Assignment 5
C/C++ Programming II

C2A5 General Information

Assignment 5 consists of FOUR (4) exercises:

C2A5E1 C2A5E2 C2A5E3 C2A5E4

All requirements are in this document.

C2A5 General Information, continued

--- No General Information for This Assignment---

Get a Consolidated Assignment 5 Report (optional)

If you would like to receive a consolidated report containing the results of the most recent version of each exercise submitted for this assignment:

`Send an empty-body email to the assignment checker with the subject line **C2A5_164440_U09339367** and no attachments.

Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report requests as many times as you wish before the assignment deadline.

C2A5E1 (4 points – C Program)

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C2A5E1_SwapObjects.c**. Also add instructor-supplied source code file **C2A5E1_main-Driver.c**. Do not write a `main` function! `main` already exists in the instructor-supplied file and it will use the code you write.

File **C2A5E1_SwapObjects.c** must contain a function named `SwapObjects`.
`SwapObjects` syntax:

```
    void SwapObjects(void *pa, void *pb, size_t size);
```

Parameters:
    `pa` – a pointer to one of the objects to be swapped
    `pb` – a pointer to the other object to be swapped
    `size` – the number of bytes in each object
Synopsis:
    Swaps the objects in `pa` and `pb`.
Return:
    void

- Do not use any kind of looping statement.
- Do not call any function that is not from the standard C library.

If `SwapObjects` dynamically allocates memory it must also free it before returning.

All dynamic allocation results must be tested for success/failure before the memory is used. If allocation fails an error message is output to `stderr` and the program is terminated with an error code.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A5E1_164440_U09339367** and with both source code files attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

Hints:
1. Merely swapping pointers `pa` and `pb` does not swap the objects to which they point.
2. The only case where dynamically-allocated memory is freed automatically is when a program exits. Good programming practice dictates that dynamically-allocated memory always be explicitly freed by the program code as soon as it is no longer needed. Relying upon a program exit to free it is a bad programming practice.

C2A5E2 (6 points – C Program)

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C2A5E2_Create2D.c**. Also add instructor-supplied source code files **C2A5E2_Type-Driver.h** and **C2A5E2_main-Driver.c**. Do not write a `main` function! `main` already exists in the instructor-supplied implementation file and it will use the code you write.

Regarding data type `Type`, which is used in this exercise…
`Type` is a typedef'd data type that is defined in instructor-supplied header file
C2A5E2_Type-Driver.h
Any file that uses this data type must include this header file using `#include`.

File **C2A5E2_Create2D.c** must contain functions named `Create2D` and `Free2D`.

`Create2D` syntax:
```
Type **Create2D(size_t rows, size_t cols);
```
Parameters:
`rows` – the number of rows in the 2-dimensional pointer array `Create2D` will create
`cols` – the number of columns in the 2-dimensional pointer array `Create2D` will create
Synopsis:
Creates a 2-dimensional pointer array of data type `Type` having the number of rows and columns specified by `rows` and `cols`. All memory needed for this array is dynamically-allocated at once using a single call to the appropriate memory allocation function. If allocation fails an error message is output to `stderr` and the program is terminated with an error code.
Return:
a pointer to the first pointer in the array

`Free2D` syntax:
```
void Free2D(void *p);
```
Parameters:
`p` – a pointer to the block of memory dynamically-allocated by `Create2D`
Synopsis:
Frees the dynamically-allocated block of memory pointed to by `p`.
Return:
void

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A5E2_164440_U09339367** and with all three source code files attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

Hints:
Make no assumptions about relationships between the sizes of pointers and the sizes of any other types, including other pointer types. If you attempt this exercise without fully understanding every aspect of note 14.4B you are asking for trouble. If you're having problems step through your code on paper to make sure it creates the memory map shown in Figure 1 on the next page.

Although the block of memory allocated by any call to a standard memory allocation function is guaranteed to be internally contiguous, the blocks allocated by multiple calls cannot be assumed to be contiguous with each other. Such is the case with the multiple memory blocks allocated by the original `Create2D` function illustrated in note 14.4B. If these blocks are not contiguous it prevents such arrays from being accessed linearly, which is a significant limitation in some applications. In addition, the fact that the original `Create2D` function must do multiple dynamic memory allocations makes it inefficient and necessitates a custom `Free2D` function. These limitations can be overcome if `Create2D` instead pre-calculates the total amount of memory needed for everything and allocates it all at once. The main disadvantage of this approach is that data alignment problems are possible when multiple data types are mixed. Although this potential issue can be solved with some added complexity, simply ignore it for this exercise.

