**C subset of C++**

**Raymond Klefstad, Ph.D.**

* an general purpose progrogramming language
* C++ can be viewed as having three levels:
  + Low-level C foundation for efficient system programming
  + Object-oriented level for defining classes, inheritance, templates, and more
  + High - level Abstract level for using template containers, algorithms, and iterators similar to scripting languages

**I/O**

* Simple Input and Output [LINK](http://personal.ee.surrey.ac.uk/Personal/R.Bowden/C/printf.html)  
  #include <stdio.h>  
  int main()  
  {  
   int i = 40;  
   double PI = 3.14159;  
   printf("Enter a number:\n");  
   scanf("%d", &i);  
   printf("I is %d\n", i);  
   printf("PI is %f\n", PI);  
   return 0;  
  }
* [Link to run code](http://pythontutor.com/visualize.html#code=%23include%20%3Cstdio.h%3E%0A%0Aint%20main%28%29%0A%7B%0A%20%20int%20i%20%3D%2040%3B%0A%20%20double%20PI%20%3D%203.14159%3B%0A%20%20printf%28%22Enter%20a%20number%3A%5Cn%22%29%3B%0A%20%20scanf%28%22%25d%22,%20%26i%29%3B%0A%20%20printf%28%22I%20is%20%)
* #include brings in declarations
* & in front of i gives its address so scanf can modify i

**Functions**

* similar to a mathematical function
* has 4 parts:
  + a name
  + a list of formal parameters, all passed by value (copy)
  + a return type
  + a compound statement to compute and return the result value
* EG   
  double square( double x )  
  {  
   return x \* x;  
  }

**The  *main* function**

* Every program must have one function named *main*
* when you run your program, main is called ([on writing main](http://www.delorie.com/djgpp/v2faq/faq22_25.html))
* [Link to run code](http://pythontutor.com/visualize.html#code=%23include%20%3Cstdio.h%3E%0A%0Adouble%20square%28%20double%20x%20%29%0A%7B%0A%20%20return%20x%20*%20x%3B%0A%7D%0A%0Aint%20main%28%29%20//%20%20int%20is%20the%20%20exit%20status%20for%20main%0A%7B%0A%20%20printf%28%22Square%20of%2012%20is%20%25g%5Cn%)  
   #include <stdio.h>

int main() //  *int* is the  *exit status* for main  
{  
 printf("Square of 12 is %g\n", square(12));  
 return 0; // 0 means program terminated  *ok*  
}

**Declaring Functions**

* called a  *function declaration* or  *function prototype*   
  double average( double x, double y );  
  double toFarenheight( double centegradeTemp );  
  double toCentegrade( double farenheightTemp );

**Defining Functions**

* called a  *function definition*   
  double average( double x, double y )  
  {  
   return ( x + y ) / 2.0;  
  }  
  double toFarenheight( double centegradeTemp )  
  {  
   return 9.0 \* centegradeTemp / 5.0 + 32.0;  
  }  
  double toCentegrade( double farenheightTemp )  
  {  
   return 5.0 \* ( farenheightTemp - 32.0 ) / 9.0;  
  }
* a function ***definition*** also acts as a function ***declaration***

**Using Functions**

* called a  *function call*   
  int main()  
  {  
   double x = 10.5;  
   double y = 32.6;  
   double z = average( x, y );  
   double centegradeTemp = 22.3;  
   double farenheightTemp = toFarenheight( centegradeTemp );  
   printf("%g\n", toFarenheight( centegradeTemp + 2.0 ));  
   printf("%g\n", average( toFarenheight( 29.7 ),  
   toFarenheight( centegradeTemp ) ) );  
   return 0;  
  }
* [Link to run code](http://pythontutor.com/visualize.html#code=%23include%20%3Cstdio.h%3E%0A%0Adouble%20average%28%20double%20x,%20double%20y%20%29%0A%7B%0A%20%20return%20%28%20x%20%2B%20y%20%29%20/%202.0%3B%0A%7D%0Adouble%20toFarenheight%28%20double%20centegradeTemp%20%29%0A%7B%0A%20%20return%209.0%20*%20centeg)
* generally, a function must be ***declared*** before it is ***called***

**Function Parameters**

* ***formals*** are those given in the definition
* x and y are the ***formal parameters*** for average   
  double average( double x, double y )  
  {  
   return ( x + y ) / 2.0;  
  }
* ***actuals*** are those supplied in a call
* 1.0 and 2.0\*x are the  ***actual parameters*** for this call to average below  
  double x = 5.0;  
  double z = average( 1.0, 2.0\*x );

**The Return Statement**

* causes a value to be returned to the function caller immediately
* type of return expression must match declared return type
* EG
* [Link to run code](http://pythontutor.com/visualize.html#code=%23include%20%3Cstdio.h%3E%0A%0Afloat%20square%28float%20x%29%0A%7B%0A%20%20float%20result%20%3D%20x%20*%20x%3B%0A%20%20printf%28%22Hello%20there!%5Cn%22%29%3B%0A%20%20return%20result%3B%0A%20%20printf%28%22Hello%20there%20again!%5Cn%22%29%3B%20//%20)  
  float square(float x)  
  {  
   float result = x \* x;  
   printf("Hello there!\n");  
   return result;  
   printf("Hello there again!\n"); // never printed  
  }  
  int main()  
  {  
   printf("%f\n", square( 2.0 ));  
   printf("%f\n", square( square( 2.0 ) ) );  
   return 0;  
  }

