**CS 37 Notes Part 1**

The first part of the course will cover the C commands that have been modified for C++. Many C commands continue to be the same for C++. For example, loops and arrays are the same between C and C++. And the last part of the course will cover the object-oriented commands of C++.

Studio: If you are using Studio, you will need to add the system(“PAUSE”) statement at the bottom of the code. Please feel free to use whatever programming environment you would like. At times, you will need to add the line #define \_CRT\_SECURE\_NO\_WARNINGS for Studio also.

**Outline:**

1. Cincout
   1. Introduces C programming
2. Coutprec and beyond
   1. C++ formatting requires iomanipulators functions with iomanip as the header file
   2. Background information – numbers can be printed in fixed notation such as 23.45 or 5431.443 or numbers can be printed in exponential notation such as 1.2e+003 or 2.46e-005.
   3. 2.463-005 means move the decimal point five places left so the number is .0000246.

**cincout1.cpp**

|  |  |  |
| --- | --- | --- |
| Notes | C programming | C++ |
|  | printf(“Enter three values “);  scanf(“%d%d%d”, &num1, &num2, &num3); | cout << “Enter three values “;  cin >> num1 >> num2 >> num3; |
| Iostream also includes system(“PAUSE”) function  use namespace std helps with not having to do std::cin whatever before the beginning of a statement | #include <stdio.h>  void main ()  {  } | #include <iostream> use namespace std;  int main ()  {  return 0;  } |
| To flush after entering a number | gets();  getchar();  fflush(); | Cin.getline  cin.ignore; |
|  | const SIZE 80; (outside function) | const int SIZE = 80; (inside main) |
| We can use both \n and endl in c++ | \n | endl |
|  | if (myage > frage)  printf(“You are older\n”);  else  if (myage > frage)  printf(“Your friend is older\n”);  else  printf(“the same age!\n”); | if (myage > frage)­­  cout << "You are older \n";  else  if (frage > myage)  cout << "Your friend is older" << endl;  else  cout << "the same age!" << endl; |
| The scope of a variable is the set of {} it is declared in. | can’t declare variables in loops | for (int i = 1; i <= 10; i++)  {  int x;  float y, z;  ...  }  // int x does not exist outside of the loop |
| C buffers (temporary holding areas are used when inputting a value. The value entered by the user was put into the standard input buffer, then scanf transfer the input into the variable. When printing a value, the value was put into the standard output buffer which was then sent to be printed to the screen. | Buffer | Stream  << is called the stream insertion operator and is used with cout statements. The values to be displayed are inserted into the standard output stream which are then displayed on the screen  >> is called the stream extraction operator and is used with cin statements, The values are extracted from the standard input stream and placed into the variables. |

|  |
| --- |
| #include<iostream> // do not include .h here  using namespace std; //if not used, must use std:: before all cout and cin statements  // eg std::cout << "Enter your age: ";  // eg std::cin >> myage;  // one line comment  // typical style for c++  // can still use /\* and \*/ for multiline comments  // #include<iostream.h> only works for older C++ compilers  int main() // must include int here as int is no longer the default  // type for the newer c++ compilers  {  cout << "Enter your age: ";  int myage;  cin >> myage;  cout << "Enter your friend's age: ";  int frage;  cin >> frage;  cout << "the two ages are: " << myage << "\t" <<frage <<endl << endl;  if (myage > frage)  cout << "You are older \n"; // can still use \n for newline or endl  else  if (frage > myage)  cout << "Your friend is older" << endl;  else  cout << "We are the same age" << endl;  return 0;  }  /\*  one possible output is:  Enter your age: 25  Enter your friend's age: 35  the two ages are: 25 35  Your friend is older  Press any key to continue . . .  In C++, a variable can be declared anywhere.  For example, in a function to make it local.  You can even declare a variable in a loop which means  it is local to that loop.  << is the stream insertion operator  >> is the stream extraction operator  In C++, in the cout statements, \n is the same as endl, that is  cout << "We are the same age \n";  is the same as  cout << "We are the same age" << endl;  \*/ |
|  |

**cincout2.cpp**

|  |  |  |
| --- | --- | --- |
| Notes | C | C++ |
| Reads up to whitespace | Scanf | Cin |
| Reads up to enter key (allows whitespace for strings) | Gets() gets\_s() | Cin.getline() |
| Buffer  Need to use otherwise return key in stream would read in as the next char/string, which happens when the program won’t let you type in a name | Getchar();  Fflush(stdin); | Cin.ignore() (The ignore statement in this program is cin.  (SIZE, ‘\n’); which ignores up to SIZE characters or until the newline character is reached, whichever happens first.) |

