CS 280
Programming Language Concepts

Spring 2025

Functions, Pointers & Arrays

Topics

- Introduction
- Overview of Short Assignment 2
- Functions
- Recursion
- Arrays
- Command Line Arguments

Introduction

- Objective of the SA 2:
  - See the posted problem statement on Canvas.
  - Writing a program for a simple counting tool for collecting information from textual files of documents prepared for a simple word processing software.

- Review of C++ features (continuation of Lecture 3 of this week)
  - Functions: definitions, and parameter passing
  - Arrays: definition, using arrays as function parameters, and using pointer variables to access arrays.
  - Using command line arguments for passing input to programs.

Short Assignment 2: Example

- Overview of the SA 2
  - The program reads lines from a file until the end of file.
  - A commented line is recognized by the sequence of characters “//” at the beginning of the line.
  - A command line is recognized by the sequence of characters ‘>>’ at the beginning of a line, followed by a command name.
  - Commands for formatting textual documents are “bold”, “italic”, “underline”, and “regular”.
  - Only one command name is considered in a command line.
  - A short form of the commands’ names can be recognized by the first two characters of the command name.
  - Command names or their short forms are case insensitive.
  - Error message is displayed if the command line prompt is not followed by a command name, or their short forms.

Short Assignment 2: Example

Line number	File contents
1	3456 george 10.25
2	>>bold
3	
4	>>
5	>>>regular
6	// 4321 staci 12.7
7	>> ITALIC
8	67899 smith 9643.45
9	
10	
11	>>Reg
12	>> IT
13	>> Bold
14	278 hello -
15	>> BO
16	+654
17 → End of File marker	

Short Assignment 2: Example (Cont'd)

- Given the shown input file contents, the generated results are as follows:
  - See the posted testing case file for further examples.

Entered file name →

Screen pointer →

```
Enter the file name to read from:
infile6
Error: No command name following a command prompt in line 4
Error: Unrecognizable command name ">regular" in line 5
Error: Unrecognizable command name "reg" in line 11

Total lines: 17
Commented lines: 1
Command lines: 5
Bold commands: 3
Italic commands: 2
Regular commands: 0
Underline commands: 0
```

Example Output Results on Vocareum Linux bash Terminal

- Output results for Case 6 with mismatching errors highlighted
  - Note: check is case insensitive.

```
RUNNING case6
Enter the name of a file to read from:

Incorrect command name at line: 4

Incorrect command name at line: 5

Incorrect command name at line: 7

Incorrect command name at line: 11

Total Lines: 12
Comented lines: 2
Number of Paragraphs: 0
Total Number of Words: 13
Bold commands: 1
Italic commands: 0
Underline commands: 0
*** /tmp/tmp.HeVrDTqjAa 2024-09-07 07:42:29.255401926 -0700
--- case6.correct        2024-09-04 09:27:44.713000000 -0700
***************
*** 9,15
  Incorrect command name at line: 11

  Total Lines: 12
! Comented lines: 2
  Number of Paragraphs: 0
  Total Number of Words: 13
  Bold commands: 1
--- 9,15 ----
  Incorrect command name at line: 11

  Total lines: 12
! Commented lines: 2
  Number of Paragraphs: 0
  Total Number of Words: 13
  Bold commands: 1
RESULT: case6 [  0  / 1 ]
```

Your output file

Case6.correct

This line should be changed to the corresponding line of the case6.correct file

C++ Functions

- Function declaration introduces an identifier that names a function and specifies the following properties (protocol):
  - number of parameters
  - type of each parameter
  - type of return value (if there is a one)
  - Basic syntax for function declaration:

```
Return_type function_name(parameter declarations);
bool isOdd(int x);//function prototype
```

- Basic syntax for function definition:

```
Return_type function_name(parameter declarations){
   statements
}
bool isOdd(int x) { // declare and define isOdd
return x % 2;
}
```

