Assignment 5
C/C++ Programming II

C2A5 General Information

--- 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 email to the assignment checker with the subject line **C2A5_*ID***, where ***ID*** is your 9-character UCSD student ID. 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 or 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 both source code files to the Assignment Checker with the subject line **C2A5E1_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your 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 all three source code files to the Assignment Checker with the subject line **C2A5E2_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your 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:

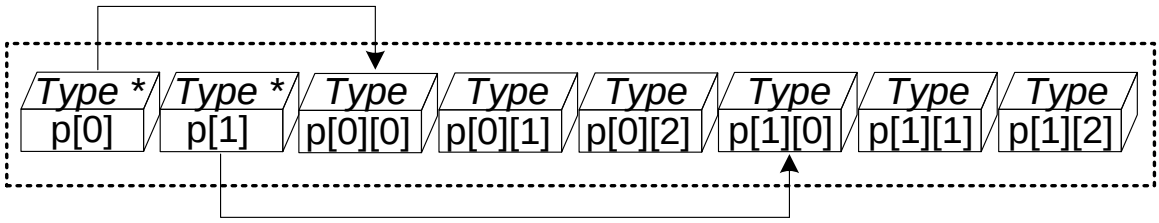


Figure 1 – Memory Map from Your Rewritten Create2D Function for a 2x3 Array

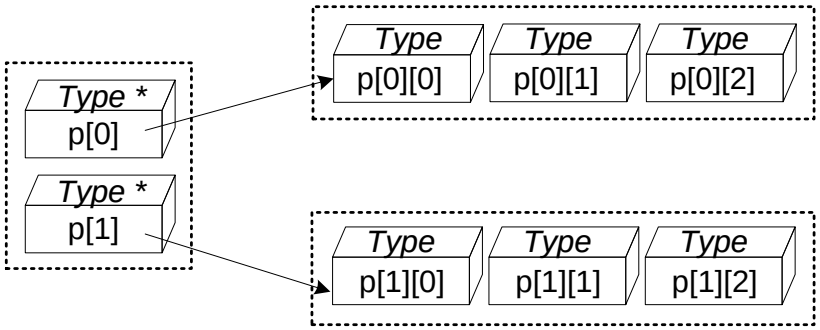


Figure 2 – Memory Map from the Original Create2D Function for a 2x3 Array

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 a state diagram as described below and put it in a PDF file named **C2A5E3_StateDiagram.pdf**. Creating it with a computer application of your choice like Word, Visio, etc. is preferred, but if you choose to draw it by hand please be aware that significant credit will be deducted if I consider it to be sloppy or hard to read.

Your state machine must analyze the contents of a string of arbitrary characters to determine if its syntax is that of a "hexadecimal floating literal" and if so, its data type. Since this is only an exercise in syntax parsing it is not necessary to know how to determine its value. This is equivalent to not needing to know what a "cat" or a "dog" is in order to be able to spell the word correctly.

Submitting your solution

Send your PDF file to the Assignment Checker with the subject line **C2A5E3_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

Requirements and Hints:

A formal definition of the syntax of hexadecimal floating literals is provided later in this document but here are a few sample expressions that are/aren't in case you want to examine them at some point. 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.

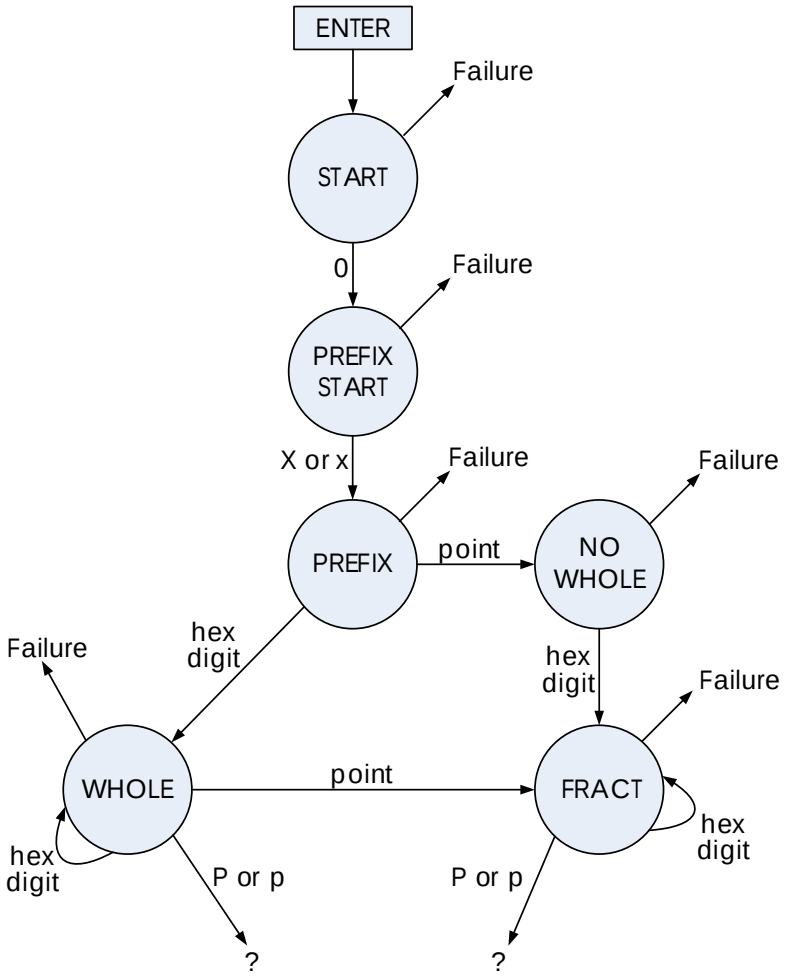
ARE:	0x1.2p0	0x1.2p9F	0x1.Fp89	0xABCp+5	0X1P-1	0xFP-5L	0X.02P08	0x6p6f	0x0p32
AREN'T:	+0x1.2p0	0x1.2pEF	0x1.Fe89	0x1.2	-0x1.2p-1	0x1.2p	0x0.0	0x1P-.1	+0x0p32