**Primitive Data Types**

* foundational types are  *built-in* or  *pre-defined*
* every data type has a size (in bytes) and a range of values
* includes integral, floating point, character and character string types

**The Integral Types**

* correspond to whole integers
* kinds of integral types:
  + char, 1 byte, -128 through 127
  + short, 2 bytes, -32768 through 32767
  + int, machine word size, now 32 bits
  + long, 4 bytes, -2147483648 through 2147483647
  + long long, 64 bit
  + also unsigned versions of all
* example literals   
  0  
  1  
  -1  
  -1234567  
  11 // decimal 11  
  011 // octal 9  
  0x11 // hex 17

**The Floating Point Types**

* corresponds to floating point real numbers
* three kinds of floating point types:
  + float (usually 4 bytes)
  + double (usually 8 bytes)
  + long double (usually 8 bytes)
* example literals   
  1.0  
  -3.000001e-10  
  30.01E40

**The Character Type**

* represents an ASCII character code
* requires one byte
* range of values   
  0 to 255
* example literals   
  '\0' null character 0  
  '\n' newline (or linefeed) 10  
  '\r' return 13  
  '\t' tab 9  
  ' ' space 32  
  '0' 48  
  'A' 65  
  'a' 97

**The Character String Type**

* represents a sequence of characters
* usually declared as a char \*   
  char \* s = "Hello";
* example literals   
  char \* t = "Hello world!";  
  printf("This is another character string.\n");  
  printf("%s\n", t); // prints: Hello World!
* we will learn all about character strings later

**Variables**

* variables must be declared before they are used
* variable declaration with initialization   
  int numberOfStudents = 30;  
  int automobileVelocity = 0;
* assignment operator changes the value in a variable   
  numberOfStudents = 0; // got rid of all the students  
  automobileVelocity += 20; // accelerated the auto

**Symbolic Constants**

* constants have a fixed value   
   #define PI 3.1415926536  
   #define NEWLINE '\n'
* it is good style to name literal constants   
  return 3.14159\*r\*r;  
  return PI\*r\*r;

**Simple Operators**

* Numeric Operators
  + +, -, \*, /, %, unary +,-
* Assignment Operators (modifies state of object)
  + =, +=, -=, \*=, /=, %=, ++, --   
    velocity = ( acceleration \* time \* time ) / 2.0;

**Using Operators Properly**

* precedence
* associativity
* parenthesis may over-ride
* memorize these operators, precedence, and associativity
* from highest to lowest
  + ++ -- (unary) + -
  + \* / %
  + + -
  + = += -= \*= /= %=

**Statements**

* Declaration Statements
  + introduces a new variable
  + variable is in scope to the end of enclosing block   
    int main()  
    {  
     double d = 2 \* PI;  
     printf("%f\n", d);  
     return 0;  
    }

**Expression Statements**

* any expression may be used as a statement
* the value is discarded   
  int main()  
  {  
   double PI = 3.14159;  
   double d = 2.0 \* PI;  
   printf("f\n", d);  
   while (d = d / 2.0);  
   square( 2.0 ); // be careful of this mistake  
   printf("%f\n", d);  
   return 0;  
  }

**Other Statements**

* if, switch, while, for, return, break
* you can declare local variables in loops
* [Link to run code](http://pythontutor.com/visualize.html#code=%23include%20%3Cstdio.h%3E%0A%0Aint%20main%28%29%0A%7B%0A%20%20for%20%28%20int%20i%3D0%3B%20i%3C10%3B%20%2B%2Bi%20%29%0A%20%20%20%20printf%28%22%25d%5Cn%22,%20i%29%3B%0A%20%20for%20%28%20int%20i%3D10%3B%20i%3E%3D0%3B%20--i%20%29%0A%20%20%20%20printf%)  
  int main()  
  {  
   for ( int i=0; i<10; ++i )  
   printf("%d\n", i);  
   for ( int i=10; i>=0; --i )  
   printf("%d\n", i);  
   return 0;  
  }

**Scoping Rules**

* a function's parameters are  *in scope* only within the function body
* we say they are  *local* to the function body
* any variable declared inside a function is also local to the function   
  int f( int a, int j ) // a and j are now in scope  
  {  
   int i = 10; // i is now in scope  
   {  
   int j = i; // new j is in scope and hides parameter j  
   int i = 30; // this new i is in scope and hides outer i  
   printf("%d\n", i \* j); // refers to inner i and j  
   } // inner j and i are now out of scope  
   printf("%d\n", i);  
   return a + j + i; // refers to the parameters and outer i  
  } // a, j, i are now out of scope  
  int main()  
  {  
   int a = 10; // a is in scope  
   int i = 20; // i is in scope  
   printf("%d\n", f( i, a )); // calls f with actual values 20 and 10  
   return 0;  
  } // a and i are now out of scope

**Output**

* output is done via printf function
* EG   
  int main()  
  {  
   printf("Hello");  
   printf("%d", 10 \* 10);  
   printf("%c", 'A');  
   printf("%f", 3.14159);  
   ...  
  }

**Input**

* input is done via scanf function
* uses address of parameter to modify its value
* it waits for a value to be entered (may require a return/enter)
* EG   
  int main()  
  {  
   int i;  
   double d;  
   char c;  
   scanf(“%d”, &i); // reads string of digits as integer  
   scanf(“%lf”, &d); // reads digits, decimal as a real number  
   scanf(“%c”, &c); // reads a single character  
   ...  
  }