Your version of `Create2D` must create a pointer array like the original version except that it must get all needed memory at once. Figure 1, below is a memory map of how the result should look for a 2-by-3 array after your version completes. Compare this with the memory map in Figure 2, produced by the original `Create2D` function. Notice that both employ the same basic concepts but the new version places everything in one contiguous block of memory rather than in multiple, possibly non-contiguous blocks:

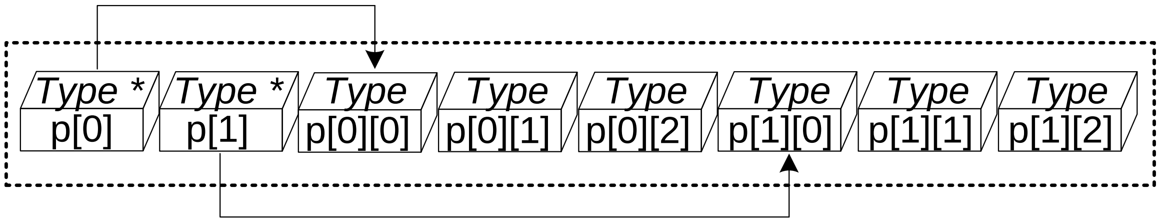


Figure 1 – Memory Map from Your Rewritten `Create2D` Function for a 2x3 Array

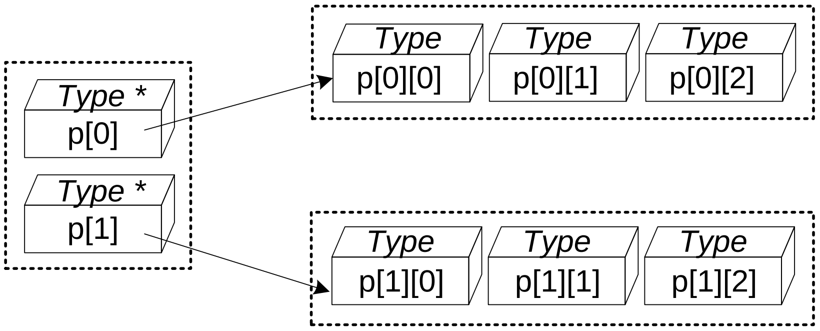


Figure 2 – Memory Map from the Original `Create2D` Function for a 2x3 Array

As in the original version, you must explicitly initialize each pointer to point to the first element of the corresponding sub-array. Your code (but not my driver file code) must work with any arbitrary data type represented by `Type`.

C2A5E3 (5 points – Diagram only – No program required)

Create the state diagram described in this exercise in **8.5"x11" portrait orientation** and put it in a PDF file named **C2A5E3_StateDiagram.pdf**. Using an application such as Word, Visio, etc. to create it is preferred, but a neat hand-drawn diagram is also acceptable. Regardless of how the diagram is created, **significant credit will be deducted** if I consider it to be sloppy or hard to read.

Your state diagram must analyze the contents of an arbitrary string of characters to determine if its syntax is that of a "decimal floating literal" and if so, its type. A formal syntax definition is provided later but a few examples that do and do not conform to that definition are provided below for your consideration. One thing that often surprises students is that any expression starting with a plus or minus sign is never a numeric literal of any kind.

YES:	1.2	1.2e0	12e+5	1E-1	0.0	0e0	5E5	.02E08	6e6f	6e6L	6.F	6.e-25F
NO:	+1.2	-1.2e0	1.2e+	1E-.1	00	+0e0	535	.e08	-6e6f	6e6+L	6F	6.e-2.5F

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A5E3_164440_U09339367** and with your PDF file attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

