;

    while (choice >=1 && choice < 5)

    {

        (\*f[choice - 1]) (choice);

         cout<< "enter a number from 1 to 4, 5 to quit ";

         cin>>choice;

    }

Notice that funcptr2.cpp actually has a function pointer array instead of a singular function pointer, which contains the explicit name of the functions themselves inside

Function pointers only work if all functions have the same signature (same return type and same parameter types). **Note that the four functions in this program are all void with one int parameter**. *This limitation may sound bad; but, most programs with data structures will use the same signature for each function which specifies the data structure and the size of the data structure.*

Note that we can substitute any letter with f just as long as it’s a variable. You just need to use the same letter for the intended purpose.

**void** (\*f)(**int**);

**void** (\*a)(**int**);

It’s impossible to declare a function pointer if all the functions had different signatures.

**void** (\*f[2])(**int**) = {two, three};

**void** two(**int** a);

**void** three(**float** a); < impossible, because it’s an int not a float

Funcptr1.cpp

* Four functions (one two three four)
* User enters 1, function one is called
* Etc.

Funcptr.cpp (Array of pointers)

Note that function 1 is \*f[0] therefore we have to subtract it by one for the right function to be called if the user enters the value **2** to run function two, we will subtract **2** – 1, and when the function is called, it is at \*f[1]. **THIS IS ALSO THE CALLING LINE**

(\*f[choice - 1]) (choice);

For example, if choice is 3, then the function at \*f[2] is called which is function three. The parameter choice which is 2 is passed to function three.

|  |
| --- |
| #include<iostream>  using **namespace** std;  *//when using function pointers, all functions must have the*  *//same signature*  **void** one(**int**);  **void** two(**int**);  **void** three(**int**);  **void** four(**int**);  **int** main()  {  **void** (\*f)(**int**);  **int** choice;      cout<< "enter a number from 1 to 4, 5 to quit ";      cin>>choice;      while (choice >=1 && choice < 5)      {          switch(choice)          {              case 1: f = one; break;              case 2: f = two; break;              case 3: f = three; break;              default: f = four; break;           }           f (choice);           cout<< "enter a number from 1 to 4, 5 to quit ";           cin>>choice;      }      cout<<"program is done" <<endl;      return 0;  }  **void** one(**int** a)  {    cout << "you entered  " << a <<" and you are in function one" <<endl;}  **void** two(**int** a)  {    cout << "you entered  " << a <<" and you are in function two" <<endl;}  **void** three(**int** a)  {    cout << "you entered  " << a <<" and you are in function three" <<endl;}  **void** four(**int** a)  {    cout << "you entered  " << a <<" and you are in function four" <<endl;}  */\**  *enter a number from 1 to 4, 5 to quit 1*  *you entered  1 and you are in function one*  *enter a number from 1 to 4, 5 to quit 2*  *you entered  2 and you are in function two*  *enter a number from 1 to 4, 5 to quit 3*  *you entered  3 and you are in function three*  *enter a number from 1 to 4, 5 to quit 4*  *you entered  4 and you are in function four*  *enter a number from 1 to 4, 5 to quit 5*  *program is done*  *\*/* |

|  |
| --- |
| #include<iostream>  using **namespace** std;  *//when using function pointers, all functions must have the*  *//same signature*  **void** one(**int**);  **void** two(**int**);  **void** three(**int**);  **void** four(**int**);  **int** main()  {  **void** (\*f[4])(**int**) = {one, two, three, four};  **int** choice;      cout<< "enter a number from 1 to 4, 5 to quit ";      cin>>choice;      while (choice >=1 && choice < 5)      {          (\*f[choice - 1]) (choice);           cout<< "enter a number from 1 to 4, 5 to quit ";           cin>>choice;      }      cout<<"program is done" <<endl;      return 0;  }  **void** one(**int** a)  {      cout << "you entered  " << a <<" and you are in function one" <<endl;  }  **void** two(**int** a)  {      cout << "you entered  " << a <<" and you are in function two" <<endl;  }  **void** three(**int** a)  {      cout << "you entered  " << a <<" and you are in function three" <<endl;  }  **void** four(**int** a)  {      cout << "you entered  " << a <<" and you are in function four" <<endl;  }  */\**  *enter a number from 1 to 4, 5 to quit 1*  *you entered  1 and you are in function one*  *enter a number from 1 to 4, 5 to quit 2*  *you entered  2 and you are in function two*  *enter a number from 1 to 4, 5 to quit 3*  *you entered  3 and you are in function three*  *enter a number from 1 to 4, 5 to quit 4*  *you entered  4 and you are in function four*  *enter a number from 1 to 4, 5 to quit 5*  *program is done*  *Press any key to continue*  *\*/* |

**Text Files**

The next three programs deal with text files.