**Files and Libraries**

* .h files include function declarations
* .c files include definitions of functions and variables
* .c files typically #include the .h files they use
* a .h file must be included where its declarations are used
* each .c file is compiled to object module
* object modules are linked together to form a program

**The if Statement**

* conditional execution of a statement
* [Link to visualize code](http://pythontutor.com/visualize.html#code=%23include%20%3Cstdio.h%3E%0A%0Aint%20main%28%29%0A%7B%0A%20%20int%20a%20%3D%201%3B%0A%20%20int%20b%20%3D%202%3B%0A%20%20if%20%28%20a%20%3C%20b%20%29%0A%20%20%20%20printf%28%22a%20%3C%20b%5Cn%22%29%3B%0A%20%20else%20if%20%28%20a%20%3E%20b%20%29%0A%20)  
  int main()  
  {  
   int a = 1;  
   int b = 2;  
   if ( a < b )  
   printf("a < b\n");  
   else if ( a > b )  
   printf("a > b\n");  
   else  
   printf("a == b\n");  
   if ( a > 0 )  
   printf("a is positive\n");  
  }

**Nesting if Statements**

* else's match nearest unmatched if
* indentation is not considered (be careful!)   
  int maxOfThree( int a, int b, int c )  
  {  
   if ( a < b )  
   if ( b < c )  
   return c;  
   else  
   return b;  
   else if ( a < c )  
   return c;  
   else  
   return a;  
  }

**if Statement Caveats**

* a syntax error that changes meaning of if statement   
  if ( e ); // extra semicoln means empty statements  
   printf("Hello\n"); // prints "Hello" even if e is false
* an awkward use of if statement   
  if ( e )  
   ; // nothing  
  else  
   printf("Hello\n");
* natural, but very harmful, mistake   
  int a = 0;  
  if ( a = 0 )   
   printf("Hello\n"); // never happens! Why?
* another awkward use of if statement   
  if ( a < b )  
   return true;  
  else  
   return false;
* better to say
  + return a < b;

**The switch Statement**

* for selecting among a set of integral values   
  int main()  
  {  
   int i = getIntegerFromUser();  
   printf("Some stuff here\n");  
   switch ( i )  
   {  
   case 1:  
   case 3:  
   case 5:  
   case 7:  
   case 9:  
   printf("%d is odd\n", i);  
   break;  
   case 0:  
   case 2:  
   case 4:  
   case 6:  
   case 8:  
   printf("%d is even\n", i);  
   break;  
   default:  
   printf("%d isn't in range 0 to 9\n", i);  
   break;  
   }  
   printf("Some more stuff here\n");  
  }

**Another switch Statement Example**

* break isn't required with return   
  bool isDigit( char c )  
  {  
   switch ( c )  
   {  
   case '0':  
   case '1':  
   case '2':  
   case '3':  
   case '4':  
   case '5':  
   case '6':  
   case '7':  
   case '8':  
   case '9':  
   return true;  
   default:  
   return false;  
   }  
  }

**switch Statement Caveats**

* forgetting the break!   
  int main()  
  {  
   int score = getScoreFromUser();  
   char grade = computeStudentsGrade( score );  
   switch ( grade )  
   {  
   case 'A':  
   printf("Excellent!\n");  
   case 'B':  
   printf("Good.\n");  
   case 'C':  
   printf("Fair - just passed.\n");  
   case 'D':  
   printf("Poor - See you next quarter.\n");  
   case 'F':  
   printf("Failed - off to McDonalds.\n");  
   default:  
   printf("Invalid Grade %d\n", grade);  
   }  
  }

**Another switch Statement Caveat**

* There are no ranges for integral values   
  bool isDigit(char c)  
  {  
   switch ( c )  
   {  
   case '0'-'9': // will subtract '9' from '0'  
   return true;  
   default:  
   return false;  
   }  
  }
* Must be listed separately   
  bool isDigit(char c)  
  {  
   switch ( c )  
   {  
   case '0':  
   case '1':  
   case '2':  
   /// do something here  
   default:  
   return false;  
   }  
  }

**The Concept of Iteration**

* also called `looping'
* allows repeating a similar action several times
* the  *break* statement will exit any loop
* the  *return* statement will also exit the loop

**The for Statement**

* the most common loop statement
* Natural for initializing, testing, then advancing
* abstract examples   
  for ( each student, s, in this class )  
   assignGradeTo( s );  
  for ( each day, d, of the quarter )  
   studyHardOnDay( d );  
  for (each station, s, on the radio tuner )  
  {  
   tuneTo( radio, s );  
   if ( youLikeTheSong( listen( radio ) )  
   break; /// terminates this for loop  
  }  
  for ( each integer, i, in the range 0 to 9 )  
   printf("%d\n", i);
* real examples   
  // print out numbers 0 through 9  
  for (int i = 0; i < 10; ++i)  
   printf("%d\n", i);  
  // read 10 integers from the input and print the sum  
  int main()  
  {  
   int valueRead = 0;  
   int sumTotal = 0;  
   for ( int i = 0; i < 10; i++ )  
   {  
   scanf("%d", &valueRead);  
   sumTotal += valueRead;  
   }  
   printf("The total is: %d\n", sumTotal);  
  }