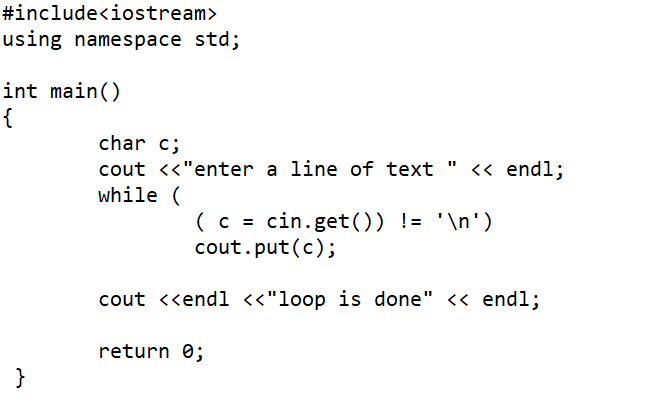
Thus we have:

1. Reads up to whitespace
2. Buffer
3. Read up to enter key
   1. Cin.getline(line, SIZE); reads up to the SIZE amount which contains whitespace

The cin.getline statement in the program is cin.getline(line, SIZE); which reads a string of SIZE characters that can contain whitespace.

C++ is an object oriented language. The standard input and output streams are objects. Functions that work on an object are called methods. getline and ignore are examples of methods.

**cincout3.cpp**



*While the input of cin.get(c) isn’t an enter, print out cout.put(c); with whitespace included*

*Ie. Read one character at a time and print it out UNTIL there is an enter key*

|  |  |  |
| --- | --- | --- |
| notes | c | C++ |
| Get a singular character | Getchar() | Cin.get(c) |
| Print a singular character | Print char? | Cout.put(c); |
|  |  |  |

**coutprec.cpp**

* This program assigns two float values and prints out both numbers using precision 2 through precision 7
* The precision number = how many digits the program will show (cout.precision(i) will print out the numbers with a total of i digits)
* If there are not enough digits specified to print the number, the number is printed in exponential/scientific notation, and are automatically rounded to the desired precision (For example, cout.precision(5); ie. 5 is the precision, and thenumber = 1234.567, cout << thenumber << endl will show 1234.6)
* *Note that the header file iomanip has been included in the program.*

For example:

cout.precision(5);

cout << x << endl;

cout << y << endl;

will print out both x and y with a total of 5 digits.

* *Once you set the precision, it stays until the program changes the precision.*

|  |
| --- |
| #include<iostream>  #include<iomanip> //must include when using io manipulators  using namespace std;  int main()  {  float num1 = 1234.567;  float num2 = 57.41111;  for (int i = 2; i <= 7; i++)  {  cout.precision(i); // specifies the number of digits to print  cout << "cout with precision " << i << " is " << num1 << endl;  cout << "cout with precision " << i << " is " << num2 << endl;  cout << endl;  }  return 0;  }  /\*  The output is:  cout with precision 2 is 1.2e+003  cout with precision 2 is 57  cout with precision 3 is 1.23e+003  cout with precision 3 is 57.4  cout with precision 4 is 1235  cout with precision 4 is 57.41  cout with precision 5 is 1234.6  cout with precision 5 is 57.411  cout with precision 6 is 1234.57  cout with precision 6 is 57.4111  cout with precision 7 is 1234.567  cout with precision 7 is 57.41111  Press any key to continue . . .  \*/ |

**coutpre2.cpp** (another way to print precision)

The previous program used cout.precision(i); In this program, this line has been replaced by the line cout << setprecision(i); Either can be used to specify the wanted precision.

|  |
| --- |
| #include<iostream>  #include<iomanip>  using namespace std;  int main()  {  float num1 = 1234.567;  float num2 = 57.41111;  for (int i = 2; i <= 7; i++)  {  cout << setprecision(i); // equivalent to cout.precision(i);  cout <<"cout with precision " << i << " is " << num1 << endl;  cout <<"cout with precision " << i << " is " << num2 << endl;  cout << endl;  }  return 0;  }  /\*  The output is:  cout with precision 2 is 1.2e+003  cout with precision 2 is 57  cout with precision 3 is 1.23e+003  cout with precision 3 is 57.4  cout with precision 4 is 1235  cout with precision 4 is 57.41  cout with precision 5 is 1234.6  cout with precision 5 is 57.411  cout with precision 6 is 1234.57  cout with precision 6 is 57.4111  cout with precision 7 is 1234.567  cout with precision 7 is 57.41111  Press any key to continue . . .  \*/ |

**format1.cpp**

This program demonstrates several examples of setw along with the left and right iomanipulators. The setw iomanipulator allows tables to be printed fairly easily.

This program introduces some other iomanipulators:

* setw(x) the next value printed will be printed using x columns
  + setw is the only iomanipulator that needs to be repeated everytime it is to be used. All the other iomanipulators stay in use until the programmer makes a change.
* left left-justify
* right right-justify

The default justification in C++ is right.

|  |
| --- |
| cout << setw(15) << last << setw(15) << first << endl; |

If you leave off the second setw(15), then first will be printed as is and not using 15 columns.

