8.1 Why pointers: Pass by pointer example

A challenging but powerful programming construct is something called a *pointer*. This section illustrates one example's beneficial usage of pointers, namely pass by pointer function parameters.

A function can only return one value. But consider a desired function that converts total inches into feet and inches, e.g., 95 inches would be converted to 7 feet and 11 inches. To effectively return two values, the function can be defined with two ***pass by pointer*** parameters, by putting a * before a parameter name, and & before the corresponding argument variable[parm].

The & passes the variable's memory address, known as a ***pointer***, rather than the variable's value. The * before the parameter name indicates the parameter is a pointer. The function's statements can update each argument variable's memory location, effectively "returning" a value. The technique is also known as ***pass by reference***, but the term pass by pointer avoids confusion with pass by reference parameters in C++ programs (which are different), and to more accurately describe this technique.

The following animation illustrates how pass by value does not work to return two values from a function.

PARTICIPATION ACTIVITY 8.1.1: Pointer example: Without pass by pointer parameters.

Animation captions:

1. ConvFeetInches' parameters are passed by value, so the arguments' values are copied into loc variables.
2. Upon return, ConvFeetInches' are discarded so the function fails to update the resFeet and res variables.

The following animation illustrates how pass by pointer effectively enables a function to return two values.

PARTICIPATION ACTIVITY 8.1.2: Pointer example: Pass by pointer parameters.

Animation captions:

1. The & before the argument indicates that a variable's memory addresses, known as a pointer, i passed to a pass-by-pointer parameter. The * before the parameter name indicates the parameter is a pointer.

2. Prepending "*" to a pointer variable's name access the value pointed to by the pointer, so the original variable is updated.
3. Upon return from ConvFeetInches, resFeet and resIn retain their updated values, effectively "returning" two values.

Pass by pointer parameters should be used only when the output values are tightly related. New programmers commonly create one function with two outputs (using pass by pointer) to reduce coding, where two functions would have been better. For example, defining two functions `int StepsToFeet(int baseSteps)` and `int StepsToCalories(int baseCalories)` is better than defining a single function `void StepsToFeetAndCalories(int baseSteps, int* feetTot, int* caloriesTot)` Defining separate functions supports modular development, and enables use of the functions in an expression as in `if (StepsToFeet(baseSteps) < 100)`.

Good candidates for multiple pass by pointer parameters might include computing the number of each type of coin to give as change, whose function might be `void ComputeChange(int totCents, int* numQuarters, int* numDimes, int* num` Another example is converting from polar coordinates to Cartesian coordinates, whose function might be `void PolarToCartesian(int radialPol, int anglePol, int* xCar, int* yCar)`. In both situations, the two outputs are tightly related.

PARTICIPATION ACTIVITY 8.1.3: Calculating change.

Complete the program to compute the number of quarter, dime, nickel, and penny coins that equal total cents, using the fewest coins (i.e., using the most possible of a larger coin first).

Load default template...

Run

```
#include <stdio.h>

void ComputeChange(int totCents) { // FIXME add four pass
   printf("FIXME: Finish this function.\n");
   return;
}

int main(void) {

   int totalCents = 67;    // Total amount of change nee
   int quartersChange = 0; // Number of quarters used fo
   int dimesChange    = 0; // Number of dimes used for c
   int nickelsChange  = 0; // Number of nickels used for
   int penniesChange  = 0; // Number of pennies used for

   ComputeChange(totalCents); // FIXME Add four pointer

   printf("Change for %d cents is:\n", totalCents);
```

PARTICIPATION ACTIVITY 8.1.4: Why pointer.

1) Complete the function declaration for MyFct with input parameters w and x, and "output" parameters y and z, in that order, all dealing with type int.

```
void MyFct(
                                    );
```

Check Show answer

2) Call a function MyFct that returns void, with four parameters, the last two being pointers. Call the function with argument variables a, b, c, and d, in that order, all being of type int.

Check Show answer

Exploring further:

- Pointers tutorial from msdn.microsoft.com

CHALLENGE ACTIVITY 8.1.1: Calling a function that has pass by pointer parameters.

Write a function call with arguments tensPlace, onesPlace, and userInt. Be sure to pass the first two arguments as pointers. Sample output for the given program:

```
tensPlace = 4, onesPlace = 1
```



```
1 #include <stdio.h>
2
3 void SplitIntoTensOnes(int* tensDigit, int* onesDigit, int DecVal){
4    *tensDigit = (DecVal / 10) % 10;
5    *onesDigit = DecVal % 10;
6    return;
7 }
```

```
8
9  int main(void) {
10    int tensPlace = 0;
11    int onesPlace = 0;
12    int userInt = 0;
13
14    userInt = 41;
15
16    /* Your solution goes here  */
17
18    printf("tensPlace = %d, onesPlace = %d\n", tensPlace, onesPlace);
19
```


Run

View your last submission

CHALLENGE ACTIVITY 8.1.2: Pass by pointer: Adjusting start/end times.

Define a function UpdateTimeWindow() with parameters timeStart, timeEnd, and offsetAmount. Each parameter is of type int. The function adds offsetAmount to each of the first two parameters. Make the first two parameters pass by pointer. Sample output for the given program:

```
timeStart = 3, timeEnd = 7
timeStart = 5, timeEnd = 9
```

```
1  #include <stdio.h>
2
3  // Define void UpdateTimeWindow(...)
4
5  /* Your solution goes here  */
6
7  int main(void) {
8     int timeStart = 0;
9     int timeEnd = 0;
10    int offsetAmount = 0;
11
12    timeStart = 3;
13    timeEnd = 7;
14    offsetAmount = 2;
15    printf("timeStart = %d, timeEnd = %d\n", timeStart, timeEnd);
16
17    UpdateTimeWindow(&timeStart, &timeEnd, offsetAmount);
18    printf("timeStart = %d, timeEnd = %d\n", timeStart, timeEnd);
19
```


Run

View your last submission

(*parm) Recall that the *parameter* is part of the function definition, while the *argument* is the item passed in a function call

8.2 Pointer basics

A ***pointer*** is a variable that contains a memory address, rather than containing data like most variables introduced earlier. The following program introduces pointers via example:

Figure 8.2.1: Introducing pointers via a simple example.

```
#include <stdio.h>

int main(void) {
   int usrInt = 0;    // User defined int value
   int* myPtr = NULL; // Pointer to the user defined int value

   // Prompt user for input
   printf("Enter any number: ");
   scanf("%d", &usrInt);

   // Output int value and location
   printf("We wrote your number into variable usrInt.\n");
   printf("The content of usrInt is: %d.\n", usrInt);
   printf("usrInt's memory address is: %p.\n", (void*) &usrInt);
   printf("\nWe can store that address into pointer variable myPtr.\n");

   // Grab location storing user value
   myPtr = &usrInt;

   // Output pointer value and value pointed by pointer
   printf("The content of myPtr is: %p.\n", (void*) myPtr);
   printf("The content of what myPtr points to is: %d.\n", *myPtr);

   return 0;
}
```

```
Enter any number: 555
We wrote your number into variable usrInt.
The content of usrInt is: 555.
usrInt's memory address is: 0x7fff5fbff908.

We can store that address into pointer variable myPtr.
The content of myPtr is: 0x7fff5fbff908.
The content of what myPtr points to is: 555.
```


The example demonstrates key aspects of working with pointers:

- *Appending "*" after a data type* in a variable declaration declares a pointer variable, as in `int* myPtr;`. One might imagine that the programming language would have a type like

"address" in addition to types like int, char, etc., but instead the language requires each pointer variable to indicate the type of data to which the address points. So valid pointer variable declarations are `int* myptr1;`, `char* myptr2;`, `double* myptr3`, and even `Seat* myptr4;` (where Seat is a struct type); all such variables will contain memory addresses.

- *Prepending "&" to any variable's name gets the variable's address*."&" is the reference operator that returns a pointer to a variable using the following form:

Construct 8.2.1: Reference operator.

```
&variableName
```

- *Prepending "*" to a pointer variable's name in an expression gets the data to which the variable points*, as in `*myPtr1`, an act known as ***dereferencing*** a pointer variable."*" is the dereference operator that allows the program to access the value pointed to by the pointer using the form:

Construct 8.2.2: Dereference operator.

```
*variableName
```

Observe the above program's output. For int variable usrInt, `printf("%p.\n", (void*) &usrInt);` prints usrInt's memory address. printf() can be used to print the memory address stored within a pointer variable using the format specifier "%p". %p expects the data type void*, but `&usrInt` is the data type int*. So, `&usrInt` is type cast to the data type void*.

Notice that memory address is a large number 0x7fff5fbff908 in contrast to short memory addresses like 96 that have appeared in earlier animations. That large number is in hexadecimal or base 16 number, which you need not concern yourself with as you will not normally print or ever have to look at such memory addresses — the memory address is printed here just for illustration.

The statement `myPtr = &usrInt;` will thus set myPtr's contents to that large address. `printf("%p.\n", (void*) myPtr);` will print myPtr's contents, which is that large address. `printf("%d.\n", *myPtr);` will instead go to that address and then print that address' contents.

The "*" (asterisk) symbol is used in two ways related to pointers. One is to indicate that a variable is a pointer type, as in `int* myPtr;`. The other is to dereference a pointer variable, as in `printf("%d.\n", *myPtr);`. Don't be confused by those two different uses; they have different meanings, both related to pointers.

The pointer was initialized to ***NULL***. In C, NULL is macro defined to represent that the pointer variable points to nothing. On most systems, NULL is defined as the value 0 because 0 is not a valid memory address.

The following animation illustrates pointers.

PARTICIPATION ACTIVITY 8.2.1: Simple pointer example.

Animation captions:

1. The pointer variable myPtr is initialized to NULL, which is a macro defined to represent that the pointer variable points to nothing.
2. User enters 555, which is stored in usrInt.
3. &usrInt evaluates to the address of usrInt, which is 76.
4. myPtr is assigned with the address of usrInt, so myPtr now points to usrInt.
5. *myPtr dereferences the pointer variable, evaluating to the data to which myPoint points. The data pointed to by myPtr is the data stored at memory location 76, which is 555.

The "*" in a pointer variable declaration has some syntactical options. We wrote `int* myPtr;`. However, also allowed is `int *myPtr;`. Many programmers find the former option that groups the "int" and "*" more intuitive, suggesting myPtr is of type "integer pointer". On the other hand, note that `int* myPtr1, myPtr2;` does *not* declare two pointers, but rather declares pointer variable myPtr1 , and int variable myPtr2. For this reason, some programmers prefer the option that groups the "*' with the variable name, as in `int *myPtr1, *myPtr2;`. Our advice: to reduce errors, it may be good practice to only declare one pointer per line, using the "int*" option.

PARTICIPATION ACTIVITY 8.2.2: Using pointers.

The following provides an example (not useful other than for learning) of assigning the address of variable vehicleMpg to the pointer variable valPtr.

1. Run and observe that the two output statements produce the same output.
2. Modify the value assigned to *valPtr and run again.
3. Now uncomment the statement that assigns vehicleMpg. PREDICT whether both output statements will print the same output. Then run and observe the output; did you predict correctly?

