Barak Barclay

Leslie Tekamp

ECE-1021

7 December 2015

HWK9

1. 7.1 Answer each of the following:

a) A pointer variable contains as its value the address of another variable.

b) The three values that can be used to initialize a pointer are 0, NULL and address.

c) The only integer that can be assigned to a pointer is 0.

1. 7.2 State whether the following are true or false. If the answer is false, explain why.

a) The address operator (&) can be applied only to constants, to expressions and to variables declared with the storage-class register.

False, the address operator can be applied only to variables. The address operator cannot be applied to variables declared with storage class register.

b) A pointer that’s declared to be void can be dereferenced.

False, a pointer to void cannot be dereferenced, because there’s no way to know exactly how many bytes of memory to dereference.

c) Pointers of different types may not be assigned to one another without a cast operation.

False, pointers of type void can be assigned pointers of other types, and pointers of type void can be assigned to pointers of other types.

//

// HWK9(3).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Use a sentinel loop - this program prompts the user to enter a message number. The program

// will then use a switch statement to pick the message to write to the screen. Write an error message if an

// invalid message number is entered. Enter a negative number to Quit.

//

// %%%% Algorthim %%%%

//

// preprocessor directives

// start main

// init variables

// while

// prompt user to enter and scan

// switch

// case (1-4)

// for (until end of string designator)

// print messagge#[i]

// end for

// default case

// error message

// end switch

// end while

// end main

//

#include <stdio.h>

void main(void)

{

int messageToWrite = 0;

int i;

char message1[] = "My message 1.";

char message2[] = " You messed-up. Do it again.";

char message3[] = " Humans are so hard to work with.";

char message4[] = " Good job.";

while (messageToWrite >= 0)

{

printf("Enter a message number (negative to quit):");

scanf\_s("%d", &messageToWrite);

switch (messageToWrite)

{

case 1:

{

for (i = 0; message1[i] != '\0'; i++)

{

printf\_s("%c", message1[i]);

}

puts("");

break;

}

case 2:

{

for (i = 0; message2[i] != '\0'; i++)

{

printf\_s("%c", message2[i]);

}

puts("");

break;

}

case 3:

{

for (i = 0; message3[i] != '\0'; i++)

{

printf\_s("%c", message3[i]);

}

puts("");

break;

}

case 4:

{

for (i = 0; message4[i] != '\0'; i++)

{

printf\_s("%c", message4[i]);

}

puts("");

break;

}

default:

{

printf("Error - Message numbers are numbers 1-4.");

puts("");

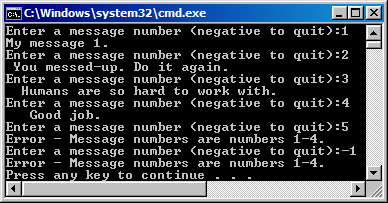
break;

}

}

}

}



//

// HWK9(4).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Now rewrite the program from problem 1, only this time use the messageArray elements to

// print the messages to the screen.

//

// %%%% Algorthim %%%%

//

// remove for loops from problem 1 and put in print string messages for the pointers

//

#include <stdio.h>

void main(void)

{

int messageToWrite = 0;

int i;

char message1[] = "My message 1.";

char message2[] = " You messed-up. Do it again.";

char message3[] = " Humans are so hard to work with.";

char message4[] = " Good job.";

char \*messageArray[5]; // an array of pointers

messageArray[0] = NULL;

messageArray[1] = message1; // message1 is an address - pointer

messageArray[2] = message2;

messageArray[3] = message3;

messageArray[4] = message4;

while (messageToWrite >= 0)

{

printf("Enter a message number (negative to quit):");

scanf\_s("%d", &messageToWrite);

switch (messageToWrite)

{

case 1:

{

printf\_s("%s", messageArray[1]);

puts("");

break;

}

case 2:

{

printf\_s("%s", messageArray[2]);

puts("");

break;

}

case 3:

{

printf\_s("%s", messageArray[3]);

puts("");

break;

}

case 4:

{

printf\_s("%s", messageArray[4]);

puts("");

break;

}

default:

{

printf("Error - Message numbers are numbers 1-4.");

puts("");

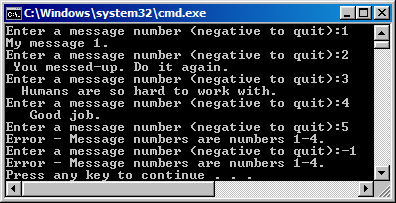
break;

}

}

}

}



//

// HWK9(5).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Calculator program. Implement in a sentinel loop.

