CST 280 Advanced C++ Programming
Week 1

Topics:

- Course introduction and syllabus
- C++ review: control structures, functions
- C-string processing
- C++ modules and information hiding
- Basic software design techniques

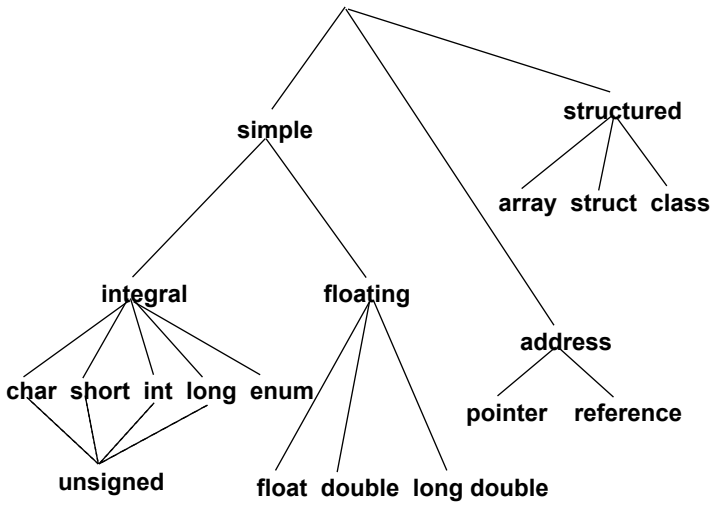
C++ Review

- General program structure
- Data types and variables
- Program control structures
- Functions
- Input/Output
- File processing
- C++ strings

C++ General Program Structure

```
// This program calculates the user's pay.

#include <iostream>
using namespace std;

int main()
{
    float hours, rate, pay;

    cout << "How many hours did you work? ";
    cin >> hours;
    cout << "How much do you get paid per hour? ";
    cin >> rate;
    pay = hours * rate;
    cout << "You have earned $" << pay << endl;

    return 0;
}
```

C++ Data Types



Declaration

- **All named storage identifiers must be *declared***

- **Variable: can be changed**
- **Constant: cannot be changed**
- **Examples**

```
const   double pi = 3.141592;
const   int numElems = 35;
int     counter;
double  salary = 25234.45;
char    code = 'X';
```

Assignment Statement

Variable = Expression

Arithmetic Operators

+	Unary plus
–	Unary minus
+	Addition
–	Subtraction
*	Multiplication
/	Floating point division Integer division
%	Modulus

Arithmetic Operator Precedence

Precedence:				
1	(	)		
2	*	/	%	(left-to-right)
3	+	–		(left-to-right)

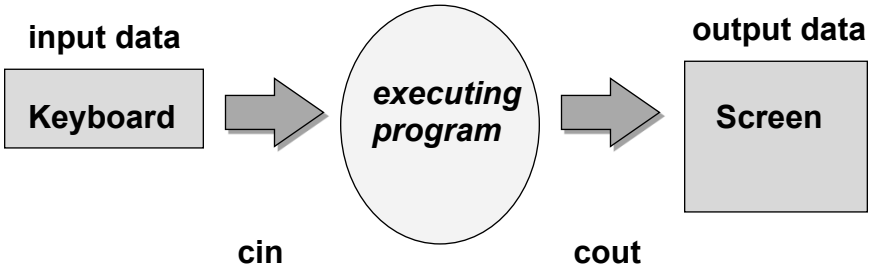
C++ Math Functions

- Require `#include<cmath>`
- Selected functions:

`abs(n)`	`tan(x)`
`sqrt(x)`	`sin(x)`
`floor(x)`	`cos(x)`
`ceil(x)`	`atan(x)`
`exp(x)`	`asin(x)`
`log(x)`	`acos(x)`
`log10(x)`	`pow(x,y)`

Keyboard and Screen I/O

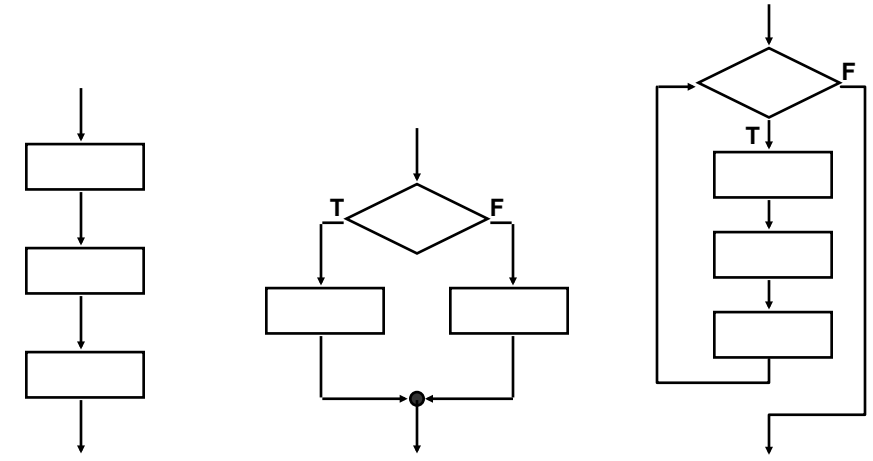
```
#include <iostream>
```



Console I/O

```
#include  <iostream>
int main()
{  int      partNumber;
   double   unitPrice;

   cout << "Enter part number followed by return : " <<  endl ;
   cin  >>  partNumber ;
   cout <<  "Enter unit price followed by return : " <<  endl ;
   cin  >>  unitPrice ;

   cout <<  "Part # " <<  partNumber
        <<  " at  Unit Cost: $ " <<  unitPrice  <<  endl;
   return 0;

}
```

