CS 211

Lab03 - Warm-Up Codes

main.c

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

// Define a structure to represent a Fruit
typedef struct Fruit_struct{
   char letter; // First letter of fruit name
   int quantity; // Number of items
   double weight; // Total weight in KG
} Fruit;

// Function to dynamically create a Fruit structure
Fruit* createFruit(char myLet, int myQnt, double myWgt) {
   Fruit* aFrt = (Fruit*)malloc(sizeof(Fruit)); // Allocate memory for Fruit struct
   aFrt->letter = myLet;   // Assign first letter of the fruit
   aFrt->quantity = myQnt; // Assign quantity
   aFrt->weight = myWgt;   // Assign total weight
   return aFrt;            // Return pointer to the allocated struct
}

int main(void) {

   // Initialize pointers to Fruit structures as NULL
   Fruit* aPtr = NULL;
   Fruit* bPtr = NULL;
   Fruit* cPtr = NULL;

   // Create Fruit structs and assign them to pointers
   aPtr = createFruit('a', 6, 1.27); // 6 apples weigh 1.27 kgs
   bPtr = createFruit('b', 5, 0.53); // 5 bananas weigh 0.53 kgs
   cPtr = createFruit('c', 95, 0.76); // 95 cherries weigh 0.76 kgs

   // Print total quantity of all fruits by summing their quantities
   printf("Total #Items = %d\n", aPtr->quantity + bPtr->quantity + cPtr->quantity);

   // Print total weight of all fruits by summing their weights
   printf("Total Weight = %0.2lf\n", aPtr->weight + bPtr->weight + cPtr->weight);

   // Free dynamically allocated memory to avoid memory leaks
   free(aPtr);
   free(bPtr);
   free(cPtr);
```

```
    return 0; // Return 0 to indicate successful execution
}
```

- The pointers are first assigned a NULL value
- Each of the pointers is assigned to different structs that are created with the create Fruit struct (with different values of parameters)
- The sum of the quantities assigned to the pointers is printed out first followed by the sum of the weight appointed to pointers.
- In the end, the pointers are deallocated
-

main2.c

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

// Define a structure to represent a Fruit
typedef struct Fruit_struct{
   char letter; // First letter of fruit name
   int quant;   // Number of items
   double weight; // Total weight in kilograms
} Fruit;

// Function to set values for a Fruit structure
void setFruitVals(Fruit* myFrt, char myLet, int myQnt, double myWgt) {
   myFrt->letter = myLet;   // Assign the first letter of the fruit
   myFrt->quant = myQnt;    // Assign the quantity
   myFrt->weight = myWgt;   // Assign the total weight
   return;                  // Return after setting values
}

int main(void) {
   Fruit* fPtr = NULL; // Initialize pointer TO Fruit structure as NULL

   // Allocate memory for three Fruit structs
   fPtr = (Fruit*)malloc(sizeof(Fruit) * 3);

   // Pointer mPtr is used to the allocated block
   Fruit* mPtr = fPtr;

   // Set values for the first Fruit
   setFruitVals(mPtr, 'a', 6, 1.27); // 6 apples weigh 1.27 kgs
   mPtr++; // Move pointer to the next Fruit in memory

   // Set values for the second Fruit
   setFruitVals(mPtr, 'b', 5, 0.53); // 5 bananas weigh 0.53 kgs
   mPtr++; // Move pointer to the next Fruit in memory

   // Set values for the third Fruit
   setFruitVals(mPtr, 'c', 95, 0.76); // 95 cherries weigh 0.76 kgs

   // Print total quantity of all fruits by accessing them through the pointer array
   printf("Total #Items = %d\n", fPtr[0].quant + fPtr[1].quant + fPtr[2].quant);

   // Print total weight of all fruits by accessing them through the pointer array
   printf("Total Weight = %0.2lf\n", fPtr[0].weight + fPtr[1].weight + fPtr[2].weight);

   // Free dynamically allocated memory to avoid memory leaks
```

```
    free(fPtr);
    return 0; // Return 0 to indicate successful execution
}
```

- The fptr is first assigned a null value
- After that, it's assigned to the first struct in the newly created 3 parallel structs
- After that mptr is assigned to be equal to fptr
- After the function is called with different parameter values and each time the mptr is assigned to the struct, and at the end incremented and assigned to the next struct
- The fptr is then called for the sum of quantities and the sum of weight respectively, which it now prints with different indexes (each index representing the next struct) since all 3 structs contain values
- At the end the pointer is freed (deallocated)

main 2

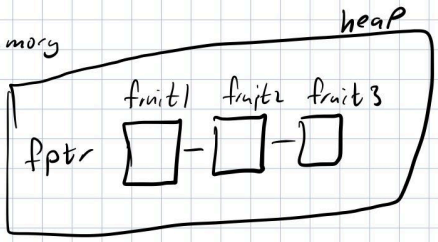
struct passed as pointer to setter function

fpointer is an array of 3 fruits

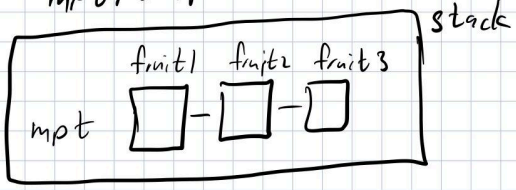
mptr = fpointer in stack memory

values copied

fptr freed



mptr = fptr



free fptr

main

	address	value
aPtr	1	Null
bPtr	2	Null
cPtr	3	Null

after create fruit on all 3

	address	value
aPtr	1	Letter, quantity, weight
bPtr	2	Letter, quantity, weight
cPtr	3	Letter, quantity, weight

after free on all 3

	address	value
aPtr	1	
bPtr	2	
cPtr	3	