// Call a Function to print a menu to the screen.Pass nothing.Return an integer(operation to do).The menu options

// are : Enter a 0 for a no operation(NoOp) option.Enter a 1 to do addition, enter a 2 to do subtraction, enter a

// 3 to do multiplication, and enter 4 to do division.Enter a negative number to Quit sentinel loop.Read in the menu

// option.Do error checking to make sure the user enters in a valid menu option.

// In the main function, use a switch statement to implement the calculator.Use a 4 element double array.In each of

// the case statements, call a function to implement the operation choice.Before you call the function, store the

// operation code(case number – same as entered from menu function) in element zero as a double.Call the Operation

// Function.

// Operation Functions - pass a four element array of type double, return nothing.In each of the math operation

// functions(+, -, \*, / ), prompt the user for two double numbers, read in the two numbers and store them in the

// array elements one and two.Calculate the answer and store it in element three.Return nothing.NOTE – Case zero

// will just write zeros to elements one, two and three – should already have zero in element zero.

//In the main outside the switch (but inside the sentinel loop), read in the 4 - element array data and print the

// results to the screen – i.e.you chose to “add” element[1] to element[2] and the result is element[3].

//

// %%%% Algorthim %%%%

//

// preprocessor directives

// uf def statements

// start main

// init variables

// while

// call uf

// switch

// case (0-4)

// set element 0 to operation code

// call specific uf

// default case

// just break

// end switch

// if operation code is >= 0

// print element values

// end if

// end while

// end main

//

// mmmm uf mmmm

//

// init variable

// while

// menu options and read in number

// if

// return num

// end if

// error check

// end while

//

// 0000 uf0 0000

//

// set all elements of array to 0

//

// 1234 uf(1-4) 1234

//

// prompt user to enter in doubles and read them in elements 1 and 2

// math operation for element 3

//

#include <stdio.h>

int uf();

void uf0(double A[]);

void uf1(double A[]);

void uf2(double A[]);

void uf3(double A[]);

void uf4(double A[]);

void main(void)

{

int n = 0;

double A[4];

while (n >= 0)

{

n = uf();

switch (n)

{

case 0:

{

A[0] = (double)n;

uf0(A);

break;

}

case 1:

{

A[0] = (double)n;

uf1(A);

break;

}

case 2:

{

A[0] = (double)n;

uf2(A);

break;

}

case 3:

{

A[0] = (double)n;

uf3(A);

break;

}

case 4:

{

A[0] = (double)n;

uf4(A);

break;

}

default:

{

puts("");

break;

}

}

if (n >= 0)

printf("\nElement[1] = %lf\nElement[2] = %lf\nElement[3] = %lf\n", A[1],A[2],A[3]);

}

}

int uf()

{

int n = 0;

while (true)

{

puts("Enter a 0 for a no operation (NoOp) option.");

puts("Enter a 1 to do addition.");

puts("Enter a 2 to do subtraction.");

puts("Enter a 3 to do multiplication.");

puts("Enter a 4 to do division.");

puts("Enter a negative number to Quit sentinel loop.");

scanf\_s("%d", &n);

if (n <= 4)

return n;

else

puts("ERROR - Enter a valid number.");

}

}

void uf0(double A[])

{

A[1] = A[2] = A[3] = 0;

}

void uf1(double A[])

{

printf("Enter a double number:");

scanf\_s("%lf", &A[1]);

printf("\nEnter another double number:");

scanf\_s("%lf", &A[2]);

A[3] = A[1] + A[2];

}

void uf2(double A[])

{

printf("Enter a double number:");

scanf\_s("%lf", &A[1]);

printf("\nEnter another double number:");

scanf\_s("%lf", &A[2]);