Load default template... Run

```
1
2 #include <stdio.h>
3
4 int main(void) {
5    double vehicleMpg = 0.0;
```

```
 6    double* valPtr = 0;
 7
 8    valPtr = &vehicleMpg;
 9
10    *valPtr = 29.6; // Assigns the number to the variable
11                    // POINTED TO by valPtr.
12
13   // vehicleMpg = 40;   // Uncomment this later
14
15    printf("Vehicle MPG = %lf\n", vehicleMpg);
16    printf("Vehicle MPG = %lf\n", *valPtr);
17    return 0;
18 }
```


PARTICIPATION ACTIVITY 8.2.3: Pointer basics.

Assume variable int `numStudents = 12` is at memory address 99, and `int* myPtr;` is at address 44. Answer "error" where appropriate.

1) What does `printf("%d", numStudents)` output?

Check Show answer

2) What does `printf("%p", (void*) &numStudents)` output?

Check Show answer

3) What does `printf("%d", *numStudents)` output?

Check Show answer

4) After `myPtr = &numStudents`, what does `printf("%d", *myPtr)` output?

Check Show answer

CHALLENGE ACTIVITY 8.2.1: Printing with pointers.

Assign numItems' address to numItemsPtr, then print the shown text followed by the value to which numItemsPtr points. End with newline.

```
Items: 99
```



```
1  #include <stdio.h>
2
3  int main(void) {
4     int* numItemsPtr = 0;
5     int numItems = 99;
6
7     /* Your solution goes here  */
8
9     return 0;
10 }
```

Run

View your last submission

8.3 The malloc and free functions

Sometimes memory should be allocated while a program is running and should persist independently of any particular function. The malloc() function carries out such memory allocation.

malloc() is a function defined within the ***standard library***, which can be used by adding `#include <stdlib.h>` to the beginning of a program. The malloc() function is named for *m*emory *alloc*ation. malloc() allocates memory of a given number of bytes and returns a pointer (i.e., the address) to that allocated memory. A basic call to malloc() has the form:

Construct 8.3.1: Malloc function.

```
malloc(bytes)
```

malloc() takes a single argument specifying the number of bytes to allocate in memory. Thus, the programmer must determine the number of bytes needed to allocate space for the desired data type. But, as you may recall, the number of bytes for various data types (e.g., int) may vary across different computer systems. Fortunately, C provides a sizeof() function that returns the number of bytes for a given data type. For example, on a system where an int is 32 bits, `sizeof(int)` returns 4, because 32-bits is 4 bytes. Calls to malloc() are typically combined with sizeof() using the form:

Construct 8.3.2: Malloc and sizeof functions.

```
malloc(sizeof(type))
```

malloc() returns a pointer to allocated memory using a `void*`, referred to as a ***void pointer***. A void pointer is a special type of pointer that stores a memory address without referring to the type of variable stored at that memory location. The void pointer return type allows a single malloc() function to allocate memory for any data type. The pointer returned from malloc() can then be written into a particular pointer variable by casting the void pointer to the data type using the form:

Construct 8.3.3: Malloc return type.

```
pointerVariableName = (type*)malloc(sizeof(type));
```

The following animation illustrates using malloc() to allocate memory for an int. int is used for introduction; malloc() is more commonly used to allocate memory for struct types, arrays, and strings.

PARTICIPATION ACTIVITY 8.3.1: The malloc() operation.

Animation captions:

1. Variable myPtr is a pointer of type int, which does not yet point to anything.
2. malloc(sizeof(int)) allocates memory for a single int. The void pointer returned by malloc is firs cast to an int pointer before myPtr is assigned with that pointer.

3. Dereferencing the pointer variable allows the program to assign and access the value pointed t by the pointer.

The malloc() function returns NULL if the function failed to allocate memory. Such failure could happen if a program has used up all memory available to the program.

free() is a function that deallocates a memory block pointed to by a given pointer, which must have been previously allocated by malloc(). In other words, free() does the opposite of malloc(). Deallocating memory using free() has the following form:

Construct 8.3.4: The free function.

```
free(pointerVariable);
```

After `free(pointerVariable);`, the program should not attempt to dereference pointerVariable, as pointerVariable points to a memory location that is no longer allocated for use by pointerVariable. Dereferencing a pointer whose memory has been deallocated is a common error, and may cause strange program behavior that is difficult to debug — if that memory had since been allocated to another variable, that variable's value could mysteriously change. Calling free with a pointer that wasn't previously allocated by malloc is also an error.

PARTICIPATION ACTIVITY 8.3.2: malloc and free.

1) Declare a variable named myValPointer as a pointer of type int, initializing the pointer to NULL in the declaration.

Check Show answer

2) Write a statement that allocates memory for a new double value using the pointer variable newInputPtr.

Check Show answer

3) Write a statement that deallocates memory for the pointer variable newInputPtr.

Check Show answer

Exploring further:

- malloc Reference Page from cplusplus.com
- More on malloc from msdn.microsoft.com
- free Reference Page from cplusplus.com
- More on free from msdn.microsoft.com

CHALLENGE ACTIVITY 8.3.1: Using malloc and pointers.

Write two statements that each use malloc to allocate an int location for each pointer. Sample output for given program:

```
numPtr1 = 44, numPtr2 = 99
```

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   int* numPtr1 = NULL;
   int* numPtr2 = NULL;

   /* Your solution goes here  */

   *numPtr1 = 44;
   *numPtr2 = 99;

   printf("numPtr1 = %d, numPtr2 = %d\n", *numPtr1, *numPtr2);

   free(numPtr1);
   free(numPtr2);

   return 0;
}
```


Run

View your last submission

8.4 Pointers with structs

The malloc() function is commonly used with struct types to allocate the appropriate block of memory for a variable of a struct type.

Figure 8.4.1: Using malloc() with a struct type.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct myItem_struct {
   int num1;
   int num2;
} myItem;

void myItem_PrintNums(myItem* itemPtr) {
   if (itemPtr == NULL) return;

   printf("num1: %d\n", itemPtr->num1);
   printf("num2: %d\n", itemPtr->num2);

   return;
}

int main(void) {
   myItem* myItemPtr1 = NULL;

   myItemPtr1 = (myItem*)malloc(sizeof(myItem));

   myItemPtr1->num1 = 5;
   (*myItemPtr1).num2 = 10;

   myItem_PrintNums(myItemPtr1);

   return 0;
}
```

```
num1: 5
num2: 10
```

Accessing a struct's member variables by first dereferencing a pointer, as in `(*myItemPtr1).num2 = 5;`, is so common that the language includes a second ***member access operator*** with the form:

Construct 8.4.1: Member access operator.

```
structPtr->memberName     // Equivalent to (*structPtr).memberName
```

The above program illustrates use of the member access operator: `myItemPtr1->num1 = 5;`.

PARTICIPATION ACTIVITY 8.4.1: The malloc, free, and -> operators.

Assuming the following is defined:

```
typedef struct Fruit_struct {
   // member variables
} Fruit;
```


1) Declare a variable named orange as a pointer of type Fruit.

Check Show answer

2) Write a statement that allocates memory for the new variable orange that points to class Fruit.

Check Show answer

3) For the variable orange, write a statement that assigns a member variable named hasSeeds to 1. Use the -> operator.

Check Show answer

4) Write a statement that deallocates memory pointed to by variable orange, which is a pointer of type Fruit.

Check Show answer

©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

5) Assuming a struct Fruit has two int data members and an int is 32 bits, what number does sizeof(Fruit) return?

Check Show answer

CHALLENGE ACTIVITY 8.4.1: Struct pointers.

Write two statements to assign numApples with 10 and numOranges with 3. Sample output for given program:


```
Apples: 10
Oranges: 3
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct bagContents_struct {
5     int numApples;
6     int numOranges;
7  } bagContents;
8
9  void bagContents_PrintBag(bagContents* itemPtr) {
10    if (itemPtr == NULL) return;
11
12    printf("Apples: %d\n", itemPtr->numApples);
13    printf("Oranges: %d\n", itemPtr->numOranges);
14
15    return;
16 }
17
18 int main(void) {
19    bagContents* groceryPtr = NULL;
```

Run

View your last submission

8.5 String functions with pointers

The C string library, introduced elsewhere, contains several functions for working with C strings. This section describes the use of char pointers in such functions. Recall that the C string library must first be included via: `#include <string.h>`

String functions accept a char pointer for a string argument. That pointer is commonly a char array variable, or a string literal (each of which is essentially a pointer to the 0th element of a char array), but could also be an explicit char pointer. Example of such functions are strcmp(), strcpy(), and strchr(), introduced elsewhere.

Figure 8.5.1: String functions accept char pointers as arguments.


```
char string1[10] = "abcxyz";
char string2[10] = "xyz";
char newText[10] = "";
char* subStr = NULL;

if (strcmp(string1, string2) == 0) {      // abcxyz does not equal xyz
   ...
if (strcmp(&string1[3], "xyz") == 0) {    // xyz equals xyz
   ...
subStr = &string1[3];                     // Points to 'x' in string1
if (strcmp(subStr, string2) == 0) {       // xyz equals xyz
   ...
strcpy(newText, subStr);                  // newText is now "xyz"
```

Table 8.5.1: Some C string modification functions.

Given:

```
char orgName[100] = "The Dept. of Redundancy Dept.";
char newText[100] = "";
char* subString = NULL;
```

strchr()	`strchr(sourceStr, searchChar)`
	
	Returns NULL if searchChar does not exist in sourceStr. Else, returns pointer to first occurrence.
		`if (strchr(orgName, 'D') != NULL) { // 'D' exists in orgName?`
		`  subString = strchr(orgName, 'D');`
		`// Points to first 'D'`
		`  strcpy(newText, subString);  // newText now "Dept. of Redundancy Dept."`
		`}`
		`if (strchr(orgName, 'Z') != NULL) { // 'Z' exists in orgName?`
		`  ... // Doesn't exist, branch not taken`
		`}`

strrchr()	`strrchr(sourceStr, searchChar)`
	
	Returns NULL if searchChar does not exist in sourceStr. Else, returns pointer to LAST occurrence (searches in reverse, hence middle 'r' in name).
		`if (strrchr(orgName, 'D') != NULL) { // 'D' exists in orgName?`
		`  subString = strrchr(orgName, 'D');`
		`// Points to last 'D'`
		`  strcpy(newText, subString);  // newText now "Dept."`
		`}`

strstr()	`strstr(str1, str2)`	

	Returns char* pointing to first occurrence of string str2 within string str1. Returns NULL if not found.	`subString = strstr(orgName, "Dept"); // Points to first 'D'`
		`if (subString != NULL) {`
		`  strcpy(newText, subString); // newText now "Dept. of Redundancy Dept."`
		`}`

The following example carries out a simple censoring program, replacing an exclamation point by a period and "Boo" by "---" (assuming those items are somehow bad and should be censored):

Figure 8.5.2: String searching example.

```
#include <stdio.h>
#include <string.h>

int main(void) {
   const int MAX_USER_INPUT = 100;  // Max input size
   char userInput[MAX_USER_INPUT];  // User defined
string
   char* stringPos = NULL;          // Index into string

   // Prompt user for input
   printf("Enter a line of text: ");
   fgets(userInput, MAX_USER_INPUT, stdin);

   // Locate exclamation point, replace with period
   stringPos = strchr(userInput, '!');

   if (stringPos != NULL) {
      *stringPos = '.';
   }

   // Locate "Boo" replace with "---"
   stringPos = strstr(userInput, "Boo");

   if (stringPos != NULL) {
      strncpy(stringPos, "---", 3);
   }

   // Output modified string
   printf("Censored: %s\n", userInput);

   return 0;
}
```

```
Enter a line of text: Hello!
Censored: Hello.

...

Enter a line of text: Boo hoo to
you!
Censored: --- hoo to you.

...

Enter a line of text: Booo!
Boooo!!!!
Censored: ---o. Boooo!!!!
```

Note above that only the first occurrence of "Boo" is replaced, as strstr() returns a pointer just to the first occurrence. (Additional code would be needed to delete all occurrences).