General Control Structures

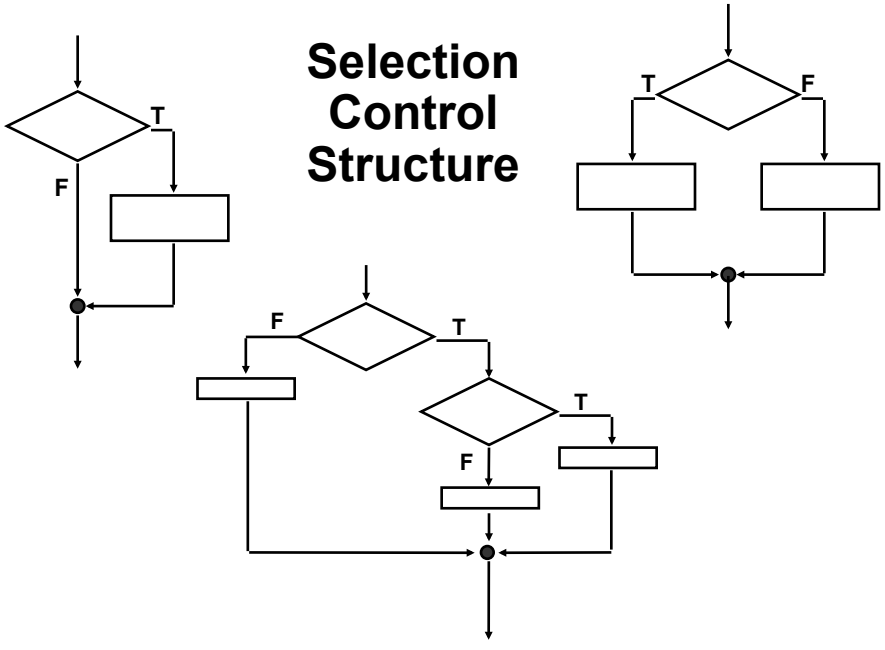


Sequence **Selection** **Repetition**

C++ Control Structures

- `if` ...
- `for` - loop
- `while` - loop
- `do...while` - loop
- `switch`

Selection Control Structure



if Statement

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int score1, score2, score3;
   double average;

   cout << "Enter 3 test scores and I will average them: ";
   cin >> score1 >> score2 >> score3;

   average = (score1 + score2 + score3) / 3.0;
   cout << fixed << showpoint << setprecision(1);
   cout << "Your average is " << average << endl;

   if (average > 95)
      cout << "Congratulations! That's a high score!" << endl;

   return 0;
}
```

if Statement

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int score1, score2, score3;
   double average;

   cout << "Enter 3 test scores and I will average them: ";
   cin >> score1 >> score2 >> score3;

   average = (score1 + score2 + score3) / 3.0;
   cout << fixed << showpoint << setprecision(1);
   cout << "Your average is " << average << endl;

   if (average = 100)
      cout << "Congratulations! That's a perfect score!";

   return 0;
}
```

Where is the error?

if Statement

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int score1, score2, score3;  // To hold three test scores
   double average;              // TO hold the average score

   cout << "Enter 3 test scores and I will average them: ";
   cin >> score1 >> score2 >> score3;

   average = (score1 + score2 + score3) / 3.0;
   cout << fixed << showpoint << setprecision(1);
   cout << "Your average is " << average << endl;

   if (average > 95)
   {
      cout << "Congratulations!\n";
      cout << "That's a high score.\n";
      cout << "You deserve a pat on the back!\n";
   }

   return 0;
}
```

if-else Statement

```
#include <iostream>
using namespace std;

int main()
{
   int number;

   cout << "Enter an integer" << endl;
   cin >> number;

   if (number % 2 == 0)
      cout << number << " is even.";
   else
      cout << number << " is odd.";

   return 0;
}
```

Nested-if

```
int main()
{
   cout << "What is your annual income? ";
   double income;    //variable definition
   cin >> income;

   if (income >= 35000)
   {
      cout << "How many years have you worked at your current job?";
      int years;
      cin >> years;

      if (years > 5)
         cout << "You qualify." << endl;
      else
      {
         cout << "You must have been employed for\n";
         cout << "more than 5 years to qualify.\n";
      }
   }
   else
      cout << "You must earn at least $35,000 to qualify" << endl;

   return 0;
}
```

Extended-if

```
int main()
{
   int testScore;
   char grade;

   cout << "Enter your numeric test score and I will\n";
   cout << "tell you the letter grade you earned: ";
   cin >> testScore;

   if (testScore < 60)
      cout << "Your grade is E.";
   else if (testScore < 70)
      cout << "Your grade is D.";
   else if (testScore < 80)
      cout << "Your grade is C.";
   else if (testScore < 90)
      cout << "Your grade is B.";
   else if (testScore <= 100)
      cout << "Your grade is A.";
   else
       cout << "We do not give scores higher than 100.";

   return 0;
}
```

Logical Operators

OR	**||**
AND	**&&**
NOT	**!**

- **Used to form *<u>compound</u>* logical expressions**

Truth Table: Logical AND

Logical expression *x*	Logical expression *y*	*x* && *y*
TRUE	TRUE	
TRUE	FALSE	
FALSE	TRUE	
FALSE	FALSE	

Truth Table: Logical OR

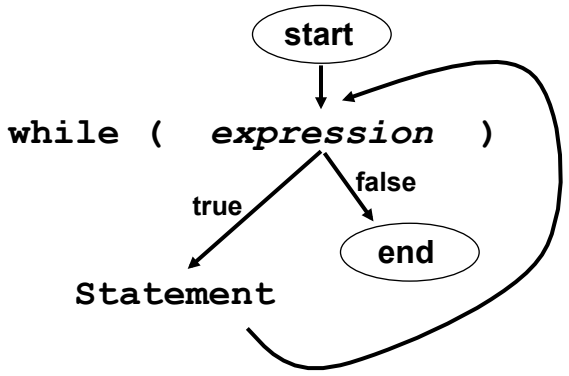
Logical expression *x*	Logical expression *y*	*x* || *y*
TRUE	TRUE	
TRUE	FALSE	
FALSE	TRUE	
FALSE	FALSE	

Truth Table: Logical NOT

Logical expression *x*	! *x*
TRUE	
FALSE	