When printing, strings should be left-justified and numbers should be right-justified so that the decimal points line up.

eg cout << left << setw(15) << last << setw(10) << first;

cout << right << payamount << endl;

**format2.cpp**

This program prints out numbers in money format ie. two decimal digits.

The values are first printed unformatted and then printed a second time in money format.

To print in money format, 3 iomanipulators must be used

cout << setprecision(2) << showpoint << fixed;

* setprecision(2) will print two decimal digits only when used with showpoint and fixed.
* showpoint specifies that a decimal point should always be printed. (15 is unformatted 15.0 is formatted with showpoint)
  + The fixed iomanipulator says to always print in fixed notation even though the setprecision is only set at 2.

|  |
| --- |
| #include<iostream>  #include<iomanip>  using **namespace** std;  **int** main()  {  **float** num1 = 234.5673;  **float** num2 = 13.9275;  **float** num3 = 15.0;      cout <<"num1 unformatted is: " << num1 << endl;      cout <<"num2 unformatted is: " << num2 << endl;      cout <<"num3 unformatted is: " << num3 << endl << endl;      cout << setprecision(2) << showpoint << fixed;      cout << "num1 formatted as a money amount is:  $" << num1 << endl;      cout << "num1 formatted as a money amount is:  $" << num2 << endl;      cout << "num1 formatted as a money amount is:  $" << num3 << endl << endl;      return 0;  }  */\**  *num1 unformatted is: 234.567*  *num2 unformatted is: 13.9275*  *num3 unformatted is: 15*  *num1 formatted as a money amount is:  $234.57*  *num1 formatted as a money amount is:  $13.93*  *num1 formatted as a money amount is:  $15.00*  *Press any key to continue . . .*  *The setprecision command only needs to be specified once in this program.*  *The dollar amounts have been rounded.*  *The showpoint stream manipulator shows output with the decimal point.*  *The fixed stream manipulator shows output in fixed notation, that is NOT*  *in exponential notation.*  *\*/* |

**coutbase.cpp**

* this program contains iomanipulators to print numbers in diff bases ie.
  + base 10 (decimal), base 8 (octal), or base 16 (hexadecimal).
    - Hexadecimal and octal numbers (along with binary) are frequently used with assembly language programming so these iomanipulators can be very useful for assembly language programmers.
* The iomanipulators used in this program are the only values that work for setbase (8, 10, 16.)
  + hex or setbase(16) to print the numeric values in hexadecimal.
  + oct or setbase(8) to print the numeric values in octal.
  + dec or setbase(10) to print the numeric values in decimal.
* Once you specify setbase it will be printed that way until another iomanipulator is specified in the program
* As the letters A through F can be printed as hexadecimal numbers, the following two iomanipulators can be used:
  + uppercase to print these hexadecimal digits using capital letters.
  + nouppercase to print these hexadecimal digits using small letters.

|  |
| --- |
| #include<iostream>  #include<iomanip>  using **namespace** std;  **int** main()  {  **int** n = 28;      cout << n << " in hexadecimal is: " << hex << n << endl;      cout << uppercase << "hex in uppercase is " << n << endl;      cout << nouppercase << "hex back in lowercase is " << n << endl;      cout << n << " in octal is: " << oct << n << endl;      cout << n << " in decimal is: " << dec << n << endl << endl;      cout << setbase(16) << "hex is " << n << endl;      cout << uppercase << "hex in uppercase is " << n << endl;      cout << nouppercase << "hex back in lowercase is " << n << endl;  cout << setbase(8) << "octal is " << n << endl;      cout << setbase(10) << "decimal is " << n << endl;      return 0;  }  */\**  *possible values for setbase stream manipulator are*  *8, 10, 16 only.*  *uppercase and nouppercase stream manipulators only work for numbers*  *28 in hexadecimal is: 1c*  *hex in uppercase is 1C*  *hex back in lowercase is 1c*  *1c in octal is: 34*  *34 in decimal is: 28*  *hex is 1c*  *hex in uppercase is 1C*  *hex back in lowercase is 1c*  *octal is 34*  *decimal is 28*  *Press any key to continue . . .*  *\*/* |

**params.cpp**

* A parameter is a value passed to a function so that the function can complete its task, this program covers parameter passing in C++.
* There are two types of parameters:
  + Value parameter – one-way communication, done exactly the same as in C.
  + Reference parameter – two-way communication, the old C way still works plus there is a newer,
* C++ way to work with reference parameters that is less cumbersome than C.