A[3] = A[1] - A[2];

}

void uf3(double A[])

{

printf("Enter a double number:");

scanf\_s("%lf", &A[1]);

printf("\nEnter another double number:");

scanf\_s("%lf", &A[2]);

A[3] = A[1] \* A[2];

}

void uf4(double A[])

{

printf("Enter a double number:");

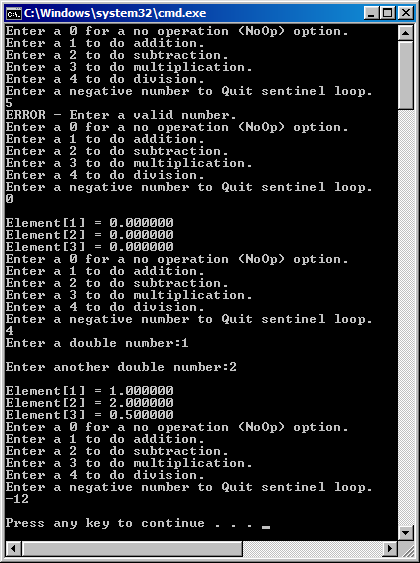
scanf\_s("%lf", &A[1]);

printf("\nEnter another double number:");

scanf\_s("%lf", &A[2]);

A[3] = A[1] / A[2];

}



//

// HWK9(6).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Redo the Calculator program, but only pass pointers to the functions and return nothing.

// So, if a function returned an int, pass an (additional pointer variable) integer pointer to function and use it

// in function and in main.

//

// %%%% Algorthim %%%%

//

// copy and paste last problem

// pass &n to uf

// change uf to void

// have uf change \*n (the data value of n)

// no changes needed in other uf's since arrays are pointers

//

#include <stdio.h>

#pragma warning(disable : 4996)

void uf(int \*n);

void uf0(double A[]);

void uf1(double A[]);

void uf2(double A[]);

void uf3(double A[]);

void uf4(double A[]);

void main(void)

{

int n = 0;

double A[4];

while (n >= 0)

{

uf(&n);

switch (n)

{

case 0:

{

A[0] = (double)n;

uf0(A);

break;

}

case 1:

{

A[0] = (double)n;

uf1(A);

break;

}

case 2:

{

A[0] = (double)n;

uf2(A);

break;

}

case 3:

{

A[0] = (double)n;

uf3(A);

break;

}

case 4:

{

A[0] = (double)n;

uf4(A);

break;

}

default:

{

puts("");

break;

}

}

if (n >= 0)

printf("\nElement[1] = %lf\nElement[2] = %lf\nElement[3] = %lf\n", A[1], A[2], A[3]);

}

}

void uf(int \*n)

{

while (true)

{

int temp = 0;

puts("Enter a 0 for a no operation (NoOp) option.");

puts("Enter a 1 to do addition.");

puts("Enter a 2 to do subtraction.");

puts("Enter a 3 to do multiplication.");

puts("Enter a 4 to do division.");

puts("Enter a negative number to Quit sentinel loop.");

scanf("%d", &\*n);

if (\*n <= 4)

{

break;

}

else

puts("ERROR - Enter a valid number.");

}

}

void uf0(double A[])

{

A[1] = A[2] = A[3] = 0;

}

void uf1(double A[])

{

printf("Enter a double number:");

scanf\_s("%lf", &A[1]);

printf("\nEnter another double number:");

scanf\_s("%lf", &A[2]);

A[3] = A[1] + A[2];

}

void uf2(double A[])

{

printf("Enter a double number:");

scanf\_s("%lf", &A[1]);

printf("\nEnter another double number:");

scanf\_s("%lf", &A[2]);

A[3] = A[1] - A[2];

}

void uf3(double A[])

{

printf("Enter a double number:");

scanf\_s("%lf", &A[1]);

printf("\nEnter another double number:");

scanf\_s("%lf", &A[2]);

A[3] = A[1] \* A[2];

}

void uf4(double A[])

{

printf("Enter a double number:");