* file1.cpp saves some ints to a text file
* file2.cpp saves some full names to a text file
* file3.cpp saves strings and numbers to a text file.

Text file commands are based on cin and cout and are less cumbersome commands than used in C (fscanf, fprint, fgets). *We will cover text files only as binary files in C++ are the same as C.*

Background information for text files:

*Recall 3 steps are needed to work with a text file.*

*1. open the file for input, output, or append.*

*2. work with the file.*

*3. make really sure to close the file.*

The 3 file modes are input, output, and append.

* Input (ios::in) reads/shows you the text file into the computer memory.
* Output (ios::out) saves/changes the the computer memory to the text file.
  + **If you open an existing file for output, the file is wiped out. (Be careful)**
* Append (ios::app) means to save the computer memory to the end of an existing text file. (to **add**)

|  |  |
| --- | --- |
| advantage | Disadvantage |
| You can view it in any text editor | You can only save one data element at a time and must format the text file as you save it (ie you can’t use enter you have to use whitespace) *The disadvantages of a text file is that you can only save one data element at a time and you must format the text file as you save it. That is, put in the whitespace.* |
|  | Example:  You wanna save 3 ints (x y z)  You have to save x first, followed by a space, then save y, followed by a space, and save z, typically followed by a new line (cumbersome) |

**To work with text files, you must include the header file fstream.h.**

#include<fstream>

*The text file commands are based on cin and cout just like in C, the text file commands were based on scanf and printf.*

|  |
| --- |
| #include<iostream>  #include<fstream>  *// must include when working with files*  using **namespace** std;  *// the file commands are similar to working with cin and cout*  **int** main()  {      ofstream outfile("c:\\nums.dat", ios::out);  *// can include a path such as "c:\\cppprogs\\nums.dat"*    **int** num;      cout <<"enter a number and ctrl z to quit ";      while( cin >> num)      {          outfile << num << endl;          cout << "next number?  ";      }      outfile.close();      cout <<endl  << endl;  *// time to retrieve and print the file*      ifstream infile("c:\\nums.dat", ios::in);      while ( infile >> num)          cout << num << "   ";      cout << endl;      infile.close();      return 0;  }  *// ios::app    writes to the end of the file (append)*  *// ios::in     opens the file for input*  *// ios::out    opens the file for output*  *//             file is overwritten if already exists*  */\**  *enter a number and ctrl z to quit 1*  *next number?  2*  *next number?  3*  *next number?  4*  *next number?  5*  *next number?  6*  *next number?  7*  *next number?  888*  *next number?  999*  *next number?  ^Z*  *1   2   3   4   5   6   7   888   999*  *Press any key to continue . . .*  *\*/* |

**file1.cpp**

What does this program do?

This program saves a collection of ints to a text file called nums.dat. One number is saved to each line of the text file. Every time a number is saved, a newline is also saved to the text file.

**Depending on the security software running on your computer, you may not be allowed to save a text file to the C: drive. If you run into this, change the drive letter to a flash drive and save your data on the flash drive. Do not shut off your security software to save text files!**

The main begins by opening the text file for output. ofstream is a data type for an output file stream outfile can be any variable name. **(do not use out because out is a reserved word, you could do ofstream o((“c:\\nums.dat”, ios::out); instead** Note that you must use \\ in the path name for the file as if only one \ is used, C++ will think we are using \n. Also, the folder needs to exist before you run your program. *You can use any extension you like for your text file such as .txt instead of .dat.*

    ofstream outfile("c:\\nums.dat", ios::out);

(Explaining the while statement) As long as the user is successfully entering a int value, the loop continues. *The user needs to press ctrl z to exit the loop when being prompted for the next number. (If you are using the Linux operating system, then use ctrl d instead of ctrl z.)*

**int** num;

    cout <<"enter a number and ctrl z to quit ";

    while( cin >> num)

    {

        outfile << num << endl;

        cout << "next number?  ";

    }

    outfile.close(); // don’t forget to close your programs!