**The while Statement**

* Natural for testing BEFORE doing an action that involves repetition
* EG   
  while ( isTooSour( coolade ) )  
   addATeaspoonOfSugar( coolade );  
  while ( waterIsTooCold( bathtub ) )  
   addAGallonOfHotWater( bathtub );  
  while ( ! understandTheHomeworkAssignment( student ) )  
  {  
   readTheHomeworkHandout( student );  
   askQuestions( student, TA );  
  }  
  while ( isStillAwake( student ) )  
   study( student );

**The do-while Statement**

* Natural for doing an action then testing for completion before repetition
* EG   
  do  
   turnIgnition( car );  
  while ( ! started( car ) );  
  do  
   pressANumber( phone );  
  while ( ! haveAConnection( phone ) );  
  do  
  {  
   readTheHomeworkHandout( student );  
   askSomeQuestions( student, TA );  
  } while ( ! understands( student, materialForWeek( w ) ) );  
  do  
   eat( person, pintOfIceCream );  
  while ( !sick( person ) );

**Nested loops**

* EG // print out a calendar   
  #define JAN 1  
  #define DEC 12  
  int days\_per\_month[]={0,31,29,31,30,31,30,31,31,30,31,30,31};  
  int main()  
  {  
   for ( int y = 2015; y <= 2020; y++ )  
   for ( int m = JAN; m <= DEC; m++ )  
   {  
   for ( int d = 1; d <= days\_per\_month[m]; d++ )  
   printf( "%d / %d / %d ", m, d, y );  
   printf("\n");  
   }  
  }

**Loop Caveats**

* loop control variable is only in scope over loop body   
  for (int i = 0; i < 10; i++ )  
   printf( "%d ", i );  
  printf( "%d\n", i ); /// i is no longer in scope
* some errors may cause an infinite loop   
  for (int i = 0; i < 10; i+1 ) /// i+1 is not advancing  
   printf( "%d ", i );  
  ...  
  int i; /// may forget to initialize  
  while ( i < 10 )  
   printf( "%d ", i ); /// not advancing!
* some errors may cause wrong values for i or incorrect number of loops   
  for (int i = 0; i <= 10; i++ ) /// wrong < operator  
   printf("%d ", i);  
  ...  
  for (int i = 1; i < 10; i++ ) /// wrong initial value  
   printf("%d ", i);

XXX**Simple Arrays**

* a fixed size, single-dimensional array of elements of the same type
* EG an array of three integers   
  int a[3] = {0, 1, 2};
* processed naturally with a for loop  
  for ( int i = 0; i < 3; i++ )  
   a[i] += 5; // add 5 to each element of array a
* can access individual elements directly   
  a[2] = a[0]; // assign value at a[0] into memory at a[2]
* can print them out   
  for ( int i = 0; i < 3; i++ )  
   printf( "%d\n", a[i] );
* you must keep track of the array size   
  #define A\_LENGTH 3  
  int a[A\_LENGTH];  
  void print()  
  {  
   for ( int i = 0; i < A\_LENGTH; i++ )  
   printf( "%d\n", a[i] );  
  }  
  **Extended Example**
* EG TimeSheet program   
  enum Day {SUN, MON, TUE, WED, THU, FRI, SAT, DAYS\_PER\_WEEK}  
  int hoursWorked[DAYS\_PER\_WEEK];  
  initTimeSheet()  
  {  
   for ( int i = 0; i < DAYS\_PER\_WEEK; i++ )  
   hoursWorked[i] = 0;  
  }  
  void print()  
  {  
   for ( Day i = SUN; i < DAYS\_PER\_WEEK; i++ )  
   printf("On day %d worked %d hours\n", i, hoursWorked[i]);  
  }  
  void recordHours(int i, int hours)  
  {  
   assert( i >= 0 && i < DAYS\_PER\_WEEK );  
   assert( hours >= 0 );  
   hoursWorked[i] = hours;  
  }  
  int totalHours()  
  {  
   int totalHours = 0;  
   for ( int i = 0; i < DAYS\_PER\_WEEK; i++ )  
   totalHours += hoursWorked[i];  
   assert( totalHours >= 0 );  
   return totalHours;  
  }  
  int main()  
  {  
   initTimeSheet();  
   recordHours(MON, 8);  
   recordHours(TUE, 9);  
   recordHours(WED, 6);  
   recordHours(THU, 9);  
   recordHours(FRI, 4);  
   print();  
   printf("Worked %d total hours this week\n", totalHours());  
   return 0;  
  }

**Character Arrays (AKA character strings)**

* character strings are arrays of characters terminated by '\0'
* tricky thing is you need an extra element for the terminator
* Three examples (of the string containing "abc")   
  char s1[4] = {'a','b','c','\0'};

Physical length = 4,

Logical lengt = 3 (not ‘\0’ inside)  
char s2[4] = "abc";  
char s3[] = "abc";

Associate Memory

**Searching a character string for a specified character**

* to find the index of an element containing a specified value   
  int findIndexOfChar(char c, char s[])  
  {  
   for ( int i = 0; s[i] != '\0'; i++ )  
   if ( s[i] == c )  
   return i;  
   return -1;  
  }
* example of use   
  int main()  
  {  
   char s[] = "Hello There";  
   int posT = findIndexOfChar( 'T', s );  
   if ( posT == -1 )  
   printf("T is not in %s\n", s);  
   else  
   printf("T is at position %d\n", posT);  
   s[posT] = 'W';  
   printf("%s\n", s); // prints: Hello Where  
  }