PARTICIPATION ACTIVITY 8.5.1: Modifying and searching strings.

1) Declare a char* variable named charPtr.

Check Show answer

2) Assuming `char* firstR;` is already declared, store in firstR a pointer to the first instance of an 'r' in the char* variable userInput.

Check Show answer

3) Assuming `char* lastR;` is already declared, store in lastR a pointer to the last instance of an 'r' in the char* variable userInput.

Check Show answer

4) Assuming `char* firstQuit;` is already declared, store in firstQuit a pointer to the first instance of "quit" in the char* variable userInput.

Check Show answer

CHALLENGE ACTIVITY 8.5.1: Find char in string.

Assign searchResult with a pointer to any instance of searchChar in personName. Hint: Use strchr().


```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void) {
5    char personName[100] = "Albert Johnson";
6    char searchChar = 'J';
7    char* searchResult = NULL;
8
```

```
   /* Your solution goes here  */

   if (searchResult != NULL) {
      printf("Character found.\n");
   }
   else {
      printf("Character not found.\n");
   }

   return 0;
```


Run

View your last submission

CHALLENGE ACTIVITY 8.5.2: Find string in string.

Assign movieResult with the first instance of The in movieTitle.

```
#include <stdio.h>
#include <string.h>

int main(void) {
   char movieTitle[100] = "The Lion King";
   char* movieResult = NULL;

   /* Your solution goes here  */

   printf("Movie title contains The? ");
   if (movieResult != NULL) {
      printf("Yes.\n");
   }
   else {
      printf("No.\n");
   }

   return 0;
}
```

Run

View your last submission

8.6 The malloc function for arrays and strings

Pointers are commonly used to allocate arrays just large enough to store a required number of data elements. Previously, the programmer had to declare arrays to have a fixed size, referred to as a ***statically allocated*** array. Statically allocated arrays may not use all the allocated memory due to the programmer not knowing the actual needed size before the program runs, thus creating larger-than-necessary arrays. Even then, the program might need a larger size.

Instead of statically allocating an array, malloc() can be used to allocate just enough memory for the array. Recall that arrays are stored in sequential memory locations. malloc() can be used to allocate a ***dynamically allocated*** array by determining the total number of bytes needed to store the desired number of elements, using the following form:

Construct 8.6.1: Dynamically allocated array.

```
pointerVariableName = (dataType*)malloc(numElements * sizeof(dataType))
```

When allocating an array, malloc() returns a pointer to memory location of the first element within the array. This memory location can be stored within a pointer variable declared as a pointer to the type of element within the array. For example, when dynamically allocating an array of integers, a pointer variable of integers "int*" is used. Notice then that an int* pointer can point to either a single integer or to an array of multiple characters. A programmer must carefully keep track of how each pointer variable is utilized within the program.

The following program illustrates how to dynamically allocate an arrays of integers.

Figure 8.6.1: Dynamically allocating an array of integers.

```
Enter number of integer values: 8
Enter 8 integer values...
Value: 4
Value: 23
Value: 14
Value: 5
Value: 4
Value: 3
Value: 7
Value: 9
You entered: 4 23 14 5 4 3 7 9
```

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   int* userVals = NULL; // No array yet
   int numVals = 0;
   int i = 0;

   printf("Enter number of integer values: ");
   scanf("%d", &numVals);

   userVals = (int*)malloc(numVals * sizeof(int));

   printf("Enter %d integer values...\n", numVals);
   for (i = 0; i < numVals; ++i) {
      printf("Value: ");
      scanf("%d", &(userVals[i]));
   }

   printf("You entered: ");
   for (i = 0; i < numVals; ++i) {
      printf("%d ", userVals[i]);
   }
   printf("\n");

   free(userVals);

   return 0;
}
```


PARTICIPATION ACTIVITY 8.6.1: Using malloc for arrays.

1) Write a malloc function call to allocate an array for 10 int variables.

```
int* itemList = NULL;
itemList = (int*)malloc(10 *
sizeof(          ));
```

Check Show answer

2) Write a malloc function call to allocate an array for 15 double variables.

```
double* priceList = NULL;
priceList = (          
)malloc(15 * sizeof(double));
```


Check Show answer

3) Using malloc, write a statement that allocates an array of 10 chars.

```
char* myStr = NULL;
myStr = (char*)malloc(
                      );
```

Check Show answer

The following program creates a string for a simple greeting given a user entered name.

Figure 8.6.2: Concatenating strings using a statically allocated array.

```
#include <stdio.h>
#include <string.h>

int main(void) {
   char nameArr[100] = "";        // User specified name
   char greetingArr[100] = "";    // Output greeting and name

   // Prompt user to enter a name
   printf("Enter name: ");
   fgets(nameArr, 100, stdin);

   // Eliminate end-of-line char
   nameArr[strlen(nameArr)-1] = '\0';

   // Modify string, hello + user specified name
   strcpy(greetingArr, "Hello ");
   strcat(greetingArr, nameArr);
   strcat(greetingArr, ".");

   // Output greeting and name
   printf("%s\n", greetingArr);

   return 0;
}
```

```
Enter name: Bill
Hello Bill.
```


The above program declares two statically allocated arrays named nameArr and greetingArr. Each array can store a string with 0 to 99 characters — keeping in mind the space needed to

store the null character at the end of the string. However, if the user enters the name "Bob", the resulting string stored within the greeting array only requires 11 characters — 6 characters for "Hello ", 3 characters for the name "Bob", 1 character for the ".", and 1 character for the null character. Thus, 88 characters are unused in the greeting array. Likewise, the program fails if the user enters a very long name.

The following program revises the earlier example by dynamically allocating a greeting array to be just large enough to store the entire greeting.

Figure 8.6.3: Concatenating strings using a dynamically allocated array.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
   char nameArr[100] = "";   // User specified name
   char* greetingPtr = NULL; // Pointer to output greeting and name

   // Prompt user to enter a name
   printf("Enter name: ");
   fgets(nameArr, 100, stdin);

   // Eliminate end-of-line char
   nameArr[strlen(nameArr) - 1] = '\0';

   // Dynamically allocate greeting
   greetingPtr = (char*)malloc((strlen(nameArr) + 8) * sizeof(char));

   // Modify string, hello + user specified name
   strcpy(greetingPtr, "Hello ");
   strcat(greetingPtr, nameArr);
   strcat(greetingPtr, ".");

   // Output greeting and name
   printf("%s\n", greetingPtr);

   free(greetingPtr);

   return 0;
}
```

```
Enter name: Julia
Hello Julia.

...

Enter name: John Smith
Hello John Smith.
```

©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

The nameArr array is used for reading the user input, so nameArr is statically allocated with a fixed size, defining a limit to the length of the name that the program can support.

In contrast, the size of the greeting is determined at runtime based upon the actual user-entered name length. `char* greetingPtr;` declares a char pointer variable named greetingPtr. Once the user has entered a name and the newline character at the end of the name string has been removed, `strlen(nameArr)` determines the number of characters within that name. In addition to the length of the string, 8 characters are needed for the greeting — 6 for the string "Hello ", 1 for the ".", and 1 for the end-of-string null character. Thus, the total number of bytes required for the greeting array can be determined using the expression `strlen(nameArr) + 8) * sizeof(char)`. For example, if the user enters the name "Julia", the total number of characters allocated for the greeting array will be 5 + 8 = 13.

The program can then create the greeting array by first copying the string "Hello " to the greeting array using the statement `strcpy(greetingPtr, "Hello ");`. The statement `strcat(greetingPtr, nameArr);` then appends the user-entered name to the end of the string stored within greetingPtr. Finally, `strcat(greetingPtr, ".");` appends a "." to the end of the string.

To create a dynamically allocated copy of a string, malloc() can be used to create a character array with a size equal to the number of characters within the source string plus one character for the null character. The following program illustrates. As the length returned by `strlen(nameArr)` does not include the null character required at the end of a string, `strlen(nameArr) + 1` is used to determine the number of characters required for the dynamically allocated string. A common error is to allocate only strlen chars for a copied string, forgetting the + 1 needed for the null character.

Figure 8.6.4: Creating a dynamically allocated copy of a string.

```
Enter name: Julia
Hello Julia

...

Enter name: John Smith
Hello John Smith
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
   char nameArr[50] = ""; // User specified name
   char* nameCopy = NULL; // Output greeting and name

   // Prompt user to enter a name
   printf("Enter name: ");
   fgets(nameArr, 50, stdin);

   // Create dynamically allocated copy of name
   // +1 is for the null character
   nameCopy = (char*)malloc((strlen(nameArr) + 1) * sizeof(char));
   strcpy(nameCopy, nameArr);


   // Output greeting and name
   printf("Hello %s", nameCopy);

   // Deallocate memory
   free(nameCopy);

   return 0;
}
```


PARTICIPATION ACTIVITY 8.6.2: Creating and modifying strings.

1) Given an existing string sentStart, complete the following statement to allocate a new string songVerse that is large enough to store the string sentStart plus seven additional characters.

```c
songVerse = (char*)malloc((
                          ) *
sizeof(char));
```

Check Show answer

2) In a single statement, copy the string pointed to by the char* sentStart to the string pointed to by the char* songVerse.

Check Show answer

CHALLENGE ACTIVITY 8.6.1: Pointers for allocating a C string.

Use strlen(userStr) to allocate exactly enough memory for newStr to hold the string in userStr (Hint: do NOT just allocate a size of 100 chars).

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>


int main(void) {
   char userStr[100] = "";
   char* newStr = NULL;

   strcpy(userStr, "Hello friend!");

   /* Your solution goes here  */

   strcpy(newStr, userStr);
   printf("%s\n", newStr);

   return 0;
}
```


Run

View your last submission

8.7 The realloc function

The ***realloc*** function re-allocates an original pointer's memory block to be the newly-specified size. realloc() can be used to both increase or decrease the size of dynamically allocated arrays. realloc() returns a pointer to the re-allocated memory. The pointer returned by realloc() may or may not differ from the original pointer. For example, if realloc() is used to increase a memory block but the function doesn't find enough available memory at the existing block's end, the function finds a large enough block of memory at another location in memory, and copies the existing block's contents to the new block.

In its most common form, the pointer returned from realloc() will be assigned to the same pointer provided as the input argument, using the form:

Construct 8.7.1: The realloc function.