This is how you save a number in the loop (endl is whitespace)*The basic idea is instead of using cout , use the variable for the output file stream.*

        outfile << num << endl;

The outfile.close(); statement closes the file.

    outfile.close();

**THIS LAST PART OF THE PROGRAM RETREIVES AND PRINTS THE TEXT FILE. The loop body prints the int just read in.**

    ifstream infile("c:\\nums.dat", ios::in);

    while ( infile >> num)

        cout << num << "   ";

    cout << endl;

    infile.close();

The file is opened

    ifstream infile("c:\\nums.dat", ios::in);

A while statement will keep processing as long as an int is being retrieved from the file

    while ( infile >> num)

        cout << num << "   ";

When the end of the file marker is reached, the while loop terminates because this part (inside the while line) cannot be done properly anymore because there aren’t any ints left

infile >> num

(Please run the three file programs and view the text files using any text editor such as notepad.)

|  |
| --- |
| #include<iostream>  #include<fstream>  *// must include when working with files*  using **namespace** std;  *// the file commands are similar to working with cin and cout*  **int** main()  {      ofstream outfile("c:\\nums.dat", ios::out);  *// can include a path such as "c:\\cppprogs\\nums.dat"*    **int** num;      cout <<"enter a number and ctrl z to quit ";      while( cin >> num)      {          outfile << num << endl;          cout << "next number?  ";      }      outfile.close();      cout <<endl  << endl;  *// time to retrieve and print the file*      ifstream infile("c:\\nums.dat", ios::in);      while ( infile >> num)          cout << num << "   ";      cout << endl;      infile.close();      return 0;  }  *// ios::app    writes to the end of the file (append)*  *// ios::in     opens the file for input*  *// ios::out    opens the file for output*  *//             file is overwritten if already exists*  */\**  *enter a number and ctrl z to quit 1*  *next number?  2*  *next number?  3*  *next number?  4*  *next number?  5*  *next number?  6*  *next number?  7*  *next number?  888*  *next number?  999*  *next number?  ^Z*  *1   2   3   4   5   6   7   888   999*  *Press any key to continue . . .*  *\*/* |

**file2.cpp**

This program is similar to file1.cpp but the data to save consists of full names (strings) instead of ints. The getline method also applies to input file stream variables. This program uses the cin.getline to allow the user to enter strings from the keyboard until ctrl z (or ctrl d) is pressed. The loop saves the name and a newline to the text file. The second part of the program retrieves and prints the text file. Note the statement:

while(infile.getline(name,30)) for the while loop. The loop will continue as long as a name was successfully read in from the text file.

A second way to retrieve and print the text file is:

ifstream infile(“c:\\people.dat”, ios::in);

while(!infile.eof())

{

infile.getline(name, 30);

cout << name << endl;

}

Infile.close();

The eof method returns true if we are at the end of the file. !eof means the loop continues processing while we are not at the end of the file. The file2.program technique is a bit shorter but some programmers do prefer using the eof method.

**file3.cpp**

This program saves an array of 4 structs to the text file students.dat. The struct is first declared and the main begins with a loop to allow the user to enter the struct data along with the calculation of the average data member of the struct. Note the ignore method as after entering in three scores, we need to ignore before allowing the user to enter the next name from the keyboard. We will need the ignore method for retrieving the text file also.

The next section saves the data to the text file students.dat. Note the whitespace that must be added to the text file. After this loop, the file is closed.

The next section retrieves the data from the text file. Note the ignore method as after entering in the three scores from the text file, we must use the ignore method to read in the next name from the text file. We need to ignore the newline character in the text file. If we do not use the ignore method, the newline character will be read in as the next name and the rest of the text file data will not be read in correctly.

The last section of the program prints out the array of structs after the file has been retrieved.

To change the data in a text file, you need to first retrieve the data into an appropriate data structure, modify the necessary values, and resave the data structure back to the text file.

If you want to add data to the end of an existing text file, just use ios::app to open the file.