C++ Functions

- A *function definition* describes the interface to, and the actions of the subprogram abstraction
  - Function declarations in C and C++ are often called *prototypes*
    - A *subprogram declaration* provides the protocol, but not the body, of the subprogram
- A *function call* is an explicit request that the subprogram be executed.
- Actual/Formal Parameter Correspondence
  - The binding of actual parameters to formal parameters is by position: the first actual parameter is bound to the first formal parameter and so forth.
- Parameter Passing Methods in C++
  - Pass-by-value: Normally implemented by copying
  - Pass-by-reference is achieved by using pointers as parameters or using special pointer type called reference type

C++ Functions

- **return** statement used to exit function, passing specified return value (if any) back to caller
- Code in function executes until return statement is reached or execution falls off end of function
  - if a function returns a type, a return statement takes single parameter indicating the value to be returned.
  - if a function return type is void, the function does not return any value and return statement takes no parameter.
  - Falling off end of function is equivalent to executing return statement with no value.

C++ Functions (Continued)

- **Examples**

```
bool isOdd(int x) { // declare and define isOdd
return x % 2;
}

void increment(int& x) {// x is passed by reference
  ++x;
}
double square(double x){// x is passed by value
   return x * x;
}
void increment(int * x) {
// x is passed by reference using a pointer parameter
  ++(*x);
}
```

C++ Functions (Continued)

- Function overloading: multiple functions can have same name as long as they differ in number/type of their arguments
- Example:

```
void print(int x) {
   std:: cout << "int has value " << x << '\n';
}
void print(double x) {
   std:: cout << "double has value " << x << '\n';
}
void demo () {
  int i = 5;
  double d = 1.414;
  print(i); // calls print(int)
  print(d); // calls print(double)
}
```

Pointers

- A pointer is a variable that contains a memory address.
- Pointers must be explicitly declared
  - In C/C++:
    - `int *ip; //declares ip as a pointer to an int`
    - `Obj *op; //declares op as a pointer to an Obj`
- Pointers need to be initialized: they must "point to" something:
  - Dereferencing (using the * operator on) a pointer that has not been initialized is an error.