```
int main()
{
   cout << "What is your annual income? ";
   double income;    // Variable definition
   cin >> income;

   cout << "How many years have you worked at "
        << "your current job? ";

   int years;
   cin >> years;

   if (income >= 35000 && years > 5)
      cout << "You qualify.";
   else
   {
      cout << "You must earn at least $35,000 or have\n";
      cout << "been employed for more than 5 years.";
   }

   return 0;
}
```

C++ `while` Loop

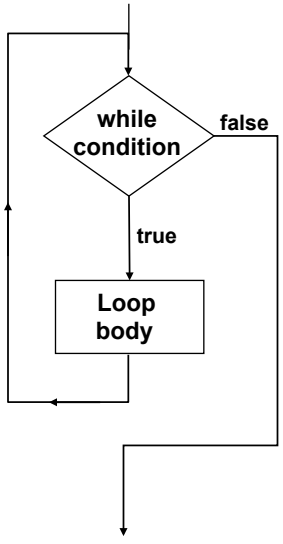


```
#include <iostream>
using namespace std;

int main()
{
   int number = 1;
   while (number <= 5)
   {
      cout << "Hello" << endl;
      number++;
   }
   cout << "That's all!\n";

   return 0;
}
```

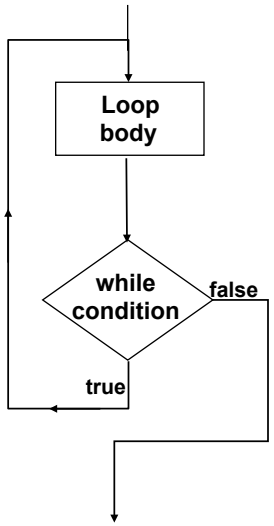
```
// This program calculates the number of soccer teams
// that a youth league may create from the number of
// available players. Input validation is demonstrated
// with while loops.
#include <iostream>
using namespace std;

int main()
{
   int players,      // Number of available players
       teamPlayers,  // Number of desired players per team
       numTeams,     // Number of teams
       leftOver;     // Number of players left over

   // Get the number of players per team.
   cout << "How many players do you wish per team?\n";
   cout << "(Enter a value in the range 9 - 15): ";
   cin >> teamPlayers;

   // Validate the input.
   while (teamPlayers < 9 || teamPlayers > 15)
   {
      cout << "You should have at least 9 but no\n";
      cout << "more than 15 per team.\n";
      cout << "How many players do you wish per team? ";
      cin >> teamPlayers;
   }
```

```
// Get the number of players available.
cout << "How many players are available? ";
cin >> players;

// Validate the input.
while (players <= 0)
{
   cout << "Please enter a positive number: ";
   cin >> players;
}

// Calculate the number of teams.
numTeams = players / teamPlayers;

// Calculate the number of leftover players.
leftOver = players % teamPlayers;

// Display the results.
cout << "There will be " << numTeams << " teams with ";
cout << leftOver << " players left over.\n";
```

_____ Loop

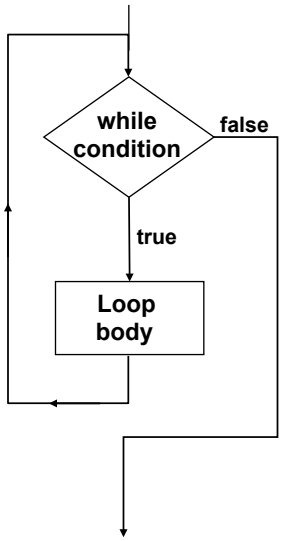


_____ Loop



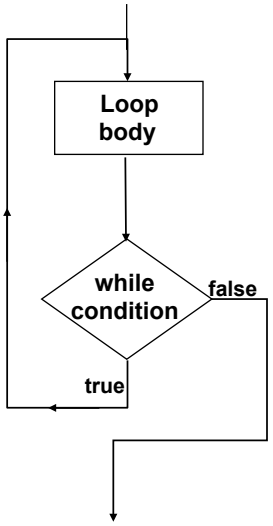
`while` Loop

Pre-Test



`do-while` Loop

Post-Test

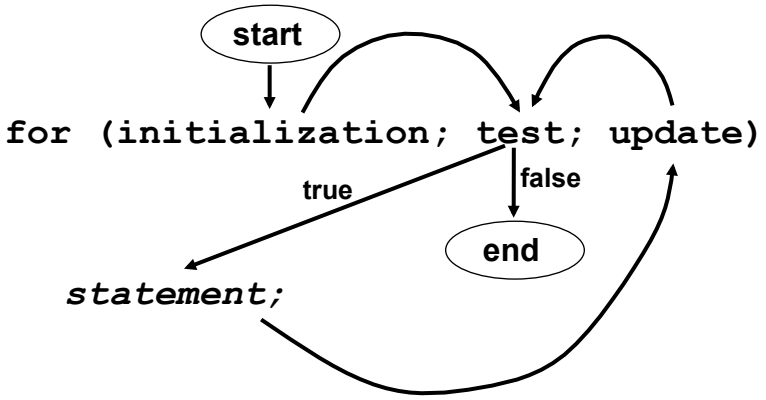


- **For *variable condition repetition***

`do-while` Loop

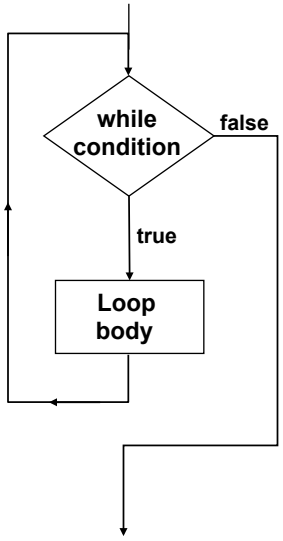
```
#include <iostream>
using namespace std;