**String Library Functions**

* important low-level C-string utilities   
   #include <string.h>  
  int strlen(char s[]);  
  int strcmp(char s1[], char s2[]);  
  char [] strdup(char s[]);  
  char [] strcpy(char s1[], char s2[]);  
  char [] strcat (char s1[], char s2[]);

**Pointers and Addresses**

* pointers contain the address of some object
* allow access to that object
* can have multiple pointers to an object   
  int i = 10;
* a pointer contains the address of some object   
  int \* p = & i; // & gives address of object i  
  int \* q = & i;  
  i = 50; // changes i directly  
  \*p = 60; // changes i indirectly  
  \*q = 70; // changes i indirectly
* pointers can be changed to point to other objects   
  int k = 20;  
  p = & k; // p now points to k  
  \*p = 60; // changes k indirectly
* zero is used for null address (means pointing to nothing)   
  p = 0; // or NULL
* indirection through null address is an error   
  \*p = 100; // should cause run-time error
* a pointer parameter can be used to “pass by reference”  
  void get\_size(int \*ip, int \*jp)

{

printf(“Enter two integers:\n:”)

scanf(“%d %d”, ip, jp);

}

int main()

{

int i, j;

get\_size(&i, &j);

printf(“i = %d j = %d\n”, i, j);

return 0;

}

**Arrays and Pointers**

* In C, arrays are implemented as pointers to first element   
  int a[4];  
  int b[2];  
  int \* p = a; same as &a[0]  
  \*p = 10;  
  p[0] = 20;  
  p = b;  
  \*p = 30;  
  p[0] = 40;
* character strings are arrays of characters   
  char s1[] = "Hello";  
  char \* s2 = "Hello"; // not the same thing as s1  
  char \* s3 = s1;  
  s3[0] = 'M'; // changes s1 to "Mello"
* pointer arithmetic   
  p[1] = 70;  
  \*(p+1) = 70; // does the same thing  
    
  for ( char \*p = s1; \*p; ++p ) // prints Mello  
   printf("%c", \*p);

**Limitations of Fixed-Size Arrays**

* size must be known at compile-time
* once it is allocated, array cannot grow
* size may depend on use
* dynamic allocation of an array gives us flexibility (use pointers with malloc() and free() )

**Arguments to main**

int main(int argc, char \*argv[]) {

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

printf("Arg %d is \"%s\"\n", i, argv[i]);

}

**$ myProg foo bar baz**

**Arg 0 is "myProg"**

**Arg 1 is "foo"**

**Arg 2 is "bar"**

**Arg 3 is "baz"**

**Character File I/O**

* **Example: copy input file to output file and count characters**

#include <stdio.h>

#include <stdlib.h> /\* for exit() \*/

int char\_freq[255] = {0};

int main(int argc, char \*argv[]) {

FILE \*ifp, \*ofp;

char \*inputFilename = argv[1];

char \*outputFilename = argv[2];

int inch; /\* why an int and not a char?? \*/

ifp = fopen(inputFilename, "r");

if ( ifp == NULL ) {

fprintf(stderr, "Can't open %s\n", inputFilename);

exit(1);

}

ofp = fopen(outputFilename, "w");

if ( ofp == NULL ) {

fprintf(stderr, "Can't open %s!\n", outputFilename);

fclose(ifd);

exit(1);  
 }

while ( (inch = fgetc(ifp) ) != EOF )

{  
 ++char\_freq[inch];

fputc(ofp, inch);

}  
 print\_char\_freq();

fclose(ifd);

fclose(ofd);  
}

**Formatted File I/O**

* **Example: adds 10 points to every score in the specified input file**

#include <stdio.h>

int main(int argc, char \*argv[]) {

FILE \*ifp, \*ofp;

char \*inputFilename = argv[1];

char \*outputFilename = argv[2];

char username[100];

int score;

ifp = fopen(inputFilename, "r");

if ( ifp == NULL ) {

fprintf(stderr, "Can't open %s\n", inputFilename);

exit(1);

}

ofp = fopen(outputFilename, "w");

if ( ofp == NULL ) {

fprintf(stderr, "Can't open %s!\n", outputFilename);

fclose(ifd);

exit(1);  
 }

while ( fscanf(ifp, "%s %d", username, &score) != EOF )  
 fprintf(ofp, "%s %d\n", username, score + 10);

fclose(ifd);

fclose(ofd);  
}

**/\* Sample input \*/**

klefstad 90

smith 80

jones 70

anderson 50

**Structs**

* a heterogenous group of data
* e.g.,

struct DataMix {

char c;  
 int i;  
 long l;  
 double d;  
 void \*p;  
 };

struct DataMix dm;

dm.c = ‘A’;

dm.i = 1024;

dm.l = 34567;

dm.p = 0;

printf("C = %c I = %d L = %d P = %d\n", dm.c, dm.i, dm.l, dm.p);

* **typedef**

typedef char Buffer[50];

struct Name {

Buffer first;

Buffer last;

};

struct Date {

int month, day, year;

};

struct Person {

struct Name name;

struct Date birthdate;

};