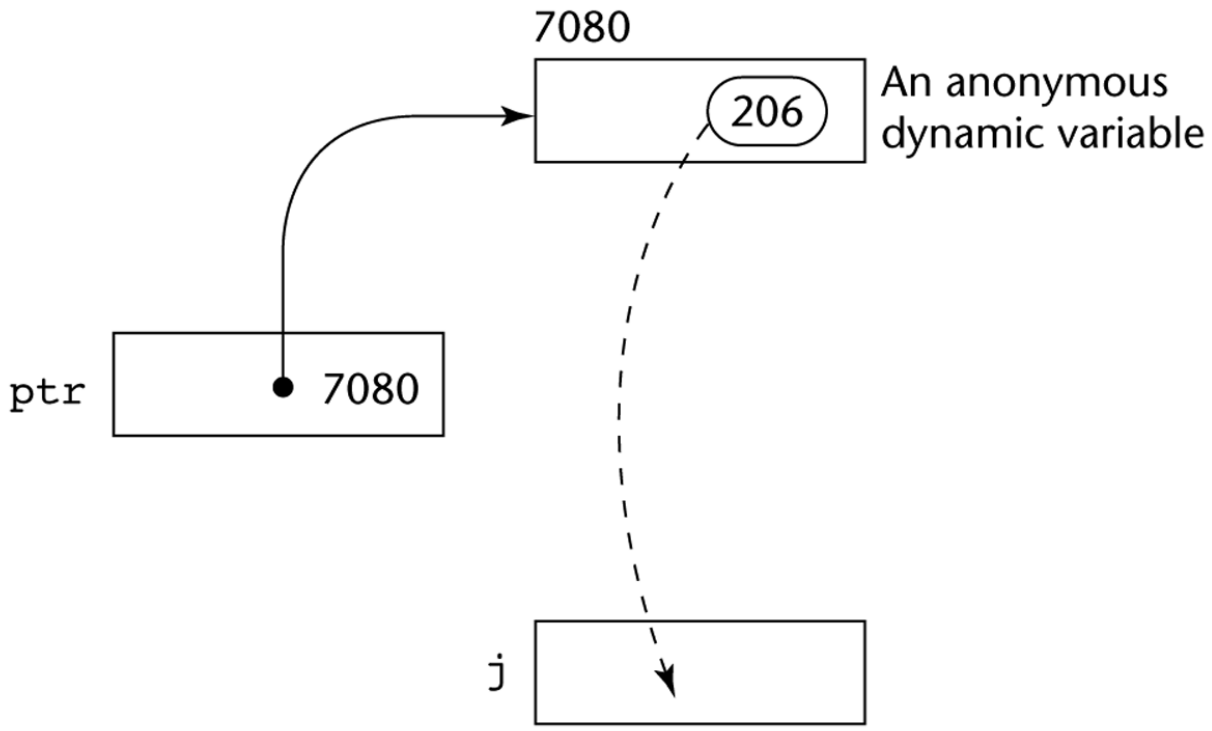
Pointer Operations

- Two fundamental operations: assignment and dereferencing
- Assignment is used to set a pointer variable's value to some useful address.
- Dereferencing yields the value stored at the location represented by the pointer's value.
  - Dereferencing can be explicit or implicit
  - C++ uses an explicit operation via *

    `j = *ptr`

    sets j to the value located at `ptr`

Pointer Assignment Illustrated



The assignment operation j = *ptr

References

- References in C++ must be explicitly declared as: "`int &x`"
  - It is a reference to an integer; the name of the reference is x. It holds address of x.
  - Similar to a constant pointer
  - No dereferencing applied by the * operator
- References in C++ must be initialized when declared:

```
int x;
int& xr = x;
// without the initialization? compile error
```

  - Note: If a parameter to a function is a reference, then it's initialized, at the time that the function is called, to refer to the variable that is passed to the function

Examples

```
int x,y,z; // integers
int& xr = x; // reference to the int x
int *yp; // a pointer to an int
yp = &y; // initialize pointer to point to y
xr = 10; // sets x (what xr refers to) to 10
y = 20;
z = *yp; // set z to the value of the int
         // variable that yp points to
*yp = xr; // set the int variable that yp
          // points to equal to the value of x
          // (what xr refers to)
```

Arrays

- An array is a group of elements of the same type in contiguous memory locations.
  - In C/C++:

```
int x[10];//an array of 10 integers allocated to x
// x[i] element is an integer.
// x is an expression of type const pointer (int*),
// whose value is &x[0] that cannot be changed
```

  - In C++ you can also use pointer variable to point to an array created dynamically:

```
int *y;
y = new int[10];//or
```

    - You can assign y the address of an array variable `y = x;//or y = &x[0];`
  - C/C++ array initialization in declaration statement:

```
int x[] = {1,2,3,4};
```

Arrays

- Accessing a member of an array `x[i]` involves some calculation to find where the ith element of the array is located in memory.
  - x[0] is the first element of the array, x[1] is the second, etc
  - C++ does not check for invalid (**out-of-bounds**) array indexes either at compile time or at run time.
    - **Out-of-bounds array index:** An index value that is either less than 0 or greater than or equal the array size.
- In C/C++
  - Arrays do not behave like first-class objects.
  - Cannot be copied by =.
  - Does not remember how many items it can store.
  - Does not check for index validity.

Arrays

- Passing Arrays As Arguments
  - Arrays are not copied to functions; instead an **array variable** (a const pointer as in C/C++), **a pointer to an array** (in C/C++ as long you declare it that way), **or a reference** (in C++ if you declare it that way), is passed.
    - C++ doesn't allow arrays to be passed by value; arrays are *always* passed by reference.
    - When an array is passed as an argument, its **base address**—the memory address of the first element of the array—is sent to the function.
  - **In C/C++ the name of the array is a constant pointer expression; the value of the expression is the address of the first element of the array.**
    - A parameter declared as `int *ap` or `int ap[]` works the same. Note there is no length member.

