CS 162 NOTES

CH 1-10, 12, 15, 17

1. OVERVIEW

* Computers can perform arithmetic and logical operations
* Computers have hardware and software
* The CPU and the main memory are hardware
* All programs are put into main memory before they can be executed
* When the power is off, everything in main memory is lost
* Hard disks, flash drives, etc, are secondary storage where you can store permanent information
* The operating system handles the overall activity of the computer and provide services
* Machine language is made up of sequences of 0’s and 1’s. every computer directly understands its own machine language.
* A bit is a binary digit, 0 or 1
* A byte is a sequence of eight bits
* Sequences of 0s and 1s are binary code
* Assembly language uses easy-to-remember instructions called mnemonics
* Assemblers are programs that translate a program written in assembly language into machine language
* Compilers are programs that translate a program written in a high-level language into machine code, called OBJECT code
* A linker links the object code with other programs provided by the IDE and used in the program to produce executable code
* Six steps are needed to execute a C++ program: edit, preprocess, compile, link, load, and execute
* A loader transfers executable code into main memory
* An algorithm is a step by step problem solving process in which a solution is arrived at in a finite amount of time
* The problem-solving process has three steps: analyze the problem and design an algorithm, implement the algorithm in a programming language, and maintain the program
* STRUCTURED design: a problem is divided into smaller sub-problems. Each sub-problem is solved and all the solutions to all the sub-problems are combined to solve the problem.
* OBJECT-ORIENTED design (OOD): a program is a collection of interacting objects.
* An object consists of data and operations on that data.

1. BASIC ELEMENTS OF C++

* Every C++ program has a function called main
* Reserved words cannot be used as identifiers in a program
* All reserved words in C++ consist of lowercase letters
* Identifiers are names of things and consist of letters, digits, and underscores—and in C++ it must begin with a letter or an underscore
* Whitespaces include blanks, tabs, and newline characters
* A data type is a set of values together with a set of allowed operations
* C++ data types have three categories: SIMPLE, STRUCTURED, and POINTERS
* SIMPLE: integral (char, int, bool, etc), floating point, and enumeration
* Arithmetic operations are addition +, subtraction -, multiplication \*, division /, and modulus % (modulus takes only integer operands)
* Mixed operands will convert an integer into a decimal
* A string is a sequence of zero or more characters. Zero characters is called a null string
* The length of a string is the number of characters in it
* Contents of a named constant cannot be changed
* C++ does not automatically initialize variables
* All variables have a name, value, data type, and size
* >> is a stream extraction operator
* << is a stream insertion operator
* \ is the escape character
* \n is a newline escape sequence
* All preprocessor commands start with #
* Preprocessor commands are processed by the preprocessor before the program goes through the compiler
* To use cin or cout you must have the header file #include <iostream> and the statement using namespace std;
* Semicolon is the statement terminator
* A C++ system has three components: ENVIRONMENT, LANGAUGE, and STANDARD LIBRARIES
* Standard libraries are not part of the C++ language, they contain functions to perform operations, like math
* C++ provides five compound operations (corresponding to + - / \* %). += -= \*= %=

1. INPUT/OUTPUT

* Cin stands for common input, and cout is common output
* When inputting data into a variable, the operator >> skips all leading whitespace
* The function “get” is used to read data on a character-by-character basis and does not skip any leading whitespace
* The function “ignore” is used to skip data in a line
* The function “putback” returns the last character retrieved by “get” and puts it back in the input stream
* The function “peek” returns the next character from the input stream but does not remove the character from the input stream
* Attempting to read invalid data into a variable causes the input stream to enter the fail state. You must use “clear” to restore the input stream to a working state
* The manipulator “setprecision” formats the output of floating-point numbers to a specified number of decimals
* The manipulator “fixed” outputs floating point numbers in the fixed decimal format
* “showpoint” outputs floating point numbers with decimal points and trailing zeros
* “setw” formats the output of an expression in a specific number of columns. The default is right-justified
* If the number of columns specified in the argument of setw is less than the number of columns needed to print the value of the expression, the output is not truncated and the output of the expression expands to the required number of columns
* Stream functions GET, IGNORE, PUTPACK, PEEK, CLEAR, UNSETF use headerfile iostream
* Manipulators SETPRECISION, SETW, SETFILL use the headerfile iomanip