* **Padding may be inserted to meet bus alignment restrictions**

struct DataMix {

char c;  
 int i;  
 long l;  
 double d; // aligned on multiple of 8???  
 void \*p;  
 };

struct DataMix dm;

printf("&C = %o &I = %o &L = %o &P = %o\n",

&dm.c, &dm.i, &dm.l, &dm.p);

printf("Each DataMix is %d bytes in size\n", sizeof dm);

**Arrays of Structs**

* **Very common to use array of structures, like a list**
* **e.g.,**

#define CLASS\_MAX\_SIZE 450

struct Person roster[+];

int number\_in\_class = 0;

**How would you insert?  
How would you find?  
How would you remove?**

**Unions**

* **allows any one of the fields to be alive**
* **size is max size of each alternative field**

struct taggedunion {  
 enum {UNKNOWN, CHAR, SHORT, INT, LONG, DOUBLE, POINTER} code;  
 union {

char c;

short s;  
 int i;  
 long l;  
 double d;  
 void \*p;  
 } un;  
};

struct taggedunion tu;

printf("Each TaggedUnion is %d bytes in size\n", sizeof tu);

**Function Pointers**

int f(int a, float b) /\* function returing int \*/

{

return a + b;

}

int g(int a, float b)

{

return a \* b;

}

int (\*fp)(int a, float b); /\* pointer to function like f \*/

* **May be assigned a function**

fp = f;

* **May be called either way below**

(\*fp)(2,5);  
or  
 fp(2,5);

printf("result = %d\n", **fp(2,5)**); /\* fp’s value is f, calls f \*/

* **Can change its value**

fp = g;

printf("result = %d", fp(2,5)); /\* now calls g not f \*/

* **Really useful for function parameters**

int squares = {0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100};

int sum(int a[], int size, int (\*fp)(int a))

{

int total = 0;

for (int i=0; i<size[ ++i)

total += fp(a[i]);

return total;

}

int div2(int i)

{

return i / 2;

}

int main()

{

int sumSquareDiv2 = sum(squares, 11, div2);

printf("%d\n", sumSquareDiv2);

}

XXX

**Defining Objects Outside the Class**

* Each instance has its own data members
* public members may be accessed using the dot operator
* EG Complex.cpp   
   #include "Complex.h"  
  int main()  
  {  
   Complex c1( 1.5, 5.3); /// c1 is born  
   Complex c2( 2.5, 2.7 ); /// c2 is born  
   c1.print( cout );  
   c2.print( cout );  
   {  
   Complex result; /// What happens here? **Note no parens!!!**  
   result.print( cout );  
   result = c1.add( c2 ); /// and here??  
   result.print( cout );  
   } /// and here???  
   c1 = Complex( 2.0, 3.0 ); /// a literal Complex number  
   c1.print( cout );  
   return 0;  
  } /// and here????

**Boolean expressions**

* they return 0 (false) or 1 (true)
* in general, non-zero is also considered true
* boolean expressions consist of
  + constants or variables
  + unary or binary expressions involving boolean expressions
  + EG   
    !a && b < c || d == 0

**Primitive type bool**

* predefined type "bool" is short for "boolean"
* has values false and true
* useful for conditions   
  bool isEqual(int x, int y)  
  {  
   return x == y;  
  }  
  int main()  
  {  
   bool b = true;  
   b = isEqual(3, 4);  
   b = false;  
   return 0;  
  }

**Equality Operators**

* a == b
  + returns true iff a and b contain the same value
* a != b
  + returns true iff a and b contain different values
* no default == or != for classes

**Relational Operators**

* a < b
* a > b
* a <= b
* a >= b
* no default relational operators for classes

**Logical Operators (short circuited)**

* a && b '  
  bool cond = divisor > 0 && numerator / divisor > 0.1;
* a || b   
  bool notADigit = c < '0' || c > '9';
* !a   
  bool isAChild = age < 18;  
  bool isAdult = !isAChild;

**The if Statement**

* conditional execution of a statement   
  int main()  
  {  
   int a = 1;  
   int b = 2;  
   if ( a < b )  
   cout << "a < b\n";  
   else if ( a > b )  
   cout << "a > b\n";  
   else  
   cout << "a == b\n";  
   if ( a > 0 )  
   cout << "a is positive\n";  
  }

**The switch Statement**

* for selecting among a set of integral values   
  int main()  
  {  
   int i = getIntegerFromUser();  
   cout << "Some stuff here\n";  
   switch ( i )  
   {  
   case 1:  
   case 3:  
   case 5:  
   case 7:  
   case 9:  
   cout << i << " is odd\n";  
   break;  
   case 0:  
   case 2:  
   case 4:  
   case 6:  
   case 8:  
   cout << i << " is even\n";  
   break;  
   default:  
   cout << i << " isn't in range 0 to 9\n";  
   break;  
   }  
   cout << "Some more stuff here\n";  
  }

**Another switch Statement Example**

* break isn't required with return   
  bool isDigit( char c )  
  {  
   switch ( c )  
   {  
   case '0':  
   case '1':  
   case '2':  
   case '3':  
   case '4':  
   case '5':  
   case '6':  
   case '7':  
   case '8':  
   case '9':  
   return true;  
   default:  
   return false;  
   }  
  }