int main()
{
   int score1, score2, score3;
   double average;
   char again;

   do
   {
      cout << "Enter 3 scores and I will average them: ";
      cin >> score1 >> score2 >> score3;

      average = (score1 + score2 + score3) / 3.0;
      cout << "The average is " << average << endl;

      cout << "Do you want to average another set? (Y/N) ";
      cin >> again;
   } while (again == 'Y' || again == 'y');

   return 0;
}
```

C++ `for` Loop

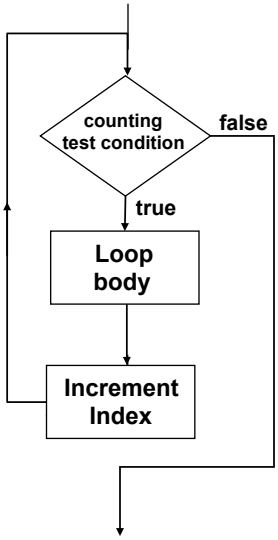


- For ***exact counting sequences***

`while` Loop



`for` Loop



for-Loop

```
int main()
{
   int num;
   int maxValue;

   cin >> maxValue;

   cout << "Number    Number Squared" << endl;
   cout << "------------------------" << endl;

   for (num = 1; num <= maxValue; num++)
      cout << num << "     " << (num * num) << endl;

   return 0;
}
```

C++ Functions

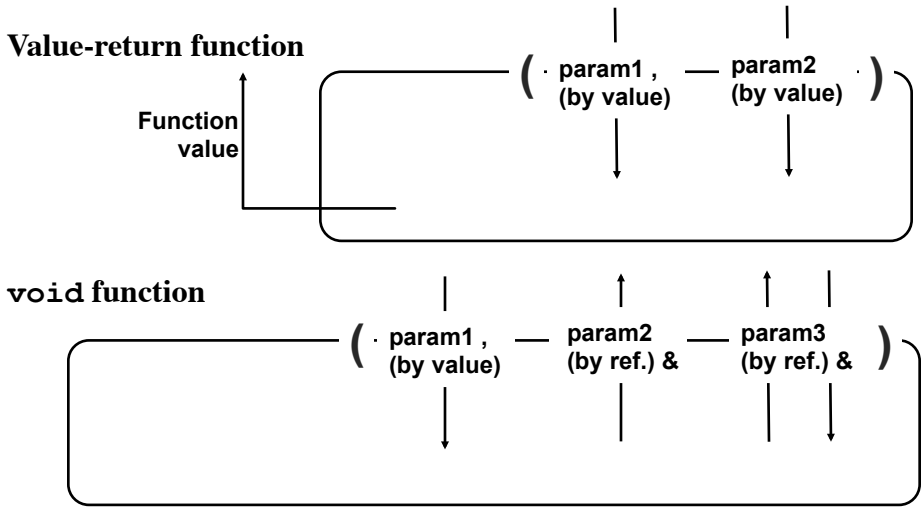
Function types?

-
-

Parameter types?

-
-

C++ Function Data Flow



Value Return Function

```
#include <iostream>
using namespace std;

int sum(int num1, int num2, int num3);

int main()
{
   int value1 = 20, value2 = 40, value3 = 60,
       total;

   total = sum(value1, value2, value3);

   cout << "The sum is " << total << endl;

   return 0;
}

// This function receives three integers and returns
// the sum
int sum(int num1, int num2, int num3)
{
   return num1 + num2 + num3;
}
```

Void Function

```
void DeltaAddr();

int main()
{
    DeltaAddr();

    return 0;
}
```

```
// This void function prints the Delta College address
// It is a VOID FUNCTION with NO PARAMETERS
void DeltaAddr()
{
    cout << "Delta College" << endl;
    cout << "1961 Delta Road" << endl;
    cout << "University Center, MI  48710" << endl;
    cout << endl;
}
```

Void Function

```
void CartToPolar(double x, double y, double& r, double& theta);

int main()
{
    double x = 3.0;
    double y = 5.0;
    double theta,radius;
    CartToPolar(x,y,radius,theta);
    cout << "(" << x << "," << y << ") in cartesian coordinates is "
         << "(" << theta << "," << radius << ") in polar coordinates";
    cout << endl << endl;

    return 0;
}

// This void function receives a cartesian coordinate and
// returns the equivalent point in polar coordinates
// It is a VOID FUNCTION with VALUE and REFERENCE PARAMETERS
void CartToPolar(double x, double y, double& r, double& theta)
{
    r = sqrt(x*x + y*y);
    theta = atan(y/x);
}
```

Program Demonstration: C++ Functions

Demo of Visual C++ development environment

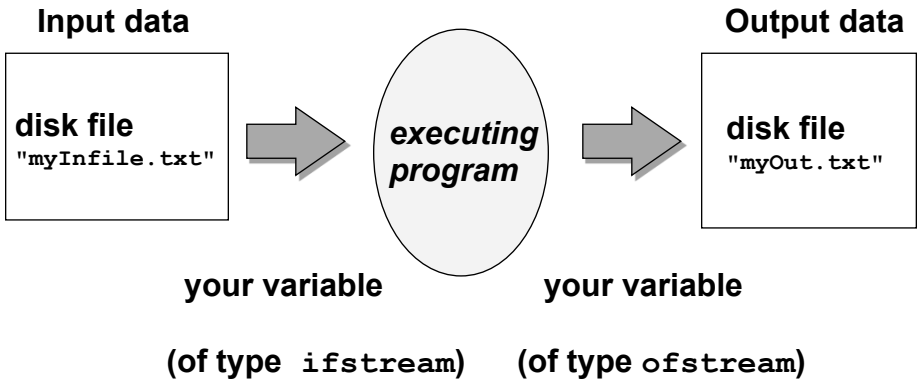
Demo of functions:

- Sum three numbers
- Print Delta College address
- Cartesian to polar coordinate conversion

File: `functDemo.cpp`

C++ Text Files for Input/Output