(Requirements and Hints are continued on the next page…)

(...Requirements and Hints continued from the previous page)

The first part of the required state diagram is provided below. The two transitions at the bottom must connect to the part you add. Please note the following:

1. Programmatically a state machine is often implemented in its own function, where **ENTER** in the diagram below indicates the function's entry point. Upon completing its task the function returns the result, which must be represented by one of the following in your diagram:
    a. **Failure** – the input string is not a hexadecimal floating literal;
    b. **Success float** – the input string is a type **float** hexadecimal floating literal;
    c. **Success double** – the input string is a type **double** hexadecimal floating literal;
    d. **Success long double** – the input string is a type **long double** hexadecimal floating literal;
2. There are 11 and only 11 total states in a correct diagram.
3. Entry and exit points do not count as states.
4. Choose meaningful state names that can be used directly as identifiers in code.
5. The next character of the string becomes available as each state is entered.
6. Never look back at previous characters or ahead at future characters; base all decisions upon the current character and the current state.
7. Do not label anything as **EOF**; there is no such thing in a string.



(Requirements and Hints are continued on the next page...)

(...Requirements and Hints continued from the previous page)

Hexadecimal Floating Literal Syntax

Since the language standards use a variant of Backus-Naur Form (BNF) notation to describe the syntax of various constructs, it is also used in the table below to describe the syntax of hexadecimal floating literals. Inter-item spacing is for table readability only and is never actually present in any numeric literal.

hexadecimal-floating-literal: hexadecimal-prefix hexadecimal-fractional-constant binary-exponent-part floating-suffix$_{opt}$ hexadecimal-prefix hexadecimal-digit-sequence binary-exponent-part floating-suffix$_{opt}$
hexadecimal-prefix: one of **0x 0X**
hexadecimal-fractional-constant: hexadecimal-digit-sequence$_{opt}$ **.** hexadecimal-digit-sequence hexadecimal-digit-sequence **.**
binary-exponent-part: **p** sign$_{opt}$ digit-sequence **P** sign$_{opt}$ digit-sequence
sign: one of **+ -**
hexadecimal-digit-sequence: hexadecimal-digit hexadecimal-digit-sequence hexadecimal-digit *(this means 2 or more)*
digit-sequence: digit digit-sequence digit *(this means 2 or more)*
floating-suffix: one of **f l F L** *(**f/F** => **float**, **l/L** => **long double**, no suffix => **double**)*
hexadecimal-digit: one of **0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F**
digit: one of **0 1 2 3 4 5 6 7 8 9**

As an example of using the table, let's assume you know nothing about the syntax of a hexadecimal floating literal.

1. Look in the table for *hexadecimal-floating-literal* followed by a colon. The two indented items below it indicate that there are two possible syntaxes, and both start with a *hexadecimal-prefix*.
2. But what if you don't know what a *hexadecimal-prefix* is? Simply look for that term followed by a colon. Below it you are told that it is a **0** followed by either a lowercase **x** or an uppercase **X**. So now you know that every *hexadecimal-floating-literal* must begin with either **0x** or **0X**.
3. The next thing that the first possible syntax for a *hexadecimal-floating-literal* indicates is that after the *hexadecimal-prefix* there must be a *hexadecimal-fractional-constant*.
4. But what if you don't know what a *hexadecimal-fractional-constant* is? Simply look for that term followed by a colon. Below it you are told that there are two possible syntaxes for it.
5. The next thing that the first possible syntax of a *hexadecimal-floating-literal* indicates is that after the *hexadecimal-fractional-constant* there must be a *binary-exponent-part*.
6. But what if you don't know what a *binary-exponent-part* is? Simply look for that term followed by a colon, etc., etc.
7. **Note:** Any item having an $_{opt}$ subscript is optional.

This should give you the idea of how you can proceed through the table to find the syntax of any of the terminology used, all the way down to the definition of what a digit is if you wish.

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.

Regarding data type `StatusCode`, which is used in this exercise…

`StatusCode` is an enumeration type consisting of members `NO_MATCH`, `TYPE_FLOAT`, `TYPE_DOUBLE`, and `TYPE_LDOUBLE`. It 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 hexadecimal floating literal, and if so, its data type (but not its value).

Return:

one of the following `StatusCode` enumerations representing the result of the string analysis:

`NO_MATCH`, `TYPE_FLOAT`, `TYPE_DOUBLE`, or `TYPE_LDOUBLE`

DetectFloats:

1. must exactly implement the state machine you diagrammed in the previous exercise.
2. must make all transition decisions based only upon the current character and the current state; storing or looking backward/forward at a previous/next character or state is not permitted.
3. must use at least 2 but no more than 3 variables only as follows:
   1) required – formal parameter `chPtr`;
   2) required – a state variable;
   3) 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 use this variable in my solution 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". Do not assume that your program works correctly based strictly upon the results with this file, however. 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 all four source code files to the Assignment Checker with the subject line **C2A5E4_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your 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 don't go to another state but instead immediately return an appropriate status value. Don't use a separate variable to indicate a potential type **float** or type **long double** string. Instead, use different states to differentiate these findings.