```
pointerVariable = (type*)realloc(pointerVariable, numElements * sizeof(type))
```

The following program computes the average of several user-entered values stored within a dynamically allocated array. The array is reallocated each time the user specifies the number of integers values.

Figure 8.7.1: Dynamically reallocating the size of an array.

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   int* userVals = NULL; // No array yet
   int numVals = 0;
   int i = 0;
   char userInput = 'c';
   int userValsSum = 0;
   double userValsAvg = 0.0;

   while (userInput == 'c') {

      printf("Enter number of integer values: ");
      scanf(" %d", &numVals);

      if (userVals == NULL) {
         userVals = (int*)malloc(numVals * sizeof(int));
      }
      else {
         userVals = (int*)realloc(userVals, numVals * sizeof(int));
      }

      printf("Enter %d integer values...\n", numVals);
      for (i = 0; i < numVals; ++i) {
         printf("Value: ");
         scanf("%d", &(userVals[i]));
      }

      // Calculate average
      userValsSum = 0;
      for (i = 0; i < numVals; ++i) {
         userValsSum = userValsSum + userVals[i];
      }
      userValsAvg = (double)userValsSum / (double)numVals;

      printf("Average = %lf\n", userValsAvg);

      printf("\nEnter 'c' to compute another average (any other key to quit): ");
      scanf(" %c", &userInput);
   }

   free(userVals);

   return 0;
}
```

```
Enter number of integer values: 3
Enter 3 integer values...
Value: 13
Value: 11
Value: 17
Average = 13.666667

Enter 'c' to compute another average (any other key to quit): c
Enter number of integer values: 7
Enter 7 integer values...
Value: 10
Value: 14
Value: 56
Value: 23
Value: 18
Value: 3
Value: 6
Average = 18.571429

Enter 'c' to compute another average (any other key to quit): q
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

PARTICIPATION ACTIVITY 8.7.1: realloc.

1) Given an int* pointer dynInputVals that points to an existing dynamically

allocated array of integers, complete the following statement to reallocate the array to have 14 elements.

```
dynInputVals = (int*)realloc(
                , 14 *
sizeof(int));
```

Check Show answer

2) Given a double* pointer sensorVals that points to an existing dynamically allocated array of 200 elements, complete the following statement to reallocate the size of the array to have only 2 elements.

```
sensorVals =
(double*)realloc(sensorVals,
                 );
```

Check Show answer

8.8 Vector ADT

structs and pointers can be used to implement a computing concept known as an ***abstract data type (ADT)***, which is a data type whose creation and update are supported by specific well-defined operations. A key aspect of an ADT is that the internal implementation of the data and operations are hidden from the ADT user, a concept known as ***information hiding***, thus allowing the ADT user to be more productive by focusing on higher-level concepts, and also allowing the ADT developer to improve the internal implementation without requiring changes to programs using the ADT.

Programmers commonly refer to separating an ADT's *interface* from its *implementation*; the user of an ADT need only know the ADT's interface (functions declared within the ADT's header file) and not its implementation (function definitions and struct data members).

PARTICIPATION ACTIVITY 8.8.1: Abstract data types (ADT).

1) ADTs (Abstract data types) are meant to

the hide values of variables from the user.

- True
- False

2) An ADT's interface is commonly separated from its implementation.

- True
- False

3) An ADT is a fixed data type, like an int or char.

- True
- False

A vector is an example of ADT that stores an ordered list of items (called elements) of a given data type, such as a vector of integers. Like an array, an individual element within the vector can be accessed using an index. However, unlike an array, a vector's size can be adjusted while a program is executing, an especially useful feature when the number of items that will be in the list is unknown at compile-time. To support such size adjustment, a vector ADT provides functions for common operations like creating the vector, inserting elements into the vector, removing elements from the vector, resizing the vector, and accessing elements at specific locations.

The following defines an ADT for a vector of integers — defining a new struct type named "vector" and the function prototypes for the vector's interface. These declarations are included in the header file for the ADT named "vector.h". Note_vector_ADT

Figure 8.8.1: struct and function prototypes for vector ADT.

```
#ifndef VECTOR_H
#define VECTOR_H

// struct and typedef declaration for Vector ADT
typedef struct vector_struct {
   int* elements;
   unsigned int size;
} vector;

// interface for accessing Vector ADT

// Initialize vector
void vector_create(vector* v, unsigned int vectorSize);

// Destroy vector
void vector_destroy(vector* v);

// Resize the size of the vector
void vector_resize(vector* v, unsigned int vectorSize);

// Return pointer to element at specified index
int* vector_at(vector* v, unsigned int index);

// Insert new value at specified index
void vector_insert(vector* v, unsigned int index, int value);

// Insert new value at end of vector
void vector_push_back(vector* v, int value);

// Erase (remove) value at specified index
void vector_erase(vector* v, unsigned int index);

// Return number of elements within vector
int vector_size(vector* v);

#endif
```


To use the vector, a program must first include the following: `#include "vector.h"`. Then, a variable for a vector can be declared and used as shown in the below animation. The definition creates a vector variable named v. Data members within the vector struct are not initialized automatically. The vector_create() function is used to initialize the vector given the specified size of the vector. The function call `vector_create(&v, 3)` initializes the vector by allocating memory for three elements, each of type int, and initializes each of those elements to the value 0.

The function call `vector_at(&v, 0)` returns a pointer to the int element stored at location 0 of the the vector v. A value can be written to this location by dereferencing the returned int pointer. For example, the statement `*vector_at(&v, 0) = 119;` assigns a value of 119 to the int element at location 0 of the vector c.

Notice that the first parameter for each of the vector's functions is a pointer to a vector ("vector*"). When calling each of these functions, this parameter will point to the specific vector on which the corresponding operation will be performed. Thus, a program can consist of multiple vectors, using the same set of the vector functions.

PARTICIPATION ACTIVITY 8.8.2: Example use of vector ADT.

Animation captions:

1. A vector variable v is declared. The function call vector_create(&v, 3) allocates memory for thre elements, each of type int, and initializes each of those elements to 0.
2. The function call vector_at(&v, index) returns a pointer to the element at the location index. A value can be written to this location by dereferencing the returned int pointer.
3. Similarly, a value can be read from this location by dereferencing the returned int pointer.

The following summarizes the functions for the vector ADT defined above:

Table 8.8.1: Vector ADT functions.

Function	Description
`void vector_create(vector* v, unsigned int vectorSize)`	Initializes the vector pointed to by v with vectorSize number of elements. Each element with the vector is initialized to 0.
`void vector_destroy(vector* v)`	Destroys the vector by freeing all memory allocated within the vector.
`void vector_resize(vector* v, unsigned int vectorSize)`	Resizes the vector with vectorSize number of elements. If the

	vector size increased, each new element within the vector is initialized to 0.
`int* vector_at(vector* v, unsigned int index)`	Returns a pointer to the element at the location index.
`unsigned int vector_size(vector* v)`	Returns the vector's size — i.e. the number of elements within the vector.
`void vector_push_back(vector* v, int value)`	Inserts the value x to a new element at the end of the vector, increasing the vector size by 1.
`void vector_insert(vector* v, unsigned int index, int value)`	Inserts the value x to the element indicated by

	position, making room by shifting over the elements at that position and higher, thus increasing the vector size by 1.
`void vector_erase(vector* v, unsigned int index)`	Removes the element from the indicated position. All elements at higher positions are shifted over to fill the gap left by the removed element. Thus, the vector size decreases by 1.

The definition of the vector's functions are implemented within a file named "vector.c" shown below. Notice from the vector's struct definition that the vector has a data member for a *dynamically allocated array* of integers and a data member for the size of the array. For ADTs, a programmer should not directly access the data members of the struct in order to adhere to the concept to information hiding.

Figure 8.8.2: Function definitions for vector ADT.

```
#include <stdio.h>
#include <stdlib.h>
#include "vector.h"

// Initialize vector with specified size
void vector_create(vector* v, unsigned int vectorSize) {
   int i = 0;

   if (v == NULL) return;

   v->elements = (int*)malloc(vectorSize * sizeof(int));;
   v->size = vectorSize;
   for (i = 0; i < v->size; ++i) {
      v->elements[i] = 0;
   }
}

// Destroy vector
void vector_destroy(vector* v) {
   if (v == NULL) return;

   free(v->elements);
   v->elements = NULL;
   v->size = 0;
}

// Resize the size of the vector
void vector_resize(vector* v, unsigned int vectorSize) {
   int oldSize = 0;
   int i = 0;
   if (v == NULL) return;

   oldSize = v->size;
   v->elements = (int*)realloc(v->elements, vectorSize * sizeof(int));
   v->size = vectorSize;
   for (i = oldSize; i < v->size; ++i) {
      v->elements[i] = 0;
   }
}

// Return pointer to element at specified index
int* vector_at(vector* v, unsigned int index) {
   if (v == NULL || index >= v->size) return NULL;

   return &(v->elements[index]);
}

// Insert new value at specified index
void vector_insert(vector* v, unsigned int index, int value) {
   int i = 0;

   if (v == NULL || index > v->size) return;

   vector_resize(v, v->size + 1);
   for (i = v->size - 1; i > index; --i) {
      v->elements[i] = v->elements[i-1];
   }
   v->elements[index] = value;
}

// Insert new value at end of vector
void vector_push_back(vector* v, int value) {
   vector_insert(v, v->size, value);
}

// Erase (remove) value at specified index
```

```
// Erase (remove) value at specified index
void vector_erase(vector* v, unsigned int index) {
   int i = 0;

   if (v == NULL || index >= v->size) return;

   for (i = index; i < v->size - 1; ++i) {
      v->elements[i] = v->elements[i+1];
   }
   vector_resize(v, v->size - 1);
}

// Return number of elements within vector
int vector_size(vector* v) {
   if (v == NULL) return -1;

   return v->size;
}
```


A common error when using ADT such as a vector is passing an incorrect pointer to the vector's functions. For example, if a NULL pointer is passed to the vector_create() function, the expression `v->elements` would attempt to dereference a NULL pointer. As a NULL ptr refers to an invalid memory address, this type of error will cause a program to terminate in an error referred to as a ***segmentation fault***, ***access violation***, or ***bad access***. The actual error name depends on the computer system you are using.

To avoid this common error, each of the functions for the vector ADT first checks if the vector pointer is NULL before trying to dereference that pointer. If the pointer is NULL, the function will either return immediately or return a value indicating an error condition. For example, the function call `vector_size(NULL)` returns the value -1, indicating an error, as a vector cannot have a negative number of elements.

The following animation illustrates `vector_insert()` and `vector_erase()`.

PARTICIPATION ACTIVITY 8.8.3: Vector ADT functions.

Animation captions:

1. A variable vector v is declared and initialized.
2. The vector_erase( ) function removes an element from the indicated position. All elements at higher positions are shifted over and the vector size is decreased by one.
3. The vector_insert( ) function inserts a value in the indicated by position. All elements in a highe position are shifted over and the vector size is increased by one.
4. The vector_size( ) function returns the number of elements within the vector. The vector_destroy( ) function frees all memory allocated within the vector.

The following modifies an earlier number smoothing program to use the vector ADT rather than directly using an array. The benefit is that the program can support an arbitrary number of user-entered integer numbers.

Figure 8.8.3: Number smoothing program using vector.