```
#include <fstream>
```

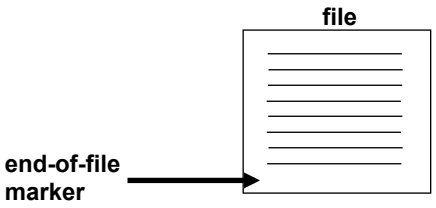


File Input/Output

- Library header required: `#include <fstream>`
- Input/output file stream declaration:
  - **Input:** `ifstream stream_identifier ;`
  - **Output:** `ofstream stream_identifier ;`
- Opening files:
  - `stream_identifier.open ("filename ");`
- Input from file:
  - `stream_identifier >> {variable list};`
- Output to file:
  - `stream_identifier << {expression list};`
- Closing files:
  - `stream_identifier.close();`

File-Controlled Loops

- Many file problems relate to processing an unknown number of input elements
- Requires method for detection of end-of-file marker



File Processing

```
fileIn >> data;        // Priming read
while ( ! fileIn.eof() )
{
   // Process data
   fileIn >> data;     // Continuation read
}
```

Note: *`streamIdentifier`*`.eof()` is a function that returns `true` when the file associated with *`streamIdentifier`* attempts to read the end-of-file marker.

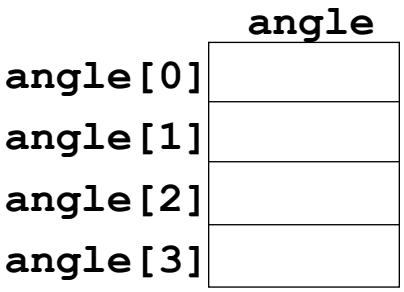
Program Demonstration

- **Read and average a list of numbers in a sequential file**
- **Text file processing with *variable number of items* using *end-of-file sentinel***
- **Includes *priming/continuation* reads**

File: `aveFile.cpp`

Arrays

Structured collection of elements, all of the same type, that is given a single name. Each component (array element) is accessed by an index that indicates the component's position within the collection



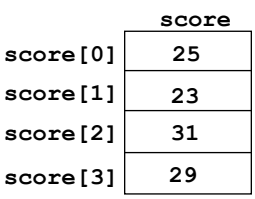
Using Arrays

Declaration:

```
DataType ArrayName[size];
```

Example:

```
int score[4];
```



Referencing:

```
score[0] = 25;
score[1] = 23;
score[2] = 31;
score[3] = 29;
```

Array Element Processing

- `for` loops often used for processing multiple array elements at once
- Example (set all array elements to zero)

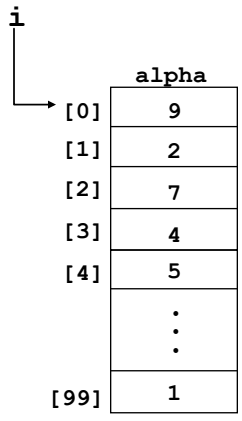
```
double alpha[100];
for (i = 0; i < 100; i++)
   alpha[i] = 0.0;
```

Array Processing

- Given:

```
double alpha[100];
for(int i = 0; i < 100; i++)
    cout << alpha[i] << endl;
```

- Variable `i` acts as a *pointer*
- `i` starts at index zero and is incremented to "point" to the next element, and so on
- Technique extensively used with array processing



```
int main()
{
   const int ARRAY_SIZE = 8;
   int numbers[ARRAY_SIZE] = {5, 10, 15, 20, 25, 30, 35};

   showValues(numbers, ARRAY_SIZE);
   return 0;
}

void showValues(int nums[], int size)
{
   for (int index = 0; index < size; index++)
      cout << nums[index] << " ";
   cout << endl;
}
```

Array Out-of-Bounds Issues

- **Value array indexes: 0 . . . *array size* –1**
- **Example:**

  `double alpha[100];`

  (implies that you can only access `alpha[0]...alpha[99]`)
- **Using index out of that range leads to "invasion" of adjacent storage elements**
- **Often a run-time error**
- **C++ does not check for *array out-of-bounds* errors**

Array Memory Allocation

- **Array sizes are allocated at compile time; cannot be changed**
- **Good array design includes definition of the largest possible array needed for any problem solution**
- **Later in course:**
  - **Coverage of dynamic arrays covered later in course**
  - **Introduction of resizable arrays (vectors)**

Passing Arrays as Parameters

- **Entire array can be passed to function as parameter**
- **Always passed as *reference* parameter**
- **Address used is base address of first array element (index `[0]`)**

Example: Array Parameters