1. CONTROL STRUCTURES (SELECTION)

* Control structures alter the normal flow of control
* The two most common are selection and repetition
* Selection incorporates decision into the program
* Relational operators == < <= > >= !=
* Characters are compared using a machine’s collating sequence
* Logical expressions evaluate to 1 (true) and 0 (false)
* There are two selection structures in C++: one-way (if), and two-way (if…else).
* The expression in if/if…else is usually logical
* An else statement gets paired with the most recent if statement (if it’s also unpaired)
* Int variables can be used to store the value of a logical expression
* Bool variables can be used to store the value of a logical expression
* The logical operators are ! (not), && (and), and || (or)
* A sequence of statements enclosed between curly braces {} is called a compound statement or a block of statements, and is treated as a single statement
* The “switch” structure is used to handle multi-way selection
* The execution of a break statement in a switch statement immediately exits the switch structure
* If certain conditions are not met in a program, it can be terminated using the “assert” function

1. CONTROL STRUCTURES (REPITITION)

* C++ has three looping (repetition) structures: while, for, and do…while
* The body of the while loop must contain a statement that eventually sets the expression to false
* A counter-controlled while loop uses a counter to control the loop, which must be initialized before the loop, and the body must contain a statement that changes the value of the counter
* A sentinel is a special value that marks the end of the input data. The sentinel must be similar to, yet differ from, all the data items
* A sentinel-controlled while loop uses a sentinel to control the loop. The while loop continues until the sentinel is read
* An EOF-controlled while loop uses an end-of-file marker to control the loop.
* In Windows, the EOF maker is entered using Ctrl+z. in UNIX Ctrl+d
* A for loop simplifies the writing of a counter-controlled while loop
* For (initialize statement; loop condition; update statement)
* Do (statement) while (expression)
* Both while and for loops are called pretest loops. A do…while is a posttest loop
* The while and for bodies may not execute at all, but the do…while loop body always executes at least once
* Executing a “continue” statement in the body of a loop skips the loop’s remaining statements and proceeds with the next iteration
* After a “continue” statement executes in a for loop, the update statement is the next statement executed

1. USER-DEFINED FUNCTIONS

* Functions, also called MODULES, are like miniature programs
* Functions enable you to divide a program into manageable tasks
* C++ system provides the standard (predefined) functions
* To use a standard function you must: 1) know the name of the headerfile, 2) include the headerfile in the program 3) know the name and type of the function, and number and types of the parameter (arguments)
* There are two types of USER-DEFINED functions: value-returning and void
* Variables defined in a function heading are called FORMAL PARAMETERS
* Expressions, variables, or constant values used in a function call are called ACTUAL PARAMETERS
* In a function call, the number of actual parameters and their types must match with the formal parameters in the order given
* A value-returning function is used (called) in either an expression or an output statement or as a parameter in a function call
* fucntionType functionName (formal parameter list) {statements}
* a function with no parameters still needs an empty parentheses
* value-returning functions return its value via the “return” statement
* a function can have more than one return statement, however, whenever a return statement executes, the remaining statements are skipped and the function exits
* A RETURN statement returns ONLY ONE VALUE
* A function prototype is the function heading without the body, and ends with a semicolon
* In a function PROTOTYPE the NAMES of the variables in the formal parameter list are OPTIONAL
* Function prototypes are placed before every function definition, including main
* When you use function prototypes, user-defined functions can appear in any order in the program
* In a FUNCTION CALL statement, you specify ONLY the ACTUAL PARAMETERS, not their data type or the function type
* A function with no data type is a void function
* A return statement without any value can be used in a void function, and is typically used to exit the function early
* A void function may or may not have parameters
* A CALL to a VOID function is a STANDALONE statement
* There are TWO types of FORMAL PARAMETERS: VALUE parameters, and REFERENCE parameters
* A VALUE parameter receives a copy of its corresponding ACTUAL parameter
* A REFERENCE parameter receives the address (memory location) of it’s corresponding ACTUAL parameter
* The corresponding actual parameter of a value parameter is an expression, a variable, or a constant value
* A constant value cannot be passed as a reference parameter
* The corresponding ACTUAL parameter of a REFERENCE parameter must be a VARIABLE
* When you include & after the data type of a formal parameter, the formal parameter becomes a reference parameter
* The stream variables should be passed by reference to a function
* If a formal parameter needs to change the value of an actual parameter, in the function heading, you must declare this formal parameter as a reference parameter
* The scope of an identifier refers to those parts of the program where it is accessible
* Variables declared within a function (or block) are called LOCAL VARIABLES
* Variables declared outside of every function definition (and block) are called GLOBAL VARIABLES
* The scope of a function name is the same as the scope of an identifier declared outside of any block
* C++ does not allow the nesting of function definitions
* An AUTOMATIC variable is a variable for which memory remains allocated throughout the execution of the program
* By default, global variables are static variables
* Two functions are said to have different formal parameter lists if both functions have: 1) a different number of formal parameters, or 2) the same number of formal parameters and the data types of the formal parameters, in the order listed, differ in at least one position.
* If you do not specify the value of a default parameter, the default value is used for that parameter
* Default values can be constants, global variables, or function calls
* You CANNOT assign a CONSTANT VALUE as a default value to a REFERENCE PARAMETER