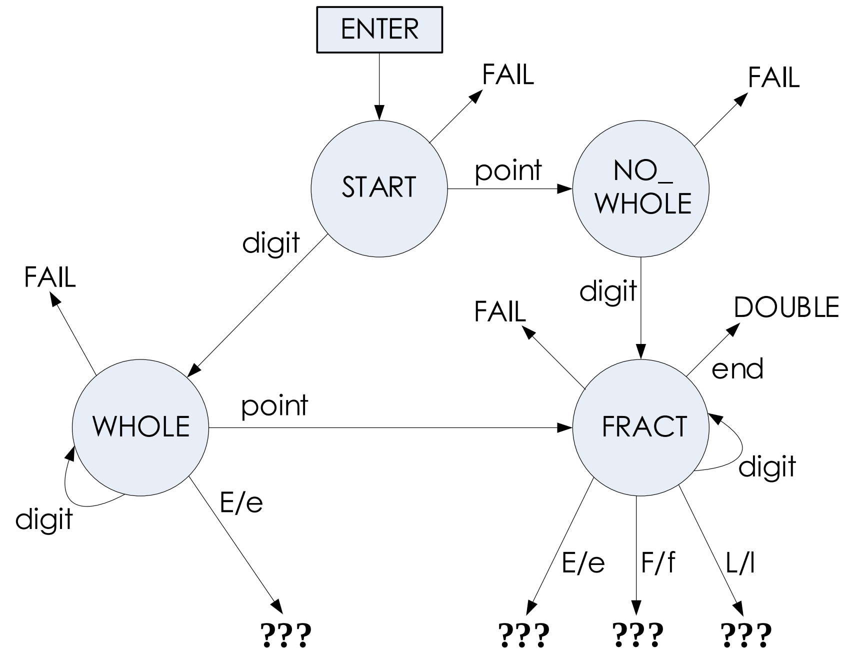
Continued on the next page…

The first part of the required state diagram is provided below and must be present in your finished diagram, with the ??? transitions at the bottom connected to what you add. A state machine is often implemented as a separate function and in this diagram the word **ENTER** represents the function's entry point. Transitions to the following four identifiers represent returns from the function and have the indicated meanings:

- **FAIL** – string did not represent a decimal floating literal.
- **FLOAT** – string represented a type **float** decimal floating literal.
- **DOUBLE** – string represented a type **double** decimal floating literal.
- **L_DOUBLE** – string represented a type **long double** decimal floating literal.

Additional Requirements:

1. A correct diagram has exactly 9 states. Function entry points and returns do not count as states.
2. State names must meaningfully indicate the current status of the string parse and must be legal identifiers that you will use unaltered in your state machine code in the next exercise.
3. Use identifiers **FAIL**, **FLOAT**, **DOUBLE**, and **L_DOUBLE** as return indicators only – never as state names.
4. The string's next character is available upon entering each state and all transitions out of that state must be based solely upon that character. Looking back or looking ahead is prohibited.
5. More than one default transition out of any state makes no logical sense and is always wrong.
6. Because the state diagram is processing a string and not a file, do not label anything as **EOF**.
7. Do not attempt to determine the numeric value of a floating literal.



Continued on the next page…

Decimal Floating Literal Syntax

The language standards use a variant of Backus-Naur Form (BNF) notation to describe the syntax of various constructs, and that notation is also used in the following table to describe the syntax of a decimal floating literal. This table is useful for creating character sequences that are decimal floating literals as well as determining if existing character sequences are such literals. Any item having an $_{opt}$ subscript is optional and inter-item spacing is for table readability only and is never actually present in any numeric literal.

decimal-floating-literal: fractional-constant exponent-part$_{opt}$ floating-suffix$_{opt}$ digit-sequence exponent-part floating-suffix$_{opt}$
fractional-constant: digit-sequence$_{opt}$ **.** digit-sequence digit-sequence **.**
exponent-part: **e** sign$_{opt}$ digit-sequence **E** sign$_{opt}$ digit-sequence
sign: one of **+ -**
digit-sequence: digit digit-sequence digit *(this means 2 or more digits)*
digit: one of **0 1 2 3 4 5 6 7 8 9**
floating-suffix: one of **f l F L** *(**f**/**F** => **float**, **l**/**L** => **long double**, no suffix => **double**)*

The following describes how to use this table to determine the various character combinations that represent a decimal floating literal:

1. Look in the table for *decimal-floating-literal* followed by a colon. The two indented lines below it are its two possible syntaxes and the goal is to determine which one, if any, matches all input characters with none left over. The first one, which starts with *fractional-constant*, is examined in the next steps.
2. Look for *fractional-constant* followed by a colon. Below it are its two possible syntaxes. The first can begin with a *digit-sequence* or a *period* but the second must begin with a *digit-sequence*.
3. Look for *digit-sequence* followed by a colon, then look for *digit* followed by a colon. This then tells us that a *digit-sequence* must consist of one or more digits from the set **0**-**9**.
4. The second thing the first syntax in step 1 indicates is that after the *fractional-constant* there can optionally be an *exponent-part*.
5. Look for *exponent-part* followed by a colon. Below it are its two possible syntaxes. Follow them through the table to determine which characters they can represent.
6. The last thing the first syntax in step 1 indicates is that a *decimal-floating-literal* can optionally end with a *floating-suffix*.
7. Look in the table for *floating-suffix* followed by a colon. This tells us that a *floating-suffix* must consist of an uppercase or lowercase letter **F** or letter **L**.
8. Continue this process for all parts of all syntaxes until you determine all possible character combinations that can form a *decimal-floating-literal*. Do any of the combinations match the entire input character sequence you are testing?

C2A5E4 (5 points – C++ Program)

Exclude any existing source code files that may already be in your IDE project and add two new ones, naming them **C2A5E4_OpenFile.cpp** and **C2A5E4_DetectFloats.cpp**. Also add instructor-supplied source code files **C2A5E4_StatusCode-Driver.h** and **C2A5E4_main-Driver.cpp**. Do not write a `main` function! `main` already exists in the instructor-supplied implementation file and it will use the code you write.

`StatusCode` is an enumeration type consisting of members `FAIL`, `FLOAT`, `DOUBLE`, and `L_DOUBLE` and is defined in instructor-supplied header file
    C2A5E4_StatusCode-Driver.h
Any file that uses this enumeration type must include this header file using `#include`.

File **C2A5E4_OpenFile.cpp** must contain a function named `OpenFile`.
`OpenFile` syntax:
```
    void OpenFile(const char *fileName, ifstream &inFile);
```
Parameters:
    `fileName` – a pointer to the name of a file to be opened
    `inFile` – a reference to the `ifstream` object to be used to open the file
Synopsis:
    Opens the file named in `fileName` in the read-only text mode using the `inFile` object. If the open fails an error message is output to `cerr` and the program is terminated with an error exit code. The error message must mention the name of the failing file.
Return:
    void if the open succeeds; otherwise, the function does not return.

File **C2A5E4_DetectFloats.cpp** must contain a function named `DetectFloats`.
DetectFloats syntax:
```
    StatusCode DetectFloats(const char *chPtr);
```
Parameters:
    `chPtr` – a pointer to the first character of a string to be analyzed
Synopsis:
    Analyzes the string in `chPtr` and determines if it represents a syntactically legal decimal floating literal, and if so, its type (but not its value).
Return:
    one of the following `StatusCode` enumerations representing the result of the string analysis:
        `FAIL`, `FLOAT`, `DOUBLE`, or `L_DOUBLE`

***** IMPORTANT *****
Significant credit will be deducted if the directions below are not followed, even if your code produces the correct results. The **DetectFloats** function:
1. must exactly implement every state and transition in the previous exercise's state diagram.
2. must only contain one loop statement, and it must contain a `switch` statement with a `case` for each diagrammed state. Those `case` names must exactly match the diagram's state names. (Code within each `case` may use `switch` statements and/or `if`/`else` statements, as you prefer.)
3. must make all transition decisions based only upon the current character and the current state. Attempting to determine the length of a string or looking backward/forward at a previous/next character or state is not permitted.
4. must use at least 2 but no more than 3 variables, but only as follows:
    a. required – formal parameter `chPtr`
    b. required – a state variable
    c. OPTIONAL – a variable that indicates the kind of item the current character represents, such as a digit, a radix point, a sign, a suffix, etc. I did not do this, but you are welcome to.

Test your program with instructor-supplied data file **TestFile5.txt**, which must be placed in the program's "working directory". However, do not assume your program works correctly based strictly upon the results with this file. The test strings it contains do not represent all possible character combinations and your program could parse them correctly while still containing one or more significant bugs.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A5E4_164440_U09339367** and with all four source code files attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

Hints:
Each state represents a new character to be examined. When you are ready to return out of the state machine function do not go to another state, but instead immediately return an appropriate status value. Do not use a separate variable to indicate a potential type **float** or type **long double**. Instead, use different states to differentiate these findings.