```
const int MAX_ARRAY = 10;
int sumArray[ MAX_ARRAY ];
ZeroOut( sumArray, MAX_ARRAY );
```

in main()

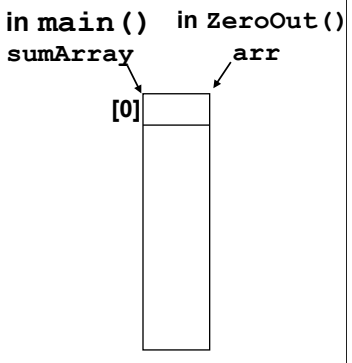
```
void ZeroOut(int arr[ ], int size )
{
  int i;

  for (i = 0; i < size; i++)
      arr[i] = 0;
}
```

Example: Array Parameters

In main()

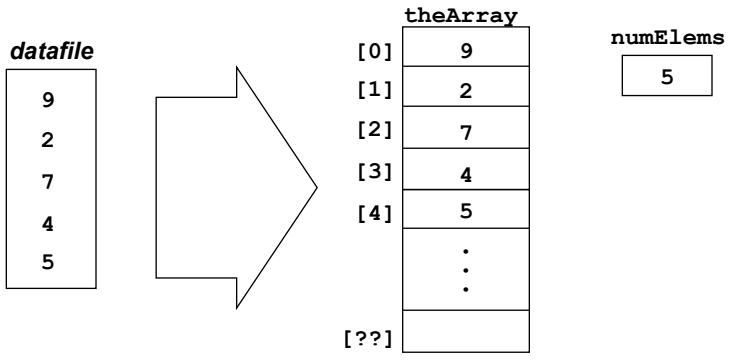
```
int sumArray[ 10 ];
ZeroOut( sumArray, 10 );
```

```
void ZeroOut(int arr[ ], int size )
{
  int i;

  for (i = 0; i < size; i++)
       arr[i] = 0;
}
```



Array Input to File

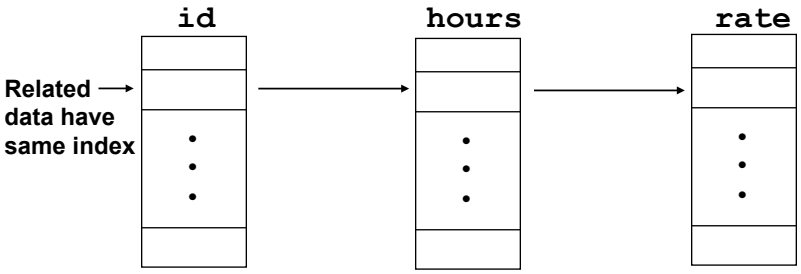
- **Read an unknown number of integer values from a file into an array**
- **Read the array and calculate the average of the list of values**



File: `aveFileArray.cpp`

Parallel Arrays

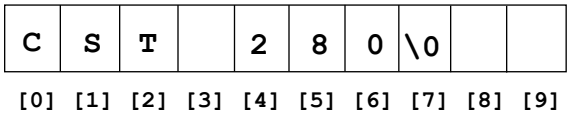
- **Multiple arrays; same size; related elements of different data types**
- **Example:**



Program Demo: `parallel.cpp`

C-Strings

- **Stored as an array of characters**
- **Terminated by *null character* ('\0')**
- **Example:**



String Declaration

- **Examples:**

```
char course[7];
char name[40];
char outputLine[100];
```

String Assignment

- **Declaration/initialization OK:**

```
char myStr[20] = "Hello World";
```

- **Assignment invalid:**

```
myStr = "Hello World";
```

A function must be used to assign strings

String Output

- **String arrays can be written directly to output: console (`cout`) or file**

```
char course[8] = "CST 280";

cout << course;
```

String Input Methods

1 `cin >> inputStr;`

Read strings delimited by white space

2 `cin.getline(inputStr, n);`

Read a string up `n-1` characters and "absorb" the new line character

String Processing

- **Storing strings as arrays allows access to individual characters**
- **Must always maintain the null (\0) character as string terminator**
- **C++ does offer some string functions, but generally string processing is weak**
- **The `string` *class* is now standard; covered soon in course**

Another Example: String Processing

```
int countVowels(char sentence[])
{
    char vowelSet[] = {'a','e','i','o','u',  ... };
    int vowels;
    int i, j;

    vowels = 0;
    i = 0;
    while (sentence[i] != '\0')
    {
      j = 0;
      while (vowelSet[j] != '\0')
      {
         if (sentence[i] == vowelSet[j])
            vowels++;
         j++;
      }
      i++;
    }

   return vowels;
}
```

File: `countVowels.cpp`

Text Encryption

- Encrypts/decrypts text using a substitution cypher:

  ABCDEFGHIJKLMNOPQRSTUVWXYZ

  KQBZCGOAWPMHVLFXEDURIYSJNT

- Decode: **HWGC WU OFFZ**

File: `encryption.cpp`

String Functions

`strlen(str)`	Return length of string
`strcpy(toStr, fromStr)`	Copy `fromStr` to `toStr`
`strcat(toStr, addStr)`	Concatenate (append) `addStr` to end of `toStr` and store result in `toStr`
`strcmp(str1, str2)`	Compare strings if… returns… `str1 == str2` 0 `str1 > str2` positive `str1 < str2` negative

all require `#include<string>`

String Assignment

- **Recall, direct c-string assignment is invalid:**

```
myStr = "Hello World";
```

- **Correct alternative:**

```
strcpy(myStr,"Hello World");
```

Invalid C-string Comparison

```
#include <iostream>
using namespace std;

int main()
{
   const int SIZE = 40;
   char firstString[SIZE], secondString[SIZE];

   // Get two strings.
   cout << "Enter a string: ";
   cin.getline(firstString, SIZE);
   cout << "Enter another string: ";
   cin.getline(secondString, SIZE);

   // Can you use the == operator to compare them?
   if (firstString == secondString)
      cout << "You entered the same string twice.\n";
   else
      cout << "The strings are not the same.\n";
   return 0;
}
```

Valid C-string Comparison

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   const int SIZE = 40;
   char firstString[SIZE], secondString[SIZE];

   // Get two strings
   cout << "Enter a string: ";
   cin.getline(firstString, SIZE);
   cout << "Enter another string: ";
   cin.getline(secondString, SIZE);

   // Compare them with strcmp.
   if (strcmp(firstString, secondString) == 0)
      cout << "You entered the same string twice.\n";
   else
      cout << "The strings are not the same.\n";
   return 0;
}
```

Building New String Functions

- C++ functions to perform:
  - –Substrings
  - –Character indexing (searching)