1. USER-DEFINED SIMPLE DATA TYPES, NAMESPACES, AND THE STRING TYPE

* An Enumeration type is a set of ordered values
* Enum typeName {value1, value2, …} \*value1 and value2 are identifiers and value1<value2<…
* No arithmetic operations are allowed on enum types
* Relational operators can be used on enum values
* Enum type values cannot be input or output directly.
* Enum types can be passed as parameters to functions either by value or reference
* A function can return a value of the enum type
* An anonymous type is one in which a variable’s values are specified without any type name
* Typedef is used to create synonyms or aliases to previously defined data types
* Anonymous types cannot be passed as parameters to functions
* A namespace member is usually a named constant, variable, function, or another namespace
* The scope of a namespace member is local to the namespace
* To access a namespace member outside the namespace—precede the namespace member name with the namespace name and scope resolution operator (Song::Song())
* The USING statement simplifies the accessing of namespace members
* The operator + can be used to concatenate two values of String type. For + to work, one of the operands must be a string variable
* Relational operators can be used on strings
* Position of the first character in a string is 0.
* String type contains functions such as at, append, clear, compare, erase, find, find\_first\_of, find\_first\_not\_of, insert, length, replace, size, substr, and swap to manipulate strings.

1. ARRAYS AND STRINGS

* A data type is simple if variables of that type can hold only one value at a time
* In a structured data type, each data item is a collection of other data items
* An array is a structured data type with a fixed number of components. Every component is of the same type, and components are accessed using their relative positions in the array
* Elements in a one-dimensional array are formed like a list
* There is no check on whether an array index is out of bounds
* An array index can be any expression that evaluates to a nonnegative integer. The value of the index must always be less than the size of the array
* There are no aggregate (whole formed of separate elements) operations on arrays, except for the input/output of character arrays (c-strings)
* An array can be initialized during declaration. If there are fewer initial values than the array size, the remaining elements are initialized to 0.
* The base address of an array is the address of the first array component. If list is one-dimensional, then the base address of list is list[0]
* When declaring a one-dimensional array as a FORMAL parameter, you usually omit the size of the array. If you specific the size of the array in the formal parameter declaration, the compiler will ignore the size.
* In a function call statement, when passing an ACTUAL parameter, you use only its name
* As PARAMETERS TO FUNCTIONS, arrays are passed ONLY by REFERENCE
* Because as parameters, arrays are passed by reference only, when declaring an array as a formal parameter, you do not use the symbol & after the data type
* A function CANNOT RETURN a value of type ARRAY
* Although as parameters, arrays are passed by reference, when declaring an array as a formal parameter, using the reserved word const before the data type prevents the function from modifying the array
* Individual array components can be passed as parameters to functions
* The sequential search algorithm searches a list for a given item, starting with the first element in the list. It continues to compare the search item until the item is found or the list has no more elements left to compare
* Selection sort sorts the list by finding the smallest (or equivalently largest) element in the list and moving it to the beginning (or end) of the list
* For a list of length *n*, selection sort makes exactly *n(n-1)/2* key comparisons and *3(n-1)* item assignments
* C-strings are null terminated, represented as ‘\0’ … or in ASCII 0.
* Input and output of c-strings is the only place where C++ allows aggregate operations
* The headerfile cstring has functions that can be used for c-string manipulation: strcpy(string copy), strcmp(string comparision), and strlen(string length)
* Cstrings are compared character by character
* Parallel arrays are used to hold related information
* In a two-dimensional array, the elements are arranged in table form
* To access an element in a two dimensional array, you need a pair of indices: one for row and one for column.
* The rows are numbered 0-(row\_size-1) and same with columns
* Matrix is a two dimensional array. The base address of matrix is matrix[0][0]
* In row processing, two dimensional array is processed one row at a time. Same in column processing
* When declaring a two-dimensional array as a formal parameter, you can omit the size of the first dimension, but NOT the second.
* When a two-dimensional array is passed as an actual parameter, the number of columns of the actual and formal arrays must match
* C++ stores (in computer memory) two dimensional arrays in a row-order form.