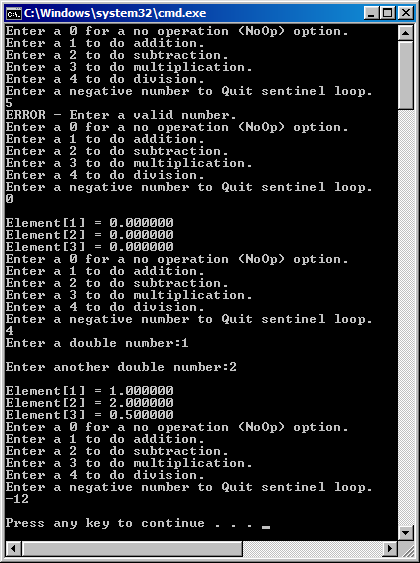
scanf\_s("%lf", &A[1]);

printf("\nEnter another double number:");

scanf\_s("%lf", &A[2]);

A[3] = A[1] / A[2];

}



//

// HWK9(7).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Write a program with a function to determine the smallest number of $20, $10, $5 and $1

// necessary to pay a user entered amount. Prompt the user to enter in an amount unsigned int. Read in the value.

// Function – Pass the unsigned int value(amount), a pointer to the number of $20 bills needed, a pointer to the

// number of $10 bills needed, a pointer to the number of $5 bills needed, and a pointer to the number of $1 bills

// needed.Return nothing.

// In the main, print out the needed number of $20, $10, $5, and $1 needed to pay the amount entered.

//

// %%%% Algorthim %%%%

//

// preprocessor directives

// start main

// init variables

// prompt user to enter and scan

// call uf

// print values

// end main

//

// 1111 uf1 1111

//

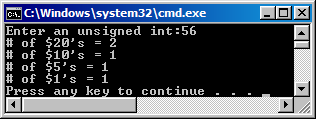
// set data values the pointers point to to n/(bill #)

// reset n after each bill to n%(bill#)

#include <stdio.h>

#pragma warning(disable : 4996)

void uf(int n, int \*twens, int \*tens, int \*fs, int \*os);



void main(void)

{

unsigned int n = 0;

int twentys = 0;

int tens = 0;

int fives = 0;

int ones = 0;

printf("Enter an unsigned int:");

scanf("%d", &n);

uf(n, &twentys, &tens, &fives, &ones);

printf("# of $20's = %d\n# of $10's = %d\n# of $5's = %d\n# of $1's = %d\n", twentys, tens, fives, ones);

}

void uf(int n, int \*twens, int \*tens, int \*fs, int \*os)

{

\*twens = n / 20;

n = n % 20;

\*tens = n / 10;

n = n % 10;

\*fs = n / 5;

n = n % 5;

\*os = n / 1;

n = n % 1;

}

//

// HWK9(8).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Write a C program that defines a float

// number and initialize it to 2.111111. Define a pointer to

// a float and initialize the pointer to the address of the

// float number. Define an integer number and initialize it to 3.

// Define a pointer to an integer number and initialize it to

// the integer number address. Define a double variable and

// initialize it to zero. Define a pointer to a double and

// initialize it to the address of the double variable.

// The program will print out the value of the float number with

// only 3 decimal places. Then print out the address of the float

// number using both the address operator and the pointer variable.

// The program will print out the value of the integer number twice in

// a field of 10 spaces: once left justified and once right justified.

// Then print out the integer number's address using both the address

// operator and the pointer variable.

// Write an assignment operator where the product is equal to the float

// number times the integer number. Print out the value of the product

// as a double. Then set the product to -99.999999. Again, print out

// the product in a field of 20 spaces with two decimal characters

// and right justified. Lastly, set the product equal to the product

// divided by the integer number - but only use the pointer variables.