Building New String Functions: `substring`

```
void substring(char inStr[], int start, int end, char outStr[])
{
   strcpy(outStr, "");                  // Empty output string

   int pos = 0;                         // To mark curr pos. of evolving substring
   for (int i = start; i <= end; i++)
   {
      outStr[pos] = inStr[i];           // Move character to output string
      pos++;                            // Manually advance position marker
   }

   outStr[pos] = '\0';                  // Add null character to complete string
}
```

Building New String Functions: `strIndex`

```
int strIndex(char inStr[], int start, char searchTarget)
{
   int pos = start;                  // Set position marker to start index
   int returnPos = -1;               // Char position marker, assume not found
   bool found = false;               // Used to exit loop when found, assume not
                                     // found at start
   while (inStr[pos] != '\0' && !found)
   {
      if (inStr[pos] == searchTarget)   // If character found,
      {
         returnPos = pos;               //    mark position
         found = true;                  //    set switch allowing exit from loop.
      }
      else
         pos++;                         // Otherwise, advance to next character

   }
   return returnPos;                 // Return value
```

Using `typedef` with String Definitions

Examples:

```
typedef char cityString[41];
typedef char String50[51];

cityString theCity;
String50 firstName;
```

Example: **`typedefDemo.cpp`**

C++ Character Strings: Example

- State code validation:
  - Read two-character string from user
  - Validate and return result to user
  - Utilizes `typedef` to reduce string-related syntax

File: `stateCodeCheck.cpp`

C++ Character Strings: Example

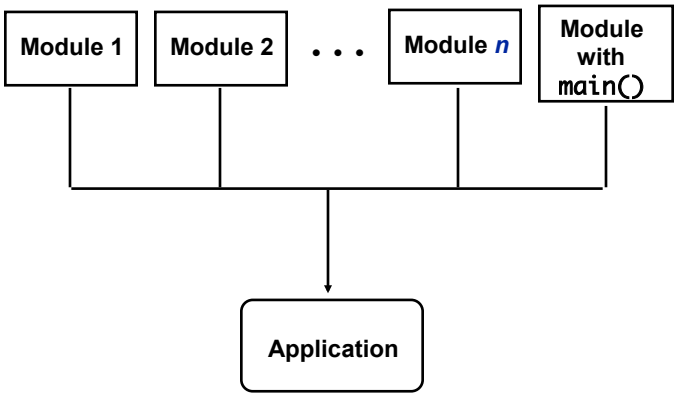
- Morse code converter:
  - Read table from data file
  - Scan character string
  - Convert individual characters to equivalent Morse code string
  - Performs table look-up

Files: `morseConvert.cpp`, `morse.txt`

C++ Modules

- Collection of one or more functions, data types, or constants
- Stored in separate physical file
- May be used by several client programs
- Can be separately compiled

Large Programs



C++ Module Layout

Specification File

```
int F1 ( );

int F2 ( );
```

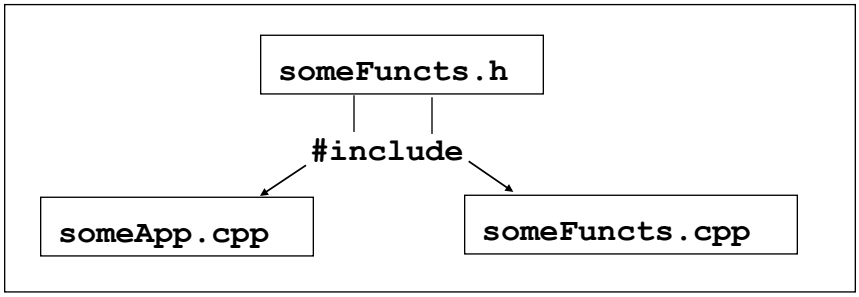
- Function prototypes
- The *public* view of the module

Implementation File

```
int F1 ( )
{
    :
    :
}


int F2 ( )
{
    :
    :
}
```

- Function definitions/logic
- The *private* view of the module

C++ Module Files



Project files

Value of Using Modules

- Division of labor
- Comprehensibility
- Managing complexity (abstraction)
- Implementation independence
- Information hiding

Example: Modular Programming

- Calculate the day of the week a given calendar date falls on
- Two program versions:
  - One file: `dateExam.cpp`
  - Two files: `dateExam2.cpp`, `datefun.h`, `datefun.cpp`

Information Hiding

Hiding the details of a function or data structure with the goal of controlling access to the details of a module or structure.

PURPOSE: To prevent high-level designs from depending on low-level design details that may be changed.

Program Demonstration

- **Sunrise/sunset table**
- **Utilizes file modularity for project**
- **Uses complex function "hidden" in a module**
- **Special features:**
  - **Use of `typedef`**
  - **Boolean (`bool`) variables**

Files: **`sunDemo.cpp, sun.h, sun.cpp`**

Programming Style

- **Develop your own style, but universal programming standards and rules exist**
- **Always write code thinking of the other people that will read it**
- **Inclusion of *comments* a universal part of software engineering and development**

C++ Comments

- **Comments: text in program ignored by compiler**
- **Types:**
  - **Double slash comments `//`**
    - ignores all text after symbol
  - **C-style comments `/* */`**
    - ignores all text between symbols

C++ Comments: Examples

```
/* This program converts inches to centimeters
   Property of:
       Delta College
       University Center, MI
*/
                  .
                  .
                  .
// Variable declarations
   int inches;
   double centimeters;
                  .
                  .
                  .
   centimeters = 2.54 * inches;  // Conversion factor
```

Program Formatting Requirements

- **Comments**
  - **Name: your name on all programs**
  - **Description comment of purpose required <u>at top of all programs</u>**
  - **Declaration: comment key variables**
  - **In-line: describe key sections of code (like input, calculations, output)**
  - **Sidebar: comment important or complex statements**
- **Identifiers**
  - **Variables: use *self-documenting* names**
  - **Constants: use for hard-coded numbers and for constant numbers that could later change**