```
#include <stdio.h>
#include <stdlib.h>
#include "vector.h"

// Get number from user
void GetNums(vector* nums) {
   int numsSize = 0; // Vector size
   int i = 0;        // Loop index

   printf("Enter number of integers to be entered: ");
   scanf("%d", &numsSize);

   vector_resize(nums, numsSize);

   for (i = 0; i < vector_size(nums); ++i) {
      printf("%d: ", i + 1);
      scanf("%d", vector_at(nums, i));
   }
}

// Smooths by setting element to average of itself and
next two elements
void FilterNums(vector* nums) {
   int i = 0;

   for (i = 0; i < vector_size(nums) - 2; ++i) {
      *vector_at(nums, i) = (*vector_at(nums, i) +
                             *vector_at(nums, i + 1) +
                             *vector_at(nums, i + 2)) / 3;
   }

   *vector_at(nums, i) = (*vector_at(nums, i) +
                          *vector_at(nums, i + 1)) / 2;

   // Last element needs no averaging
}

// Print all elements within the vector
void PrintsNums(vector* nums) {
   int i = 0;

   printf("Numbers: ");
   for (i = 0; i < vector_size(nums); ++i) {
      printf("%d ", *vector_at(nums, i));
   }
   printf("\n");
}

int main(void) {
   vector nums;

   vector_create(&nums, 0);

   GetNums(&nums);
   PrintsNums(&nums);
   FilterNums(&nums);
   PrintsNums(&nums);

   vector_destroy(&nums);

   return 0;
}
```



```
Enter number of integers to be
entered: 10
1: 10
2: 20
3: 30
4: 40
5: 50
6: 60
7: 70
8: 80
9: 90
10: 100
Numbers: 10 20 30 40 50 60 70 80
90 100
Numbers: 20 30 40 50 60 70 80 90
95 100
```

©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

PARTICIPATION ACTIVITY 8.8.4: Vector declaration and use.

Write a single statement for each answer.

1) Declare a vector named vals.

Check Show answer

2) Initialize a vector vals to size 10 with all elements set to 0.

Check Show answer

3) Assign the value of the element held at index 8 of vector vals to an int variable x.

Check Show answer

4) Write 555 into element at index 2 of vector vals.

Check Show answer

5) Store 777 into the second element of vector vals.

Check Show answer

©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

6) Append the value 37 to the vector vals.

Check Show answer

7) Set the int variable sz to the size of the vector vals.

Check Show answer

8) Erase element 0 of vector vals.

Check Show answer

CHALLENGE ACTIVITY 8.8.1: Operator overloading.

Modify the existing vector's contents, by erasing 200, then inserting 100 and 102 in the shown locations. Use Vector ADT's erase() and insert() only. Sample output of below program:

```
100 101 102 103
```

```
#include <stdio.h>
#include <stdlib.h>

// struct and typedef declaration for Vector ADT
typedef struct vector_struct {
  int* elements;
  unsigned int size;
} vector;

// Initialize vector with specified size
void vector_create(vector* v, unsigned int vectorSize) {
  int i = 0;

  if (v == NULL) return;

  v->elements = (int*)malloc(vectorSize * sizeof(int));
  v->size = vectorSize;
  for (i = 0; i < v->size; ++i) {
     v->elements[i] = 0;
```

Run

View your last submission

(*Note_vector_ADT) Note to instructors: For the vector ADT we use different convention for naming the functions. All functions supporting the vector ADT begin with "vector_", which is intended to indicate the functions are associated with the "vector" type. The following portion of the function corresponds to operation being performed on the vector ADT. The names for these operations for the vector ADT closely match the names of corresponding operations on the vector class in C++.

8.9 Why pointers: A list example

To further elaborate on the need for pointers, this section describes another of many situations where pointers are useful.

The vector ADT (or arrays) stores a list of items in contiguous memory locations, which enables immediate access to any element i of vector v by using `vector_at(&v, i)`. Recall that inserting an item within a vector requires making room by shifting higher-indexed items. Similarly, erasing an item requires shifting higher-indexed items to fill the gap. Each shift of an item from one element to another element requires a few processor instructions. This issue exposes the ***vector insert/erase performance problem***. For vectors with thousands of elements, a single call to insert() or erase() can require thousands of instructions, so if a program does many inserts or erases on large vectors, the program may run very slowly. The following animation illustrates shifting during an insert.

PARTICIPATION ACTIVITY 8.9.1: Vector insert performance problem.

Animation captions:

1. Inserting an item at a specific location in a vector requires making room for the item by shifting higher-indexed items.

The following program can be used to demonstrate the issue. The user inputs a vector size vectorSize, and a number numOps of elements to insert. The program then carries out several tasks, namely it resizes the vector to size vectorSIze, writes an arbitrary value to all vectorSIze elements, does numOps push_backs, numOps inserts, and numOps erases.

Figure 8.9.1: Program illustrating how slow vector inserts and erases can be.

```
#include <stdio.h>
#include <stdlib.h>
#include "vector.h"

int main(void) {
   vector tempValues;  // Dummy vector to demo vector ops
   int vectorSize = 0; // User defined vector size
   int numOps = 0;     // User defined number of inserts
   int i = 0;          // Loop index

   vector_create(&tempValues, 0);

   printf("Enter initial vector size: ");
   scanf("%d", &vectorSize);

   printf("Enter number of inserts: ");
   scanf("%d", &numOps);

   printf("  Resizing vector...");
   fflush(stdout);
   vector_resize(&tempValues, vectorSize);

   printf("done.\n");
   printf("  Writing to each element...");
   fflush(stdout);

   for (i = 0; i < vectorSize; ++i) {
      *vector_at(&tempValues, i) = 777; // Any value
   }

   printf("done.\n");
   printf("   Doing %d inserts at end...", numOps);
   fflush(stdout);

   for (i = 0; i < numOps; ++i) {
      vector_insert(&tempValues, vector_size(&tempValues), 888); // Any value
   }

   printf("done.\n");
   printf("   Doing %d inserts at beginning...", numOps);
   fflush(stdout);

   for (i = 0; i < numOps; ++i) {
      vector_insert(&tempValues, 0, 444);
   }

   printf("done.\n");
   printf("   Doing %d removes...", numOps);
   fflush(stdout);


   for (i = 0; i < numOps; ++i) {
      vector_erase(&tempValues, 0);
   }

   printf("done.\n");

   return 0;
}
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

```
Enter initial vector size: 100000
Enter number of inserts: 40000
  Resizing vector...done.                        (fast)
  Writing to each element...done.                (fast)
   Doing 40000 inserts at end...done.            (fast)
   Doing 40000 inserts at beginning...done.      (SLOW)
   Doing 40000 removes...done.                   (SLOW)
```


The video shows the program running for different vectorSize and numOps values; notice that for large vectorSize and numOps, the resize, writes, and numOps push_backs all run quickly, but the numOps inserts and numOps erases take a noticeably long time. The `fflush(stdout);` forces any characters written to stdout to be displayed on the screen before doing each task, lest the characters be held in the buffer until a task completes.

Video 8.9.1: Vector inserts.

The vector_push_backs() are fast because they do not involve any shifting of elements, whereas each vector_insert() requires 500,000 elements to be shifted — one at a time. 7,500 inserts thus requires 3,750,000,000 (over 3 billion) shifts.

One way to make inserts or erases faster is to use a different approach for storing a list of items. The approach does not use contiguous memory locations. Instead, each item contains a "pointer" to the next item's location in memory, as well as, the data being stored. Thus, inserting a new item B between existing items A and C just requires changing A to point to B's memory location, and B to point to C's location, as shown in the following animation.

PARTICIPATION ACTIVITY 8.9.2: A list avoids the shifting problem.

Animation captions:

1. List's first two items initially: (A, C, ...). Item A points to the next item at location 88. Item C points to next item at location 113 (not shown).
2. To insert new item B after item A, the new item B is first added to memory at location 90.
3. Item A is updated to point to location 90. Item B is set to point to location 88. New list is (A, B, C ...). No shifting of items after C was required.

A list whose items each point to the next item avoids the element shifting problem.

The animation begins with a list having some number of items, with the first two items being A and C. The first item has data A, and an address 88 pointing to the next item's location in memory, which just happens to be adjacent to the first item's location. That second item has data C, and an address 113 pointing to the next item (not shown). The animation shows a new item being created at memory location 90, having data B. To keep the list in sorted order, item B should go between A and C in the list. So item A's next pointer is changed to point to B's location of 90, and B's next pointer is set to point to 88.

A ***linked list*** is a list wherein each item contains not just data but also a pointer — a *link* — to the next item in the list. Comparing vectors and linked lists:

- *Vector*: Stores items in contiguous memory locations. Supports quick access to i'th element via `vector_at(&v, i)`, but may be slow for inserts or deletes on large lists due to necessary shifting of elements.
- *Linked list*: Stores each item anywhere in memory, with each item pointing to the next item in the list. Supports fast inserts or deletes, but access to i'th element may be slow as the list must be traversed from the first item to the i'th item. Also uses more memory due to storing a link for each item.

PARTICIPATION ACTIVITY 8.9.3: Vector performance.

1) Appending a new item to the end of a 1000 element vector requires how many elements to be shifted?

Check Show answer

2) Inserting a new item at the beginning of a 1000 element vector requires how many elements to be shifted?

Check Show answer

8.10 A first linked list

A common use of pointers is to create a list of items such that an item can be efficiently inserted somewhere in the middle of the list, without the shifting of later items as required for a vector. The following program illustrates how such a list can be created. A struct is defined to represent each list item, known as a ***list node***. A node is comprised of the data to be stored in each list item, in this case just one int, and a pointer to the next node in the list. A special node named head is created to represent the front of the list, after which regular items can be inserted.