//

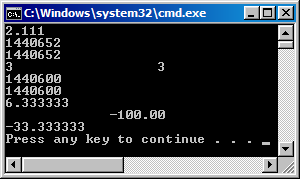
// %%%% Algorthim %%%%

//

// follow the problem statement (just init variables, print stuff and do caculations in certain ways)

//

#include <stdio.h>

#pragma warning(disable : 4996)

void main(void)

{

float f = 2.111111;

float \*fPtr = &f;

int n = 3;

int \*nPtr = &n;

double d = 0;

double \*dPtr = &d;

printf("%.3f\n", f);

printf("%d\n%d\n", &f, fPtr);

printf("%-10d%10d\n", n, n); //if it was meant to have both in one field of 10 spaces, the 10's would be changed to 5's

printf("%d\n%d\n", &d, dPtr);

d = (double)f \* (double)n;

printf("%lf\n", d);

d = -99.999999;

printf("%20.2lf\n", d);

d = \*dPtr / (double)\*nPtr;

printf("%lf\n", d);

}

//

// HWK9(9).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Write a C program that compares passing

// variables to a function, verses passing pointers.

// Define a function which will be passed two floats and return a double.

// The function will add the two floats and return the sum.

// Define a function which will be passed two float pointers,

// a double pointer and return nothing. The function will

// add the two float values and store the result in the

// double address using only pointer variables.

// Define two float variables and a double variable,

// initialize all floats variables to 0.0.

// initialize the double to -99.00.

// Define two float pointers and a double pointer,

// initialize all pointers to NULL.

// Prompt the user to enter in two float numbers to add together.

// Read in the two float numbers.

// assign all pointers with the address for their corresponding variable

// call the first function - pass by value.

// print the result to the screen.

// Prompt the user to enter in two float numbers to add together.

// Read in the two float numbers.

// call the second function - pass by reference.

// print the result to the screen using the pointer variable.

//

// %%%% Algorthim %%%%

//

// follow the problem statement (just init variables, call functions, write functions, print stuff and do caculations in certain ways)

//

#include <stdio.h>

#pragma warning(disable : 4996)

double uf1(float f1, float f2);

void uf2(float \*f1, float \*f2, double \*d);

void main(void)

{

float f1 = 0.0;

float f2 = 0.0;

double d = -99.0;

float \*f1Ptr = NULL;

float \*f2Ptr = NULL;

double \*dPtr = NULL;

printf("Enter in a float number to be added:");

scanf("%f", &f1);

printf("\nEnter in another float number to be added:");

scanf("%f", &f2);

f1Ptr = &f1;

f2Ptr = &f2;

dPtr = &d;

printf("%lf", uf1(f1, f2));

printf("\nEnter in a float number to be added:");

scanf("%f", &f1);

printf("\nEnter in another float number to be added:");

scanf("%f", &f2);

uf2(f1Ptr, f2Ptr, dPtr);

printf("%lf", \*dPtr);

}

double uf1(float f1, float f2)

{

double d = 0.0;

d = (double)f1 + (double)f2;

return d;

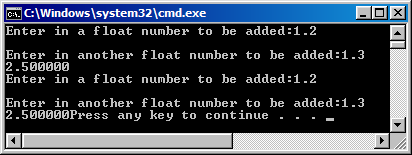
}

void uf2(float \*f1, float \*f2, double \*d)

{

\*d = (double)\*f1 + (double)\*f2;

}



//

// HWK9(10).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Write a C program that only passes pointers to

// to its functions. Do not use SIZE or array indexing in any function.

// The program creates an integer array of SIZE 10 and initializes

// the array to zero. Calls a function to fill the array with random

// integers between 0 and (SIZE/2 - 1). Calls a function to print the array

// values. Prompts the user to enter in an integer between 0 and (SIZE/2 - 1)

// to search for; reads in the integer value to search the array for. Calls

// a function to search the array for a match to the entered integer,

// when a match is found, it prints the element number in the array

// where the match was found.