1. RECORDS (STRUCTS)

* A struct is a collection of a fixed number of components, which can be of different types
* Syntax: struct structName {dataType1 identifier 1; dataType2 identifier2; …};
* Struct is a definition, no memory is allocated. Memory is allocated for the struct variables only when you declare them
* Components of a struct are called members of the struct, and are accessed by name
* The dot (.) operator is called the member access operator
* Members of a struct are accessed by using the dot (.) operator. EX: employeeType is a struct, employee is a variable of type employeeType, and name is a member of type employee. So employee.name accesses the member name—meaning employee.name is a variable and can be manipulated like other variables
* The only built-in operations on a struct are the assignment and the member access operations
* Neither arithmetic nor relational operations are allowed on structs
* As a parameter to a function, a struct can be passed wither by reference or by value
* A function CAN return a value of type struct
* A struct can be a member of another struct

10.

* A class is a collection of a fixed number of components. Components of a class are called the MEMBERS of a class, and they are accessed by name.
* Members of a class are classified into one of three categories: PRIVATE, PROTECTED, AND PUBLIC
* Private members of a class are not directly accessible outside of the class
* Public members of a class are directly accessible outside of a class
* By default, all members of a class are private
* A member of a class can be a FUNCTION OR a VARIABLE
* If any member of a class is a function, you usually use the function prototype to declare it
* You cannot initialize a variable when you declare it within the definition of the class.
* A member function of a class is called a constant function if it’s heading contains the word CONST at the end. A constant member function of a class cannot modify the member variables of the class
* In UML diagram of a class, the top box has the name of the class, the middle has the member variables and their data types, and the last box has the member function name, parameter list, and the return type of the function. A + sign in front of a member indicates that it’s public. A – sign indicates private. A # indicates a protected member
* A class is a definition. No memory is allocated for the class itself, memory is allocated for the class variable when you declare them
* Class variables are called CLASS OBJECTS, or CLASS INSTANCES, or simply OBJECTS
* A class member is accessed using the class variable name and the (.) operator, followed by the member name
* The only built-in operations on classes are the assignment and member selection
* As parameters to functions, classes can be passed by value or references
* A function can return a value of type class.
* Any program (or software) that uses a class is considered a “client” of the class.
* A member function of a class that MODIFIES the value of the member variables is called a MUTATOR FUNCTION
* A member function of a class that only accesses the values of the member variables is called a ACCESSOR function
* A constant member function of a class can only call the other constant member functions of the class
* CONSTRUCTORS guarantee that the member variables are initialized when an object is declared
* The NAME of a CONSTRUCTOR is the same as the name of the class
* A class can have more than one constructor, but only one destructor
* A constructor without parameters is a default constructor.
* Constructors automatically execute when a class object enters its scope
* Destructors automatically execute when a class object goes out of scope, and it has no parameters
* Constructors and destructors are functions without any type; they are neither value-returning nor void. As a result they cannot be called like other functions
* A data type that separates the logical properties from the implementation details is called an abstract data type. (ADT)
* Classes were specifically designed in C++ to handles ADT’s
* To implement an ADT you must represent the data and write the related algorithms to implement the operations
* A precondition is a statement specifying the conditions that must be true before the function is called
* A postcondition specifies what is true after the function call is completed
* A PUBLIC STATIC member, function or variable, of a class can be accessed using the class name and the scope resolution operator ::
* For each static variable of a class C++ allocates only one memory space. All objects of the class refer to the same memory space
* STATIC member variables of a class exist even when no object of the class type exists
* Non-static member variables of a class are called the instance variables of the class