**switch Statement Caveats**

* forgetting the break!   
  int main()  
  {  
   int score = getScoreFromUser();  
   char grade = computeStudentsGrade( score );  
   switch ( grade )  
   {  
   case 'A':  
   cout << "Excellent!\n";  
   case 'B':  
   cout << "Good.\n";  
   case 'C':  
   cout << "Fair - just passed.\n";  
   case 'D':  
   cout << "Poor - See you next quarter.\n";  
   case 'F':  
   cout << "Failed - off to OCC.\n";  
   default:  
   cout "Invalid Grade " << grade << endl;  
   }  
  }

**Simple Menu User Interface**

* a simple user interface will do the following:
  + present a menu
  + read a character command from the user
  + evaluate the command appropriately

**Menu Presentation**

* EG   
  void presentMenu()  
  {  
   cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\n"  
   << " \* PIGGY BANK MENU \*\n"  
   << " \* \*\n"  
   << " \* OPTION ENTER \*\n"  
   << " \* \*\n"  
   << " \* Show Balance (in $) B or b \*\n"  
   << " \* Show Coins in the Bank C or c \*\n"  
   << " \* Deposit Coins D or d \*\n"  
   << " \* Get Coins for Purchase P or p \*\n"  
   << " \* \*\n"  
   << " \* Quit Q or q \*\n"  
   << " \* \*\n"  
   << " \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\n";  
  }

**Reading the Command Character**

* the prompt parameter allows us to specify a message for the user
* EG   
  char getChoice( const char \* prompt )  
  {  
   char ch;  
   cout << prompt << " (followed by enter): ";  
   cin >> ch;  
   return ch;  
  }

**Evaluation of the command**

* EG   
  void evaluateCommand( Coins & piggyBank, char choice )  
  {  
   switch ( choice )  
   {  
   case 'B': case 'b':  
   cout << "Balance is $ " << piggyBank.total() << endl;  
   break;  
   case 'C': case 'c':  
   cout << piggyBank << endl;  
   break;  
   case 'D': case 'd':  
   cout << "How many quarters? ";  
   ...  
   break;  
   case 'P': case 'p':  
   ...  
   case 'Q': case 'q':  
   cout << "Done with Piggy Bank.\n\n";  
   exit(0); /// causes the program to terminate  
   default:  
   cout << "Invalid command " << choice << endl;  
   break;  
   }  
  }

**Putting it all together**

* EG   
   #include <iostream>  
   #include "Coins.h"  
  int main()  
  {  
   Coins piggyBank(0,0,0,0);  
   while ( true )  
   {  
   presentMenu();  
   char command = getChoice("Enter a command character");  
   evaluateCommand( piggyBank, command );  
   }  
  }

**The Concept of Iteration**

* also called `looping'
* allows repeating a similar action several times
* the  *break* statement will exit any loop
* the  *return* statement will also exit the loop

**The for Statement**

* the most common loop statement
* Natural for initializing, testing, then advancing
* **abstract examples**   
  for ( each student, s, in this class )  
   assignGradeTo( s );  
  for ( each day, d, of the quarter )  
   studyHardOnDay( d );  
  for (each station, s, on the radio tuner )  
  {  
   radio.tuneTo( s );  
   if ( youLikeTheSong( radio.listen() )  
   break; /// terminates this for loop  
  }  
  for ( each integer, i, in the range 0 to 9 )  
   cout << i << endl;
* **real examples**   
  // print out numbers 0 through 9  
  for ( int i = 0; i < 10; ++i )  
   cout << i << endl;  
  // read 10 integers from the input and print the sum  
  int main()  
  {  
   int valueRead = 0;  
   int sumTotal = 0;  
   for ( int i = 0; i < 10; i++ )  
   {  
   cin >> valueRead;  
   sumTotal += valueRead;  
   }  
   cout << "The total is: " << sumTotal << endl;  
  }

**The while Statement**

* Natural for testing BEFORE doing an action that involves repetition
* EG   
  while ( coolade.isTooSour() )  
   coolade.addATeaspoonOfSugar();  
  while ( bathtub.waterIsTooCold() )  
   bathtub.addAGallonOfHotWater();  
  while ( ! student.understandTheHomeworkAssignment() )  
  {  
   student.readTheHomeworkHandout();  
   student.askQuestions( TA );  
  }  
  while ( student.isStillAwake() )  
   student.study();

**The do-while Statement**

* Natural for doing an action then testing for completion before repetition
* EG   
  do  
   car.turnIgnition();  
  while (! car.started() );  
  do  
   phone.pressANumber();  
  while (! phone.haveAConnection() );  
  do  
  {  
   student.readTheHomeworkHandout();  
   student.askSomeQuestions(TA);  
  } while ( !student.understands( materialForWeek( w ) ) );  
  do  
   person.eat( pintOfIceCream );  
  while ( !person.sick() );

**Loop Caveats**

* loop control variable is only in scope over loop body   
  for (int i = 0; i < 10; i++ )  
   cout << i;  
  cout << i; /// i is no longer in scope
* some errors may cause an infinite loop   
  for (int i = 0; i < 10; i+1 ) /// i+1 is not advancing  
   cout << i;  
  ...  
  int i; /// may forget to initialize  
  while ( i < 10 )  
   cout << i; /// not advancing!
* some errors may cause wrong values for i or incorrect number of loops   
  for (int i = 0; i <= 10; i++ ) /// wrong < operator  
   cout << i;  
  ...  
  for (int i = 1; i < 10; i++ ) /// wrong initial value  
   cout << i;