Figure 8.10.1: A basic example to introduce linked lists.

```
-1
555
777
999
```

```
#include <stdio.h>
#include <stdlib.h>

typedef struct IntNode_struct {
   int dataVal;
   struct IntNode_struct* nextNodePtr;
} IntNode;

// Constructor
void IntNode_Create
(IntNode* thisNode, int dataInit, IntNode* nextLoc) {
   thisNode->dataVal = dataInit;
   thisNode->nextNodePtr = nextLoc;
   return;
}

/* Insert newNode after node.
 Before: thisNode -- next
 After:  thisNode -- newNode -- next
 */
void IntNode_InsertAfter
(IntNode* thisNode, IntNode* newNode) {
   IntNode* tmpNext = NULL;

   tmpNext = thisNode->nextNodePtr; // Remember next
   thisNode->nextNodePtr = newNode; // this -- new -- ?
   newNode->nextNodePtr = tmpNext;  // this -- new -- next
   return;
}

// Print dataVal
void IntNode_PrintNodeData(IntNode* thisNode) {
   printf("%d\n", thisNode->dataVal);
   return;
}

// Grab location pointed by nextNodePtr
IntNode* IntNode_GetNext(IntNode* thisNode) {
   return thisNode->nextNodePtr;
}

int main(void) {
   IntNode* headObj  = NULL; // Create intNode objects
   IntNode* nodeObj1 = NULL;
```

```
   IntNode* nodeObj2 = NULL;
   IntNode* nodeObj3 = NULL;
   IntNode* currObj  = NULL;

   // Front of nodes list
   headObj = (IntNode*)malloc(sizeof(IntNode));
   IntNode_Create(headObj, -1, NULL);

   // Insert nodes
   nodeObj1 = (IntNode*)malloc(sizeof(IntNode));
   IntNode_Create(nodeObj1, 555, NULL);
   IntNode_InsertAfter(headObj, nodeObj1);

   nodeObj2 = (IntNode*)malloc(sizeof(IntNode));
   IntNode_Create(nodeObj2, 999, NULL);
   IntNode_InsertAfter(nodeObj1, nodeObj2);

   nodeObj3 = (IntNode*)malloc(sizeof(IntNode));
   IntNode_Create(nodeObj3, 777, NULL);
   IntNode_InsertAfter(nodeObj1, nodeObj3);

   // Print linked list
   currObj = headObj;
   while (currObj != NULL) {
      IntNode_PrintNodeData(currObj);
      currObj = IntNode_GetNext(currObj);
   }

   return 0;
}
```


PARTICIPATION ACTIVITY 8.10.1: Inserting nodes into a basic linked list.

Animation captions:

1. The headObj pointer points to a special node that represents the front of the list. When the list first created, no list items exists, so the head node's nextNodePtr pointer is null.
2. To insert a node in the list, the new node nodeObj1 is first created with the value 555.
3. To insert the new node, tmpNext is pointed to the head node's next node, the head node's nextNodePtr is pointed to the new node, and the new node's nextNodePtr is pointed to tmpNex
4. A second node nodeObj2 with the value 999 is inserted at the end of the list, and a third node nodeObj3 with the value 777 is created.
5. To insert nodeObj3 after nodeObj1, tmpNext is pointed to the nodeObj1's next node, the nodeObj1's nextNodePtr is pointed to the nodeObj3, and nodeObj3's nextNodePtr is pointed to tmpNext.

The most interesting part of the above program is the InsertAfter() function, which inserts a new node after a given node already in the list. The above animation illustrates.

PARTICIPATION ACTIVITY 8.10.2: A first linked list.

Some questions refer to the above linked list code and animation.

1) A linked list has what key advantage over a sequential storage approach like an array or vector?

- An item can be inserted somewhere in the middle of the list without having to shift all subsequent items.
- Uses less memory overall.
- Can store items other than int variables.

2) What is the purpose of a list's head node?

- Stores the first item in the list.
- Provides a pointer to the first item's node in the list, if such an item exists.
- Stores all the data of the list.

3) After the above list is done having items inserted, at what memory address is the last list item's node located?

- 80
- 82
- 84
- 86

4) After the above list has items inserted as above, if a fourth item was inserted at the front of the list, what would happen to the location of node1?

- Changes from 84 to 86.
- Changes from 84 to 82.
- Stays at 84.

In contrast to the above program that declares one variable for each item allocated by the malloc function, a program commonly declares just one or a few variables to manage a large number of items allocated using the malloc function. The following example replaces the above main()

function, showing how just two pointer variables, currObj and lastObj, can manage 20 allocated items in the list.

Figure 8.10.2: Managing many new items using just a few pointer variables.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct IntNode_struct {
   int dataVal;
   struct IntNode_struct* nextNodePtr;
} IntNode;

// Constructor
void IntNode_Create
(IntNode* thisNode, int dataInit, IntNode* nextLoc) {
   thisNode->dataVal = dataInit;
   thisNode->nextNodePtr = nextLoc;
   return;
}

/* Insert newNode after node.
 Before: thisNode -- next
 After:  thisNode -- newNode -- next
 */
void IntNode_InsertAfter
(IntNode* thisNode, IntNode* newNode) {
   IntNode* tmpNext = NULL;

   tmpNext = thisNode->nextNodePtr; // Remember next
   thisNode->nextNodePtr = newNode;// this -- new -- ?
   newNode->nextNodePtr = tmpNext; // this -- new -- next
   return;
}

// Print dataVal
void IntNode_PrintNodeData(IntNode* thisNode) {
   printf("%d\n", thisNode->dataVal);
   return;
}

// Grab location pointed by nextNodePtr
IntNode* IntNode_GetNext(IntNode* thisNode) {
   return thisNode->nextNodePtr;
}

int main(void) {
   IntNode* headObj  = NULL; // Create intNode objects
   IntNode* currObj  = NULL;
   IntNode* lastObj  = NULL;
   int i             = 0;   // Loop index

   headObj = (IntNode*)malloc(sizeof(IntNode)); // Front of nodes list
   IntNode_Create(headObj, -1, NULL);
   lastObj = headObj;

   for (i = 0; i < 20; ++i) {                       // Append 20 rand nums
      currObj = (IntNode*)malloc(sizeof(IntNode));
      IntNode_Create(currObj, rand(), NULL);

      IntNode_InsertAfter(lastObj, currObj);    // Append curr
      lastObj = currObj;                        // Curr is the new last item
   }


   currObj = headObj;                           // Print the list
```

```
-1
1481765933
1085377743
1270216262
1191391529
812669700
553475508
445349752
1344887256
730417256
1812158119
147699711
880268351
1889772843
686078705
2105754108
182546393
1949118330
220137366
1979932169
1089957932
```

```
   while (currObj != NULL) {
      IntNode_PrintNodeData(currObj);
      currObj = IntNode_GetNext(currObj);
   }

   return 0;
}
```

PARTICIPATION ACTIVITY 8.10.3: Managing a linked list.

Finish the program so that it finds and prints the smallest value in the linked list.

Load default template... Run

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 typedef struct IntNode_struct {
6    int dataVal;
7    struct IntNode_struct* nextNodePtr;
8 } IntNode;
9
10 // Constructor
11 void IntNode_Create
12 (IntNode* thisNode, int dataInit, IntNode* nextLoc) {
13    thisNode->dataVal = dataInit;
14    thisNode->nextNodePtr = nextLoc;
15    return;
16 }
17
18 /* Insert newNode after node.
19  Before: thisNode -- next
```

Normally, a linked list would be implemented as an ADT using a set interface functions and struct type declaration, such as IntList. Data members of that struct might include a pointer to the list head, the list size, and a pointer to the list tail (the last node in the list). Supporting functions might include InsertAfter (insert a new node after the given node), PushBack (insert a new node after the last node), PushFront (insert a new node at the front of the list, just after the head), DeleteNode (deletes the node from the list), etc.

CHALLENGE ACTIVITY 8.10.1: Linked list negative values counting.

Assign negativeCntr with the number of negative values in the linked list, including the list head.

```
1 #include <stdio.h>
2 #include <stdlib.h>
```

```
3  typedef struct IntNode_struct {
4     int dataVal;
5     struct IntNode_struct* nextNodePtr;
6  } IntNode;
7
8  // Constructor
9  void IntNode_Create(IntNode* thisNode, int dataInit, IntNode* nextLoc) {
10    thisNode->dataVal = dataInit;
11    thisNode->nextNodePtr = nextLoc;
12 }
13
14 /* Insert newNode after node.
15  Before: thisNode -- next
16  After: thisNode -- newNode -- next
17 */
18 void IntNode_InsertAfter(IntNode* thisNode, IntNode* newNode) {
```


Run

View your last submission

8.11 Memory regions: Heap/Stack

A program's memory usage typically includes four different regions:

- ***Code*** — The region where the program instructions are stored.
- ***Static memory*** — The region where global variables (variables declared outside any function) as well as static local variables (variables declared inside functions starting with the keyword "static") are allocated. The name "static" comes from these variables not changing (static means not changing); they are allocated once and last for the duration of a program's execution, their addresses staying the same.
- ***The stack*** — The region where a function's local variables are allocated during a function call. A function call adds local variables to the stack, and a return removes them, like adding and removing dishes from a pile; hence the term "stack." Because this memory is automatically allocated and deallocated, it is also called ***automatic memory***.
- ***The heap*** — The region where the malloc function allocates memory, and where the free function deallocates memory. The region is also called ***free store***.

The following animation illustrates:

PARTICIPATION ACTIVITY 8.11.1: Use of the four memory regions.

Animation captions:

1. The code regions store program instructions. myGlobal is a global variable and is stored in the static memory region. Code and static regions last for the entire program execution.
2. Function calls push local variables on the program stack. When main() is called, the variables myInt and myPtr are added on the stack.
3. new allocates memory on the heap for an int and returns the address of the allocated memory which is assigned to myPtr. delete deallocates memory from the heap.
4. Calling MyFct() grows the stack, pushing the function's local variables on the stack. Those loca variables are removed from the stack when the function returns.
5. When main() completes, main's local variables are removed from the stack.

PARTICIPATION ACTIVITY 8.11.2: Stack and heap definitions.

Automatic memory | Static memory | Code | Free store | The stack | The heap

A function's local variables are allocated in this region while a function is called.

The memory allocation and deallocation operators affect this region.

Global and static local variables are allocated in this region once for the duration of the program.

Another name for "The heap" because the programmer has explicit control of this memory.

Instructions are stored in this region.

Another name for "The stack" because the programmer does not explicitly control this memory.

Reset

8.12 Memory leaks

A program that allocates memory but then loses the ability to access that memory, typically due to failure to properly destroy/free dynamically allocated memory, is said to have a ***memory leak***. The program's available memory has portions leaking away and becoming unusable, much like a water pipe might have water leaking out and becoming unusable. A memory leak may cause a program to occupy more and more memory as the program runs. Such occupying of memory can slow program runtime, or worse can cause the program to fail if the memory becomes completely full and the program is unable to allocate additional memory. The following animation illustrates.

PARTICIPATION ACTIVITY 8.12.1: Memory leak can use up all available memory.

Animation captions:

1. Memory is allocated for newVal each loop iteration, but the loop does not deallocate memory once done using newVal, resulting in a memory leak.
2. Each loop iteration allocates more memory, eventually using up all available memory, causing the program to fail.

Failing to free allocated memory when done using that memory, resulting in a memory leak, is a common error. Many programs that are commonly left running for long periods, such as web browsers, suffer from known memory leak problems — just do a web search for "<your-favorite-browser> memory leak" and you'll likely find numerous hits.

Some programming languages, such as Java, use a mechanism called ***garbage collection*** wherein a program's executable includes automatic behavior that at various intervals finds all unreachable allocated memory locations (e.g., by comparing all reachable memory with all previously-allocated memory), and automatically freeing such unreachable memory. Some C/C++ implementations include garbage collection but those implementations are not standard. Garbage collection can reduce the impact of memory leaks, at the expense of runtime overhead. Computer scientists debate whether new programmers should learn to explicitly free memory versus letting garbage collection do the work.