12. POINTERS, CLASSES, VIRTUAL FUNCTIONS AND ABSTRACT CLASSES

* Pointer variables contain the addresses of other variables as their values
* No name is associated with the pointer data type
* A pointer variable is declared using the \* asterisk between the data type and the variable. For example int \* p; char \* ch; declare p and ch to be pointer variables. He value of p points to a memory space of type int, and the value of ch points to a memory space of type char.
* & is called the address of operator. It returns the address of its operand. For example: p = &num; sets the value of p to the address of num.
* When used as a unary operator, \* is called the dereferencing operator.
* An interesting property of pointers is that they can be used to access the variable they point to directly. This is done by preceding the pointer name with the *dereference operator* (\*). The operator itself can be read as "value pointed to by".
* It is important to clearly differentiate that foo refers to the value 1776, while \*foo (with an asterisk \* preceding the identifier) refers to the value stored at address 1776, which in this case is 25. Notice the difference of including or not including the *dereference operator* (I have added an explanatory comment of how each of these two expressions could be read):
* The memory location indicated by the value of a pointer variable is accessed by using the dereferencing operator. EX: if p is a pointer, \*p = 25; sets the value of the memory location indicated by the value of p to 25.
* You can use the member access operator arrow -> to access the component of an object pointed to by a pointer
* Pointer variables are initialized using either 0 (the integer zero), NULL, or the address of a variable of the same type
* The only number that can be directly assigned to a pointer is 0
* The only arithmetic operations allowed on pointers are increment ++ and decrement –, addition of an integer to a pointer variable, subtraction of an integer from a pointer variable, and subtraction of a pointer from another pointer
* Pointer ARITHMETIC is DIFFERENT than normal math. When an integer is added to a pointer, the value added to the value of the pointer variable is the integer times the size of the object to which the pointer is pointing. Similarly, when an int is subtracted from a pointer, the value subtracted from the value of the pointer variable is the int times the size of the object to which the pointer is pointing.
* Pointer variables can be compared using relational operators
* The value of one pointer variable can be assigned to another pointer variable of the same type
* A variable created during program execution is called a dynamic variable
* The operator NEW is used to create a dynamic variable
* The operator DELETE is used to deallocate the memory occupied by the dynamic variable
* The operator NEW has TWO FORMS: one to create a single dynamic variable, and one to create an array of dynamic variables
* If p is a pointer of type int: p = new int; allocates storage of type int somewhere in memory and stores the address of the allocated storage in p.
* The operator DELETE has TWO FORMS: one to deallocate memory occupied by a single dynamic variable, and one to deallocate the memory occupied by an array of dynamic variables (delete p) or (delete [] p)
* The array name is a constant pointer. It always points to the same memory location, which is the location of the first array component
* To CREATE a DYNAMIC ARRAY: (if p is a pointer of type int) p = new int [10]; creates an array of 10 components of type int. The base address of the array is stored in p. we call p a dynamic array.
* Array notation can be used to access the components of a dynamic array.
* An array created during program execution is called a DYNAMIC ARRAY
* C++ allows a program to create dynamic multidimensional arrays.
* Ex: int \*\*board; the variable board is a pointer to a pointer.
* In a SHALLOW COPY, two or more pointers of the same type point to the same memory space (meaning they point to the same data).
* In a DEEP COPY, two or more pointers of the same type have their own copies of the data.
* If a class has a destructor, the destructor is automatically executed whenever a class object goes out of scope.
* If a class has a pointer member variable, the built-in assignment operators provide a shallow copy of the data
* A copy constructor executes when an object is declared and initialized using the value of another object, and when an object is passed by value as a parameter
* C++ allows a user to pass an object of a derived class to a formal parameter of the base class type
* The address of operator can be used to return the address of a private member variable of a class

15. RECURSION

* The process of solving a problem by reducing it to smaller versions of itself is called RECURSION
* Every recursive definition has ONE or MORE BASE CASES
* The solution of the problem in a BASE CASE is OBTAINED DIRECTLY
* A function is called recursive if it calls itself
* Recursive algorithms are implemented using recursive functions
* Every recursive function MUST HAVE one or more base cases
* The GENERAL CASE must eventually be reduced to a BASE CASE
* The BASE case STOPS the RECURSION
* Logically you can think of a recursive function as having an unlimited number of copies of itself. Every recursive call has its own code and its own set of parameters and local variables. After completing a particular recursive call, control goes back to the calling environment (which is the PREVIOUS call). The CURRENT (recursive) call must execute completely before control goes back to the previous call. The execution in the previous call begins from the point immediately following the recursive call.
* A function is called DIRECTLY RECURSIVE if it calls itself.
* A function that calls another function and eventually results in the original function being called is said to be INDIRECTLY RECURSIVE
* A recursive function in which the last statement executed is the recursive call is called a TAIL RECURSIVE FUNCTION
* To design a recursive function: 1) understand the problem requirements, 2) determine the limiting conditions—ie. For a list, the limiting condition is the number of elements in the list, 3) identify the base cases and provide directly solutions to each base case. 4) identify the general cases and provide a solution to each general case in terms of smaller versions of itself