No punctuation is needed for value parameters.

|  |  |  |
| --- | --- | --- |
|  | C | C++ |
|  | Use \*’s in the function and &’s in the call for simple variable types for the old C way for reference parameters. | Use &’s only in the parameter list of the function for the newer C++ way of working with reference parameters. |
|  | Strings and arrays can only be passed by reference in C/C++ using the same technique as for C. | |

You can either put the functions first followed by the main OR

Put the main followed by the functions. If you put the main first, you must prototype the functions. (Prototype – declaration of the functions that are coming later in the program.)

Functions in this program

1. Swaps numbers, swap by value
2. Swaps numbers with old C way, swap by reference
3. Swap numbers, C++, swap by reference
4. Review passing of strings with first name and last name (pass by reference only)

Structure:

1. The program begins with the four prototypes as the main comes first.
2. In the main, int variables a and b are declared, entered by the user, and printed.
3. Functions:
   1. Swapbyvalue
      1. Used to demonstrate value parameters
      2. The values of a and b are printed after returning to the main
      3. In the output, note that the values of a and b stay the same after we return from the main. The parameters a and b are passed to x and y in the function. ***X and y are swapped, but not returned to the main as value parameters use one-way communication.***
   2. Swapbyptr
      1. The old C way of handling reference parameters.
      2. Upon return to the main, a and b are printed and note that the values did indeed swap this time.
      3. The parameters a and b are passed to x and y in the function, x and y are swapped, and the values of x and y are passed back to a and b in the main as reference parameters use two-way communication.
   3. Swapbyref
      1. newer C++ way of handling reference parameters.
      2. Like swapbyptr, this function passes a and b to x and y where x and y are swapped. Lastly, x and y are passed back to a and b. Upon return to the main, a and b are printed again and note that the values have swapped back to their original values.
   4. String functions
      1. Strings and arrays are pass by reference only
      2. The main then declares two string variables and immediately prints out the two strings. Note that the output for the two strings is garbage as the two strings have not been given values yet. The getnames function is then called and the strings are printed upon return to the main.
      3. The getnames function passes two strings to allow the user to enter a first name and last name which are passed by automatically as strings and arrays can only be passed by reference in C/C++. Note that the \*’s only appear in the parameter list of the function. Also, note the ignore statement as we previously read in two numbers and are about to enter in two strings in this function.

|  |
| --- |
| #include<iostream>  using **namespace** std;  *// prototypes*  **void** swapbyvalue(**int**, **int**);  **void** swapbyptr(**int** **\***, **int** **\***);  **void** swapbyref(**int** **&**, **int** **&**);  **void** getnames(**char** **\***, **char** **\***);*// strings are passed by reference only*  **int** main()  {  **int** a, b;      cout <<"enter two numbers ";      cin >> a >> b;      cout <<"the two numbers are " <<a <<"\t" <<b << endl;      swapbyvalue(a,b);      cout <<"the two numbers after swapbyvalue are " <<a <<"\t" <<b << endl;      swapbyptr(&a, &b);*// \t is tab*      cout <<"the two numbers after swapbyptr are " <<a <<"\t" <<b << endl;      swapbyref(a,b);      cout <<"the two numbers after swapbyref are " <<a <<"\t" <<b << endl << endl;  **char** first[10], last[15];      cout << "before the names are entered the values are " << first << ' ' << last << endl;      getnames(first, last);      cout << "after the names are entered the values are " << first << ' ' << last << endl;      return 0;  }  **void** swapbyvalue(**int** x, **int** y)  {  **int** t = x;      x = y;      y = t;  }  **void** swapbyptr(**int** **\***x, **int** **\***y)  {  **int** t = \*x;      \*x = \*y;      \*y = t;  }  **void** swapbyref(**int** **&**x, **int** **&**y)  {  **int** t = x;      x = y;      y = t;  }  **void** getnames(**char** **\***f, **char** **\***l)  {      cin.ignore(10, '\n');*//needed when entering in string or character data*  *//after entering in numeric data*      cout << "enter a first name ";      cin.getline(f, 10);      cout << "enter a last name ";      cin.getline(l, 15);  }  */\**  *enter two numbers 5 10*  *the two numbers are 5   10*  *the two numbers after swapbyvalue are 5 10*  *the two numbers after swapbyptr are 10  5*  *the two numbers after swapbyref are 5   10*  *before the names are entered the values are ??D,?���x?@ �?@*  *enter a first name Joan Ann*  *enter a last name Jones*  *after the names are entered the values are Joan Ann Jones*  *Press any key to continue . . .*  *\*/* |
| enter two numbers 4 7  the two numbers are 4 7  the two numbers after swapbyvalue are 4 7  the two numbers after swapbyptr are 7 4  the two numbers after swapbyref are 4 7  before the names are entered the values are  enter a first name minh anh  enter a last name bui  after the names are entered the values are minh anh bui |