PARTICIPATION ACTIVITY 8.12.2: Memory Leaks.

Garbage collection Memory leak Unusable memory

Memory locations that have been dynamically allocated but can no longer be used by a program.
Occurs when a program allocates memory but loses the ability to access that memory.

Automatic process of finding unreachable allocated memory locations freeing that unreachable memory.

Reset

8.13 C example: Employee list using vector ADT

PARTICIPATION ACTIVITY 8.13.1: Managing an employee list using a vector ADT.

The following program allows a user to add to and list entries from a vector ADT, which maintains a list of employees.

1. Run the program, and provide input to add three employees' names and related data. Then use the list option to display the list.
2. Modify the program to implement the deleteEntry function.
3. Run the program again and add, list, delete, and list again various entries.

Current file: **main.c**

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4  #include "vector_employee.h"
5
6  // Add an employee
7  void AddEmployee(vector *employees) {
8     Employee theEmployee;
9
10    printf("\nEnter the name to add: \n");
11    fgets(theEmployee.name, 50, stdin);
12    theEmployee.name[strlen(theEmployee.name) - 1] = '\0'; // Remove trailing newline
```

```
13
14    printf("Enter %s's department: \n", theEmployee.name);
15    fgets(theEmployee.department, 50, stdin);
16    theEmployee.department[strlen(theEmployee.department) - 1] = '\0'; // Remove trailing n
17
18    printf("Enter %s's title: \n", theEmployee.name);
```

```
a
Rajeev Gupta
Sales
```


Run

Below is a solution to the above problem.

PARTICIPATION ACTIVITY 8.13.2: Managing an employee list using a vector ADT (solution).

Current file: **main.c**

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4  #include "vector_employee.h"
5
6  // Add an employee
7  void AddEmployee(vector *employees) {
8     Employee theEmployee;
9
10    printf("\nEnter the name to add: \n");
11    fgets(theEmployee.name, 50, stdin);
12    theEmployee.name[strlen(theEmployee.name) - 1] = '\0'; // Remove trailing newline
13
14    printf("Enter %s's department: \n", theEmployee.name);
15    fgets(theEmployee.department, 50, stdin);
16    theEmployee.department[strlen(theEmployee.department) - 1] = '\0'; // Remove trailing n
17
18    printf("Enter %s's title: \n", theEmployee.name);
19    fgets(theEmployee.title, 50, stdin);
```

```
a
Rajeev Gupta
Sales
```


Run

8.14 Ch 8 Warm up: Contacts (C)

You will be building a linked list. Make sure to keep track of both the head and tail nodes.

(1) Create three files to submit.

- Contacts.h - Struct definition, including the data members and related function declarations
- Contacts.c - Related function definitions
- main.c - main() function

(2) Build the ContactNode struct per the following specifications:

- Data members
- char contactName[50]
- char contactPhoneNum[50]
- struct ContactNode* nextNodePtr
- Related functions
- CreateContactNode() (2 pt)
- InsertContactAfter() (2 pts)
    - Insert a new node after node
- GetNextContact() (1 pt)
    - Return location pointed by nextNodePtr
- PrintContactNode()

Ex. of PrintContactNode() output:

```
Name: Roxanne Hughes
Phone number: 443-555-2864
```

(3) In main(), prompt the user for three contacts and output the user's input. Create three ContactNodes and use the nodes to build a linked list. (2 pts)

Ex:

```
Person 1
Enter name:
Roxanne Hughes
Enter phone number:
```

```
443-555-2864
You entered: Roxanne Hughes, 443-555-2864

Person 2
Enter name:
Juan Alberto Jr.
Enter phone number:
410-555-9385
You entered: Juan Alberto Jr., 410-555-9385

Person 3
Enter name:
Rachel Phillips
Enter phone number:
310-555-6610
You entered: Rachel Phillips, 310-555-6610
```


(4) Output the linked list. (2 pts)

Ex:

```
CONTACT LIST
Name: Roxanne Hughes
Phone number: 443-555-2864

Name: Juan Alberto Jr.
Phone number: 410-555-9385

Name: Rachel Phillips
Phone number: 310-555-6610
```

LAB ACTIVITY 8.14.1: Ch 8 Warm up: Contacts (C) 9 / 9

Submission Instructions

Deliverables

`Contacts.h` , `Contacts.c` and `main.c` *You must submit these file(s)*

Compile command

`gcc Contacts.c main.c -Wall -o a.out -lm` *We will use this command to compile your cc*

Submit your files below by dragging and dropping into the area or choosing a file on your hard drive

Contacts.h	Contacts.c	main.c
Drag file here or Choose on hard drive.	Drag file here or Choose on hard drive.	

Submit for grading This lab can only be submitted once every 3 r

Latest submission - 7:46 PM on 04/03/18 — Submission passed all tests ✓ — Total sco 9

☐ Only show failing tests — Download this subr

1: Unit test

Tests that CreateContactNode() initializes a node with name "Jane Doe" and phone number 555-5555".

Test feedback

```
Node correctly initialized.
```

2: Compare output

Input

```
Roxanne Hughes
443-555-2864
Juan Alberto Jr.
410-555-9385
Rachel Phillips
310-555-6610
```

Your output correctly starts with

```
Person 1
Enter name:
Enter phone number:
You entered: Roxanne Hughes, 443-555-2864

Person 2
Enter name:
Enter phone number:
You entered: Juan Alberto Jr., 410-555-9385

Person 3
Enter name:
Enter phone number:
You entered: Rachel Phillips, 310-555-6610
```

3: Compare output

Input

```
David Frederick
240-555-2104
Bernard Green
802-555-4986
Hillary Jones
212-555-9467
```


Your output correctly starts with

```
Person 1
Enter name:
Enter phone number:
You entered: David Frederick, 240-555-2104

Person 2
Enter name:
Enter phone number:
You entered: Bernard Green, 802-555-4986

Person 3
Enter name:
Enter phone number:
You entered: Hillary Jones, 212-555-9467
```

4: Unit test

Tests that InsertContactAfter() correctly inserts Juan Alberto Jr.'s ContactNode after Roxan Hughes's ContactNode and GetNextContact() correctly returns Juan Alberto Jr.'s ContactNc

Test feedback

```
InsertContactAfter() correctly inserts new node a
```

5: Compare output

Input

```
Roxanne Hughes
443-555-2864
Juan Alberto Jr.
410-555-9385
Rachel Phillips
310-555-6610
```



```
Person 1
```

Your output

```
Enter name:
Enter phone number:
You entered: Roxanne Hughes, 443-555-2864

Person 2
Enter name:
Enter phone number:
You entered: Juan Alberto Jr., 410-555-9385

Person 3
Enter name:
Enter phone number:
You entered: Rachel Phillips, 310-555-6610

CONTACT LIST
Name: Roxanne Hughes
Phone number: 443-555-2864

Name: Juan Alberto Jr.
Phone number: 410-555-9385

Name: Rachel Phillips
Phone number: 310-555-6610
```


6: Compare output

Input

```
David Frederick
240-555-2104
Bernard Green
802-555-4986
Hillary Jones
212-555-9467
```

```
Person 1
Enter name:
Enter phone number:
You entered: David Frederick, 240-555-2104

Person 2
Enter name:
Enter phone number:
You entered: Bernard Green, 802-555-4986
```

Your output

```
Person 3
Enter name:
Enter phone number:
You entered: Hillary Jones, 212-555-9467

CONTACT LIST
Name: David Frederick
Phone number: 240-555-2104

Name: Bernard Green
Phone number: 802-555-4986

Name: Hillary Jones
Phone number: 212-555-9467
```


5 previous submissions

7:43 PM on 4/3/18	7 / 9	View
7:37 PM on 4/3/18	7 / 9	View
7:34 PM on 4/3/18	5 / 9	View
7:30 PM on 4/3/18	5 / 9	View
7:25 PM on 4/3/18	7 / 9	View

8.15 Ch 8 Program: Playlist (C)

You will be building a linked list. Make sure to keep track of both the head and tail nodes.

(1) Create three files to submit.

- Playlist.h - Struct definition and related function declarations
- Playlist.c - Related function definitions

- main.c - main() function

Build the PlaylistNode class per the following specifications. Note: Some functions can initially be function stubs (empty functions), to be completed in later steps.

- Private data members
- char uniqueID[50]
- char songName[50]
- char artistName[50]
- int songLength
- PlaylistNode* nextNodePtr
- Related functions
- CreatePlaylistNode() (1 pt)
- InsertPlaylistNodeAfter() (1 pt)
  - Insert a new node after node
- SetNextPlaylistNode() (1 pt)
  - Set a new node to come after node
- GetNextPlaylistNode()
  - Return location pointed by nextNodePtr
- PrintPlaylistNode()

Ex. of PrintPlaylistNode output:

```
Unique ID: S123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237
```

(2) In main(), prompt the user for the title of the playlist. (1 pt)

Ex:

```
Enter playlist's title:
JAMZ
```

(3) Implement the PrintMenu() function. PrintMenu() takes the playlist title as a parameter and outputs a menu of options to manipulate the playlist. Each option is represented by a single character. Build and output the menu within the function.

If an invalid character is entered, continue to prompt for a valid choice. Hint: Implement Quit before implementing other options. Call PrintMenu() in the main() function. Continue to execute the menu until the user enters q to Quit. (3 pts)

Ex:

```
JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


(4) Implement "Output full playlist" menu option. If the list is empty, output: **`Playlist is empty`** (3 pts)

Ex:

```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

3.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

4.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

5.
Unique ID: SD567
```

```
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306
```

(5) Implement the "Add song" menu item. New additions are added to the end of the list. (2 pts)

Ex:


```
ADD SONG
Enter song's unique ID:
SD123
Enter song's name:
Peg
Enter artist's name:
Steely Dan
Enter song's length (in seconds):
237
```

(6) Implement the "Remove song" function. Prompt the user for the unique ID of the song to be removed.(4 pts)

Ex:

```
REMOVE SONG
Enter song's unique ID:
JJ234
"All For You" removed
```

(7) Implement the "Change position of song" menu option. Prompt the user for the current position of the song and the desired new position. Valid new positions are 1 - *n* (the number of nodes). If the user enters a new position that is less than 1, move the node to the position 1 (the head). If the user enters a new position greater than *n*, move the node to position *n* (the tail). 6 cases will be tested:

- Moving the head node (1 pt)
- Moving the tail node (1 pt)
- Moving a node to the head (1 pt)
- Moving a node to the tail (1 pt)
- Moving a node up the list (1 pt)
- Moving a node down the list (1 pt)

Ex:

```
CHANGE POSITION OF SONG
Enter song's current position:
3
Enter new position for song:
2
"Canned Heat" moved to position 2
```


(8) Implement the "Output songs by specific artist" menu option. Prompt the user for the artist's name, and output the node's information, starting with the node's current position. (2 pt)

Ex:

```
OUTPUT SONGS BY SPECIFIC ARTIST
Enter artist's name:
Janet Jackson

2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

4.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197
```

(9) Implement the "Output total time of playlist" menu option. Output the sum of the time of the playlist's songs (in seconds). (2 pts)

Ex:

```
OUTPUT TOTAL TIME OF PLAYLIST (IN SECONDS)
Total time: 1461 seconds
```


LAB ACTIVITY 8.15.1: Ch 8 Program: Playlist (C) 25 / 26

Submission Instructions

Deliverables

`Playlist.h` , `Playlist.c` and `main.c` *You must submit these file(s)*

Compile command

```
gcc Playlist.c main.c -Wall -o a.out -lm
```

We will use this command to compile your cc

Submit your files below by dragging and dropping into the area or choosing a file on your hard drive

Playlist.h	Drag file here or **Choose on hard drive.**	**Playlist.c**	Drag file here or **Choose on hard drive.**	**main.c**

Submit for grading

Latest submission - 10:44 AM on 04/05/18 Total score:

☐ Only show failing tests **Download this subr**

1: Unit test

Tests that CreatePlaylistNode(node, "SD123", "Peg", "Steely Dan", 237, NULL) initializes a no correctly.

Test feedback `Node correctly initialized.`

2: Unit test

Tests that InsertPlaylistNodeAfter() correctly inserts "All For You"'s node after "Peg"'s node.

Test feedback `InsertPlaylistNodeAfter() correctly inserts new n`

3: Unit test

Tests that SetNextPlaylistNode() correctly sets "Canned Heat" to be "All For You"'s next node

Test feedback `SetNextPlaylistNode() correctly sets next node.`

4: Compare output

Input

```
JAMZ
q
```

Your output correctly starts with

```
Enter playlist's title:
```

5: Compare output

Input

```
JAMZ
q
```


Your output

```
Enter playlist's title:

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```

6: Compare output

Input

```
Dance List
q
```

Your output

```
Enter playlist's title:

Dance List PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


7: Compare output

```
JAMZ
```

Input

```
o
q
```

Your output correctly ends with

```
JAMZ - OUTPUT FULL PLAYLIST
Playlist is empty

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


8: Compare output

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
o
q
```

```
ADD SONG
Enter song's unique ID:
Enter song's name:
Enter artist's name:
Enter song's length (in seconds):

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```

Your output correctly ends with

```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


9: Compare output

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
```

```
I Got The News
Steely Dan
306
o
q
```

Your output correctly ends with

```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

3.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

4.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

5.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

```
q - Quit

Choose an option:
```

10: Compare output

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
r
JJ234
q
```


Your output correctly

```
REMOVE SONG
Enter song's unique ID:
"All For You" removed.

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
```

ends with

```
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


11: Compare output

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
r
JJ234
o
q
```


Your output ends with

Your program produced no output

Expected output ends with

```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

3.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

4.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


12: Compare output

```
JAMZ
a
SD123
Peg
Steely Dan
```

Input

```
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
c
1
3
o
q
```



```
CHANGE POSITION OF SONG
Enter song's current position:
Enter new position for song:
"Peg" moved to position 3

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
JAMZ - OUTPUT FULL PLAYLIST
1.
```

Your output correctly ends with

```
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

2.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

3.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

4.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

5.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

13: Compare output

```
JAMZ
```

Input

```
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
c
5
3
o
q
```



```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391
```

Your output correctly ends with

```
3.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

4.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

5.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


14: Compare output

```
Program generated too much output.
Output restricted to 50000 characters.

Check program for any unterminated loops generating output.
```



```
JAMZ
a
SD123
Peg
Steely Dan
237
a
```

Input

```
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
c
4
1
o
q
```



```
J456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
398.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
399.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
400.↵
Unique ID: J345↵
```

```
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
401.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
402.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
403.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
404.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
405.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
406.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
407.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson
```

```
Song Length (in seconds): 391↵
↵
408.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
409.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197↵
↵
410.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
411.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391

412.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
413.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
414.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237↵
↵
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

```
415.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
416.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
417.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
418.↵
Unique ID: SD123↵
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237↵
↵
419.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

420.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330↵

421.
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
422.↵
Unique ID: SD123↵
```

```
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
423.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
424.
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
425.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
426.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
427.↵
Unique ID: JJ234
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
428.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
429.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

```
Song Length (in seconds): 197↵
↵
430.↵
Unique ID: SD123↵
Song Name: Peg
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
431.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
432.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai
Song Length (in seconds): 330↵
↵
433.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
434.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
435.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
436.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

Your output ends with

```
437.
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
438.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
439.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
440.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
441.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
442.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
443.↵
Unique ID: JJ234
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
444.↵
Unique ID: J345↵
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

```
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
445.↵
Unique ID: JJ456↵
Song Name: Black Eagle
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
446.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
447.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson
Song Length (in seconds): 391↵
↵
448.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
449.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
450.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
451.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
```

```
Song Length (in seconds): 391↵
↵
452.↵
Unique ID: J345↵
Song Name: Canned Heat
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
453.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
454.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
455.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
456.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
457.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
458.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

```
459.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
460.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai
Song Length (in seconds): 330↵
↵
461.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
462.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
463.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
464.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
465.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
466.↵
Unique ID: SD123↵
```

```
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
467.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
468.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai
Song Length (in seconds): 330↵
↵
469.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
470.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
471.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
472.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
473.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
```

```
Song Length (in seconds): 197↵
↵
474.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
475.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
476.↵
Unique ID: J345↵
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
477.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
478.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
479.
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 391↵
↵
480.
Unique ID: J345
Song Name: Canned Heat↵
Artist Name: Jamiroquai↵
Song Length (in seconds): 330↵
↵
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

```
481.↵
Unique ID: JJ456↵
Song Name: Black Eagle↵
Artist Name: Janet Jackson↵
Song Length (in seconds): 197↵
↵
482.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
483.↵
Unique ID: JJ234↵
Song Name: All For You↵
Artist Name: Janet Jackson
Song
```


Expected output ends with

```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

2.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

3.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

4.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

5.
```

```
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


15: Compare output

Output differs. See highlights below. Special character legend

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
```

```
I Got The News
Steely Dan
306
c
3
5
o
q
```



```
 playlist's title:↵
↵
JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:
ADD SONG
Enter song's unique ID:↵
Enter song's name:↵
Enter artist's name:
Enter song's length (in seconds):↵
↵
JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:↵
ADD SONG↵
Enter song's unique ID:↵
Enter song's name:↵
Enter artist's name:
Enter song's length (in seconds):↵
↵
JAMZ PLAYLIST MENU↵
```

```
a - Add song↵
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
```



```
Choose an option:
ADD SONG
Enter song's unique ID:↵
Enter song's name:↵
Enter artist's name:↵
Enter song's length (in seconds):↵

JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:↵
ADD SONG↵
Enter song's unique ID:↵
Enter song's name:
Enter artist's name:↵
Enter song's length (in seconds):↵
↵
JAMZ PLAYLIST MENU↵
a - Add song
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵

Choose an option:
ADD SONG
Enter song's unique ID:↵
Enter song's name:↵
```

Your output ends with

```
Enter artist's name:↵
Enter song's length (in seconds):↵

JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:
CHANGE POSITION OF SONG↵
Enter song's current position:↵
Enter new position for song:↵
"Canned Heat" moved to position 5↵
↵
JAMZ PLAYLIST MENU↵
a - Add song
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg↵
Artist Name: Steely Dan↵
Song Length (in seconds): 237↵
↵
2.↵
Unique ID: JJ234↵
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

3.
Unique ID: J345
Song Name: Canned Heat
```

```
Artist Name: Jamiroquai
Song Length (in seconds): 330

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


Expected output ends with

```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

3.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

4.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

5.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330
```

```
JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


16: Compare output

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
c
3
2
```

```
o
q
```

Your output correctly ends with

```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

3.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

4.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

5.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```

17: Compare output

Output differs. See highlights below. Special character legend

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
c
2
4
o
q
```


©zyBooks 04/05/18 21:45 261830
Julian Chan
WEBERCS2250ValleSpring2018

```
 playlist's title:↵
↵
JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
```

```
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:
ADD SONG
Enter song's unique ID:↵
Enter song's name:↵
Enter artist's name:
Enter song's length (in seconds):↵
↵
JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:↵
ADD SONG↵
Enter song's unique ID:↵
Enter song's name:↵
Enter artist's name:
Enter song's length (in seconds):↵
↵
JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵

Choose an option:
ADD SONG
Enter song's unique ID:↵
Enter song's name:↵
Enter artist's name:
Enter song's length (in seconds):↵
↵
JAMZ PLAYLIST MENU↵
```

```
a - Add song↵
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:
ADD SONG↵
Enter song's unique ID:↵
Enter song's name:↵
Enter artist's name:↵
Enter song's length (in seconds):↵
↵
JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song↵
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:
ADD SONG↵
Enter song's unique ID:↵
Enter song's name:↵
Enter artist's name:↵
Enter song's length (in seconds):↵
↵
JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:↵
CHANGE POSITION OF SONG
Enter song's current position:↵
Enter new position for song:↵
```

Your output ends with

```
"All For You" moved to position 4↵
↵
JAMZ PLAYLIST MENU↵
a - Add song↵
r - Remove song↵
c - Change position of song
s - Output songs by specific artist↵
t - Output total time of playlist (in seconds)↵
o - Output full playlist↵
q - Quit↵
↵
Choose an option:↵
JAMZ - OUTPUT FULL PLAYLIST↵
1.↵
Unique ID: SD123↵
Song Name: Peg↵
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

3.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```



```
JAMZ - OUTPUT FULL PLAYLIST
1.
```

Expected output ends with

```
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

3.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

4.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

5.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


18: Compare output

```
JAMZ
```

Input

```
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
s
Janet Jackson
q
```


Your output correctly ends with

```
OUTPUT SONGS BY SPECIFIC ARTIST
Enter artist's name:
2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

4.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

JAMZ PLAYLIST MENU
```

```
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


19: Compare output

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
s
Steely Dan
q
```



```
OUTPUT SONGS BY SPECIFIC ARTIST
```

Your output correctly ends with

```
Enter artist's name:
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

5.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```


20: Compare output

Input

```
JAMZ
a
SD123
Peg
Steely Dan
237
a
JJ234
All For You
Janet Jackson
391
a
J345
Canned Heat
Jamiroquai
330
a
```

```
JJ456
Black Eagle
Janet Jackson
197
a
SD567
I Got The News
Steely Dan
306
t
q
```


Your output correctly ends with

```
OUTPUT TOTAL TIME OF PLAYLIST (IN SECONDS)
Total time: 1461 seconds

JAMZ PLAYLIST MENU
a - Add song
r - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```

5 previous submissions

10:41 AM on 4/5/18	21 / 26	View
10:39 AM on 4/5/18	21 / 26	View
9:58 AM on 4/5/18	25 / 26	View
5:35 PM on 4/4/18	22 / 26	View
5:33 PM on 4/4/18	23 / 26	View