17. LINKED LISTS

* A LINKED LIST is a list of items, called NODES, in which the order of the nodes is determined by the address, called a link, stored in each node
* The pointer to a linked list—that is, the pointer to the first node in the list—is stored in a separate location called the FIRST or the HEAD
* A linked list is a DYNAMIC DATA STRUCTURE
* The length of a linked list is the number of nodes in the list
* Item insertion and deletion from a linked list do not require data movement, only the pointers are adjusted
* A (single) linked list is traversed in only one direction
* The search on a linked list is sequential
* The first (or head) pointer of a linked list is always fixed, pointing to the first node in the list
* To traverse a linked list, the program must use a pointer different than the head pointer of the list, initialized to the first node in the list
* In a DOUBLY linked list, every node has two links: one points to the next node, and one points to the previous node
* A doubly linked list can be traversed in either direction
* In a doubly linked list, item insertion and deletion require the adjustment of two pointers in a node
* A linked list in which the last node points to the first node is called a CIRCULAR LINKED LIST

NOTES

void insert(node \*& head, int position, int newInt)

//it passes the head by reference. You can pass parameter by reference in C++ including pointers. It means the head could be modified by insert.

You would want to pass a pointer by reference if you have a need to modify the pointer rather than the object that the pointer is pointing to.

This is similar to why double pointers are used; using a reference to a pointer is slightly safer than using pointers.

void fn(int nArg)

{

nArg = 10;

// value of nArg at this point is 10

}

void parent(void)

{

int n1 = 0;

fn(n1);

// value of n1 at this point is still 0

}

Here the parent() function initializes the integer variable n1to 0. The value of n1 is then passed to fn(). Upon entering the function, nArg is equal to 0, the value passed. fn()changes the value of nArg to 10 before returning to parent(). Upon returning to parent(), the value of n1 is still 0.

The reason for this behavior is that C++ doesn’t pass a variable to a function. Instead, C++ passes the value contained in the variable at the time of the call. That is, the expression is evaluated, even if it is just a variable name, and the result is passed.

In the example, the value of n1, which is 0, was passed to fn(). What the function does with that value has no effect on n1.

PASSING POINTER VALUES IN C++

Like any other intrinsic type, a pointer may be passed as an argument to a function:

void fn(int\* pnArg)

{

\*pnArg = 10;

}

void parent(void)

{

int n = 0;

fn(&n); // this passes the address of i

// now the value of n is 10

}

In this case, the address of n is passed to the function fn()rather than the value of n. The significance of this difference is apparent when you consider the assignment within fn().

Suppose n is located at address 0x100. Rather than the value 10, the call fn(&n) passes the value 0x100. Within fn(), the assignment \*pnArg = 10 stores the value 10 in the int variable located at location 0x100, thereby overwriting the value 0. Upon returning to parent(), the value of n is 10 because n is just another name for 0x100.

PASSING BY REFERENCE IN C++

C++ provides a shorthand for passing arguments by address — a shorthand that enables you to avoid having to hassle with pointers. The following declaration creates a variable n1 and a second reference to the same n1 but with a new name, nRef:

int n1; // declare an int variable

int& nRef = n1; // declare a second reference to n1

nRef = 1; // now accessing the reference

// has the same effect as accessing n1;

// n1 is now equal to 1

A reference variable like nRef must be initialized when it is declared because every subsequent time that its name is used, C++ will assume that you mean the variable that nRef refers to.

Reference variables find their primary application in function calls:

void fn(int& rnArg)// declare reference argument

{

rnArg = 10; // change the value of the variable...

} //...that rnArg refers to

void parent(void)

{

int n1 = 0;

fn(n1); // pass a reference to n1

// here the value of n1 is 10

}

This is called *passing by reference.* The declaration int& rnArg declares rnArg to be a reference to an integer argument. The fn() function stores the value 10 into the intlocation referenced by rnArg.

Passing by reference is the same as passing the address of a variable. The reference syntax puts the onus on C++ to apply the “address of” operator to the reference rather than requiring the programmer to do so.

You cannot overload a pass by value function with its pass by reference equivalent. Thus, you could not define the two functions fn(int) and fn(int&) in the same program. C++ would not know which one to call.

* In a function call you specify ONLY the actual parameters, not its data type