//

// %%%% Algorthim %%%%

//

// preprocessor directives

// start main

// init variables

// init array

// call uf1

// call uf2

// prompt user to enter and read in

// call uf3

// end main

//

// 1111 uf1 1111

//

// using pointers to the starting and ending addresses of the array:

// while

// set array values to rand num (starting with element 0)

// increment start

// end while

//

// 2222 uf2 2222

//

// using pointers to the starting and ending addresses of the array:

// while

// print array value in element number (starting with element 0)

// increment start

// end while

// 2222 uf2 2222

//

// using pointers to the starting and ending addresses of the array:

// init variable

// while

// if

// print element #

// break

// end if

// increment start

// increment element

// end while

#include <stdio.h>

#include <stdlib.h>

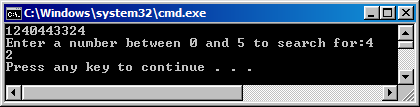
#pragma warning(disable : 4996)

#define S 10

void uf1(int \*start, int \*end);

void uf2(int \*start, int \*end);

void uf3(int \*start, int \*end, int \*n);



void main(void)

{

int A[S] = {};

int \*start = &A[0];

int \*end = &A[9];

int n = 0;

int \*nPtr = &n;

size\_t i;

for (i = 0; i < S; i++)

{

A[i] = 0;

}

uf1(start, end);

uf2(start, end);

printf("\nEnter a number between 0 and 5 to search for:");

scanf("%d", &n);

uf3(start, end, nPtr);

}

void uf1(int \*start, int \*end)

{

while (start <= end)

{

\*start = rand() % (S / 2);

start++;

}

}

void uf2(int \*start, int \*end)

{

while (start <= end)

{

printf("%d", \*start);

start++;

}

}

void uf3(int \*start, int \*end, int \*n)

{

int element = 0;

while (start <= end)

{

if (\*start == \*n)

{

printf("%d\n", element);

break;

}

element++;

start++;

}

}

//

// HWK9(11).cpp

//

// By: Barak Barclay

// Date: 7 Dec 2015

//

// Problem Statement: Write a C program that compares passing:

// 1) an array and its size to a function, with 2) passing the

// starting and ending pointers to the array. By passing the

// array name (pointer and the array size) you can step through

// the array by incrementing an array index number. When passing

// the starting and ending array pointers, you will step through

// the array by incrementing a temp pointer through the array

// element addresses. Both functions will return nothing - arrays

// pass by reference. Define a float array to have 20 elements - use

// a preprocessor define statement to set the SIZE. Initialize the

// array to 0.0. Initialize the start and ending pointers to the

// correct addresses. Then fill the array with each elements number

// - element 0 = 0, element one = 1, etc. Call the second function

// and print out the array. Prompt the user to enter in an integer

// scaling number from 0 to 10. Call first function (pass the array

// and the array size) and fill the array with random floats

// between 0 and the scaling number. Call a second function

// to print out the array, pass the function the starting array

// pointer and an ending array pointer.

//

// %%%% Algorthim %%%%

//

// preprocessor directives

// start main

// init variables

// init array

// call uf1

// call uf2

// prompt user to enter and read in

// call uf3

// end main

//

// 1111 uf1 1111

//

// using pointers to the starting and ending addresses of the array:

// while

// set array values to rand num (starting with element 0)

// increment start

// end while

//

// 2222 uf2 2222

//

// using pointers to the starting and ending addresses of the array:

// while

// print array value in element number (starting with element 0)

// increment start

// end while

// 2222 uf2 2222

//

// using pointers to the starting and ending addresses of the array:

// init variable

// while

// if

// print element #

// break

// end if

// increment start

// increment element

// end while

#include <stdio.h>

#include <stdlib.h>

#pragma warning(disable : 4996)

#define S 20

void uf1(float A[], size\_t i, int n);

void uf2(float \*start, float \*end);

void main(void)

{

float A[S] = {};

int n = 0;

size\_t i;

for (i = 0; i < S; i++)

{

A[i] = 0.0;

}

float \*start = &A[0];

float \*end = &A[19];

for (i = 0; i < S; i++)

{

A[i] = i;

}

uf2(start, end);

printf("\nEnter a number between 0 and 10 to search for:");

scanf("%d", &n);

uf1(A, i, n);

uf2(start, end);

}

void uf1(float A[], size\_t i, int n)

{

for (i = 0; i < S; i++)

{

A[i] = (float)(rand() % n);

}

}

void uf2(float \*start, float \*end)

{

while (start <= end)

{

printf("%f\t", \*start);

start++;

}

}