Arrays

- Example

```
void ZeroOut((int intArray [], int arrLength){
  int i;
  for(i = 0; i < arrLength; i++)
     intArray[i] = 0;
}
```

- The declaration of the function parameter as a pointer works the same in the function definition.

```
void ZeroOut((int *intPtr, int arrLength);
```

- Using the reserved word **const** in the declaration of a parameter, that is passed by reference, prevents a function from modifying the caller's argument.

```
void ZeroOut((const int *intPtr, int arrLength);
```

- C-Style string is an array of characters that ends with the *null character* ('\0')

```
char myString [] = "Hello"; //it is the same as
char myString [] = {'H', 'e', 'l', 'l', 'o', '\0'};//or
char * myString = "Hello";
```

  - we can convert the C++ string into a C-style string by using the **c_str()** function.

The main Function

Entry point to a program is always the function called main that has return type of **int.** It can be declared to take either no arguments or two arguments as follows

```
int main();
int main(int argc, char* argv[]);
```

- Command Line Arguments
  - When a program executes, the runtime environment may pass parameters to it.
  - These parameters are strings specified on the command line when the program is run.
  - They appear in the program as an array of strings passed to main function.

The main Function

- C/C++ command line arguments:

```
int main(int argc, char *argv[ ])
```

  - `argc` number of **C-style strings** provided to program, representing the number of command line arguments
  - `argc` will be 1 if no command line arguments are passed (only one entry in the array: the name of the program)
  - `argv` array of pointers to C-style strings, it is an array ([ ]) of pointers (*) to char.
  - `argv[0]` is the name of the program.
  - `argv[argc]` is guaranteed to be 0 (i.e., null pointer)

The main Function

- What can I do with a command line arg?
  - Use it as a C-string.
  - Convert it to a std::string

```
std::string progname(argv[0]);
```

- Remember that you MUST check argc to make sure that a particular argument has been passed to your program. If argc == 4, then there are values in argv[0], argv[1], argv[2] and argv[3]. Going outside those bounds is a mistake.
- Return value of main:
  - typically passed back to operating system.
  - can also use function `void exit(int)` to terminate program, passing integer return value back to operating system.
  - return statement in main is optional.

Recursive Solutions

- A recursive function calls itself.
- Each recursive call solves an identical, but smaller, problem.
- Test for base case enables recursive calls to stop.
- Eventually, one of smaller problems must be the base case.

Questions for constructing recursive solutions

1. How to define the problem in terms of smaller problems of same type?
2. How does each recursive call diminish the size of the problem?
3. What instance of problem can serve as base case?
4. As problem size diminishes, will you reach base case?

Finding a Recursive Solutions

- A recursive solution to a problem must be written carefully
- The idea is for each successive recursive call to bring you one step closer to a situation in which the problem can easily be solved
- This easily solved situation is called the base case
- Each recursive algorithm must have at least one base case, as well as a general (recursive) case

General format for Many Recursive Functions

```
if   (some easily-solved condition)     // Base case

     solution statement

else                                    // General case

     recursive function call
```

Some examples . . .

Writing a Recursive Function to Find the Sum of the Numbers from 1 to n

DISCUSSION

The function call Summation(4) should have value 10, because that is 1 + 2 + 3 + 4

For an easily-solved situation, the sum of the numbers from 1 to 1 is certainly just 1

So our base case could be along the lines of

```
if  (n == 1)
        return 1;
```

Writing a Recursive Function to Find the Sum of the Numbers from 1 to n

Now for the general case. . .

The sum of the numbers from 1 to n, that is,
1 + 2 + . . . + n can be written as

n + the sum of the numbers from 1 to (n - 1),
that is, n + 1 + 2 + . . . + (n - 1)

or, n + Summation(n - 1)

And notice that the recursive call Summation(n - 1) gets us "closer" to the base case of Summation(1)

Finding the Sum of the Numbers from 1 to n

```
int   Summation (/* in */  int   n)
{
    if  (n == 1)      // Base case
        return  1;
    else              // General case
        return (n + Summation (n - 1));
}
```

Summation(4) Trace of Call