**Simple Arrays**

* a fixed size, single-dimensional array of elements of the same type
* EG an array of three integers   
  int a[3] = {0, 1, 2};
* processed naturally with a for loop   
  for ( int i = 0; i < 3; i++ )  
   a[i] += 5; // add 5 to each element of array a
* can access individual elements directly   
  a[2] = a[0]; // assign value at a[0] into memory at a[2]
* can print them out   
  for ( int i = 0; i < 3; ++i )  
   cout << a[i] << endl;
* you must keep track of the array size   
  const int A\_LENGTH = 3;  
  class ArrayHolder  
  {  
  private:  
   int a[A\_LENGTH];  
   ...  
  public:  
   void print( ostream & out )  
   {  
   for ( int i = 0; i < A\_LENGTH; i++ )  
   out << a[i] << endl;  
   }  
  };

**Extended Example**

* EG class TimeSheet   
   #include <iostream>  
  const int DAYS\_PER\_WEEK = 7;  
  class TimeSheet  
  {  
  private:  
   int hoursWorked[DAYS\_PER\_WEEK];  
  public:  
   TimeSheet()  
   {  
   for ( int i = 0; i < DAYS\_PER\_WEEK; i++ )  
   hoursWorked[i] = 0;  
   }  
   void print( ostream & out )  
   {  
   for ( int i = 0; i < DAYS\_PER\_WEEK; i++ )  
   out << "On day "  
   << i  
   << " worked "  
   << hoursWorked[i]  
   << " hours\n";  
   }  
   void recordHours(int i, int hours)  
   {  
   if ( !(i >= 0 && i < DAYS\_PER\_WEEK && hours >= 0 ))  
   printf(“Error: invalid input to recordHours\n”);  
   hoursWorked[i] = hours;  
   }  
   int totalHours()  
   {  
   int totalHours = 0;  
   for ( int i = 0; i < DAYS\_PER\_WEEK; i++ )  
   totalHours += hoursWorked[i];  
   if ( totalHours < 0 ) cout << “Error: negative hours\n”;  
   return totalHours;  
   }  
  };
* EG using class TimeSheet   
  int main()  
  {  
   TimeSheet mySheet;  
   mySheet.recordHours(MON, 8);  
   mySheet.recordHours(TUE, 9);  
   mySheet.recordHours(WED, 6);  
   mySheet.recordHours(THU, 9);  
   mySheet.recordHours(FRI, 4);  
   mySheet.print( cout );  
   cout << "Worked "  
   << mySheet.totalHours()  
   << " total hours this week\n";  
   return 0;  
  }

**Character Arrays (AKA character strings)**

* character strings are arrays of characters terminated by '\0'
* tricky thing is you need an extra element for the terminator
* Three examples (of the string containing "abc")   
  char s1[4] = {'a','b','c','\0'};  
  char s2[4] = "abc";  
  char s3[] = "abc";

**Searching a character string for a specified character**

* to find the index of an element containing a specified value   
  int findIndexOfChar(char c, char s[]/char\* s)  
  {  
   for ( int i = 0; s[i] != '\0'; i++ )  
   if ( s[i] == c )  
   return i;  
   return -1;  
  }
* example of use   
  int main()  
  {  
   char s[] = "Hello There";  
   int posT = findIndexOfChar( 'T', s );  
   if ( posT == -1 )  
   cout << "T is not in " << s << endl;  
   else  
   cout << "T is at position " << posT << endl;  
   s[posT] = 'W';  
   cout << s << endl; // prints: Hello Where  
  }

**String Library Functions**

* important low-level C-string utilities   
   #include <cstring>  
  int strlen(const char s[]);  
  char [] strcpy(char dst[], const char src[]);  
  char [] strcat (char dst[], const char src[]);

int strcmp(const char s1[], const char s2[]);

// not until HW4 char [] strdup(const char s[]);

[Examples from Internet that don't work](http://www.cplusplus.com/forum/beginner/159569/)

**String Class**

* always useful to use a class around a character array

#include <iostream>  
const int STRING\_LENGTH = 128; // max length of a string

class String  
{  
private:  
 char buffer[STRING\_LENGTH];

static bool streq( char \*buf1, char \*buf2 )  
 {  
 int i;  
 for ( i = 0; buf1[i] != '\0' && buf2[i] != '\0'; i++ )  
 if ( buf1[i] != buf2[i] )  
 return false;  
 return buf1[i] == buf2[i];  
 }  
public:  
 String( const char s[] = "" )  
 {  
 int i;  
 for ( i = 0; s[i] != '\0' && i < STRING\_LENGTH - 1; i++ )  
 buffer[i] = s[i];  
 buffer[i] = '\0';

// **better**: strcpy(this->buffer, s); let strcpy do the for loop

}  
 void print( ostream & out )  
 {  
 out << buffer;  
 }  
 void read( istream & in )  
 {  
 in >> buffer; // will read next word from in  
 // **better**: in.getline(buffer, STRING\_LENGTH); will read line into buffer

}

bool operator == ( String w2 )  
 {  
 return streq( this->buffer, w2.buffer );  
 // **better**: strcmp(this->buffer, w2.buffer) == 0;

}

};  
istream & operator >> ( istream & in, String & w )  
{  
 w.read( in );  
 return in;  
}  
ostream & operator << ( ostream & out, String w )  
{  
 w.print( out );  
 return out;  
}