Program Formatting Requirements (continued)

- Formatting Lines and Expressions
  - Break long lines into two to avoid word wrap when printing
  - Align multi-line statements for clarity
  - Align comments for clarity
- White space
  - Use double-spacing to separate key areas of code
- Indentation
  - Follow traditional rules for indenting code sections
    - » Indent all code in program blocks ({ })
    - » Indent code in if-statements and loops

Topic:
Basic Software Design Techniques

- General program structure
- Data types and variables
- Program control structures
- Functions
- Input/Output
- File processing
- C++ strings

Top-Down Programming

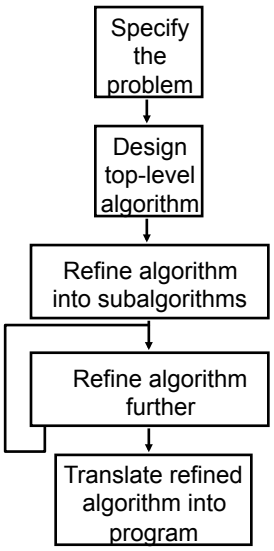
Necessary for solving software problems that are:

- **large**
- **complex**

An Algorithm Is . . .

A logical sequence of discrete steps that describes a complete solution to a given problem computable in a finite amount of time.

Top-Down Design



- Decompose problem into smaller problems
- Refine (add detail to) algorithm in *stepwise* fashion
- Eventually, refinement is sufficient for translation into programming language

Basic Design Techniques

- **High-level decomposition**
- **Graphical model:**
  - **Data flow diagram**
  - **Hierarchical structure chart**
- **Text specifications**
- **Pseudocode**

Decomposition

- Very high-level sequential steps to solve problem
- Example:

Input data values
Calculate average
Print result

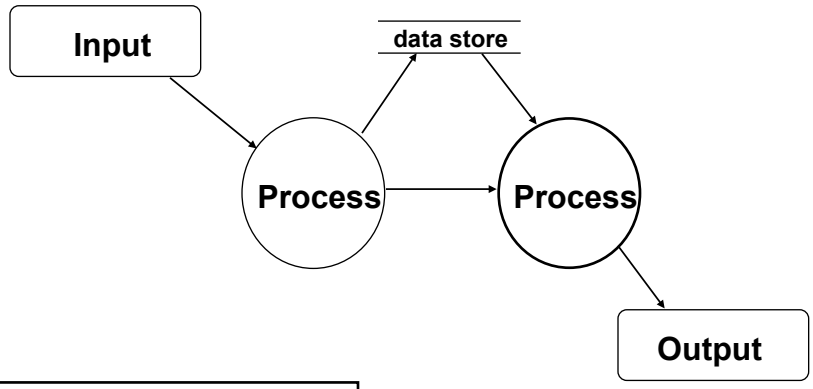
Problem Decomposition Guidelines

- Work to decompose any action into no more than 5 - 9 sub-actions
- Practice abstraction. Postpone details till later
- Debug at each level of refinement

Practice

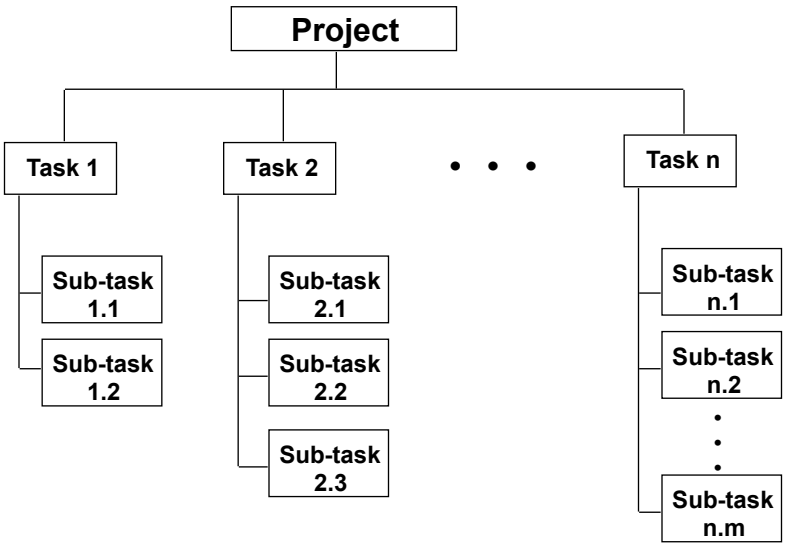
- **Decompose the major modules of an online course management (eLearning) system**

Data Flow Diagram



Each arrow represents information passed between modules, data stores, and Input/Output devices

Hierarchical Structure Chart



Program Module Specifications

- Title
- Description
- Input
- Processing requirements
- Special algorithms
- Output
- Error-handling
- Example

Pseudocode

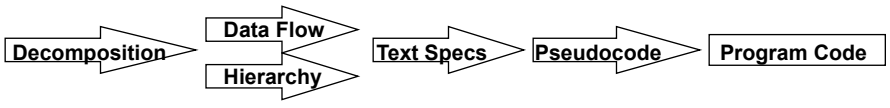
- **Pseudocode = "almost" code**
- **Very useful for "language-independent" algorithm representation**
- **Example:**

Procedure Average(array_in, array_size, average_out)

```
sum <-- 0;
FOR i <--  1 to array_size
      sum <--  sum + array_in[i];
average_out <--  sum/array_size
```

Top-Down Design

- Utilize design techniques mentioned
- Typical "top-down" progression of design:



- Software/system requirements emerge in "step-wise" fashion and are refined in detail at each step.
- May help with question: Where do I start?