Program 1

```
(base) Computers-Air:C computer$ gcc practice.c
(base) Computers-Air:C computer$ ./a.out 2

There are 2 types of donuts in our store today!!!

1.Enter donut type: Eclair
How many are we starting with today? 10
How many do we want to sell? 8

2.Enter donut type: Glazed
How many are we starting with today? 5
How many do we want to sell? 4

***Donut list:***
Eclair: start-10, hope to sell-8
Glazed: start-5, hope to sell-4
```

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

typedef struct donut{

  char *name; /*remember this is JUST A POINTER.  WE NEED TO MALLOC SPACE*/
  int **info;  /*remember this is JUST A POINTER.  WE NEED TO MALLOC SPACE*/

}donut;

/*this function gets user input into our space for donut structs*/
int inventory(donut*c, int num)
{
  int i=0;
  int malloc_check=1;

  while(i<num && malloc_check)
  {
    c->name=malloc(20); /*setting aside space, should check if malloc returns NULL*/
    c->info=malloc(sizeof(int*)*2);
    c->info[0]=malloc(sizeof(int));
    c->info[1]=malloc(sizeof(int));

    if(c->name && c->info && c->info[0] && c->info[1]) //checking if malloc returned null anywhere
    {
        printf("\n%d.Enter donut type: ", (i+1));
        scanf("%s", c->name); /*remember to use fgets if you use a space*/
        getchar();

        printf("How many are we starting with today? ");
        scanf("%d", (c->info[0])); //can use -> or * .

        printf("How many do we want to sell? ");
        scanf("%d", (*c).info[1]);

        c++; /*incrementing to our next struct...remember you could also use the [] notation instead of
 moving the actual pointer!!!*/
        i++;
```

```
    }

    else
    {
      malloc_check=0;
    }
  }

  return malloc_check;
}


/*we are freeing the memory we allocated for each char pointer (allocated in the function above)*/
void free_inventory(donut *c, int num)
{
  int i=0;

  for(int i=0; i<num;i++)
  {
    free(c->name);
    free(c->info[0]);//notice we are free-ing from the inside out...
    free(c->info[1]);
    free(c->info);
    c++;
  }
}


/*print out donuts and number to screen*/
void print_donuts(donut *c, int num)
{
  int i;

  printf("\n***Donut list:***\n");

  for(i=0;i<num;i++)
  {
    printf("%s: start-%d, hope to sell-%d\n", c->name, *(c->info[0]), *(c->info[1]));
    c++;
  }
}


int main (int argc, char **argv)
{
  printf("\nThere are %s types of donuts in our store today!!!\n", argv[1]); /*using the second command
 line arg*/
  int num=atoi(argv[1]);
  donut *store=malloc(sizeof(donut)*num);  /*remember store is just a pointer (8 bytes).  The address
 held inside is the address of a block of memory.  This block of memory has as many bytes as
 sizeof(donut)*num  */

  inventory(store, num);
```

```
  print_donuts(store,num);
  free_inventory(store,num);

  free(store); /*don't forget to free all memory allocated*/

}
```

USING VALGRIND:

Remember I mentioned about memory leaks before? One way to check is a program called Valgrind. It is on Omega (you could also download it and other programs to check...just using the one on Omega)

```
[fiq8745@omega ~]$ vim donuts.c
[fiq8745@omega ~]$ gcc donuts.c
donuts.c: In function 'free_inventory':
donuts.c:54: error: redefinition of 'i'
donuts.c:52: error: previous definition of 'i' was here
donuts.c:54: error: 'for' loop initial declaration used outside C99 mode
[fiq8745@omega ~]$ gcc std=c99 donuts.c
gcc: std=c99: No such file or directory
donuts.c: In function 'free_inventory':
donuts.c:54: error: redefinition of 'i'
donuts.c:52: error: previous definition of 'i' was here
donuts.c:54: error: 'for' loop initial declaration used outside C99 mode
[fiq8745@omega ~]$ gcc -std=c99 donuts.c
[fiq8745@omega ~]$ ./a.out

There are (null) types of donuts in our store today!!!
Segmentation fault
[fiq8745@omega ~]$ valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --track-origins=yes --
num-callers=20 --track-fds=yes ./a.out 2
==11161== Memcheck, a memory error detector
==11161== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==11161== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==11161== Command: ./a.out 2
==11161==

There are 2 types of donuts in our store today!!!

1.Enter donut type: Eclair
How many are we starting with today? 5
How many do we want to sell? 5

2.Enter donut type: Sprinkle
How many are we starting with today? 10
How many do we want to sell? 9

***Donut list:***
Eclair: start-5, hope to sell-5
Sprinkle: start-10, hope to sell-9
==11161==
==11161== FILE DESCRIPTORS: 3 open at exit.
==11161== Open file descriptor 2: /dev/pts/30
==11161==    <inherited from parent>
==11161==
==11161== Open file descriptor 1: /dev/pts/30
==11161==    <inherited from parent>
==11161==
==11161== Open file descriptor 0: /dev/pts/30
==11161==    <inherited from parent>
==11161==
==11161==
==11161== HEAP SUMMARY:
==11161==     in use at exit: 0 bytes in 0 blocks
==11161==   total heap usage: 9 allocs, 9 frees, 120 bytes allocated
==11161==
==11161== All heap blocks were freed -- no leaks are possible
```

```
==11161==
==11161== For counts of detected and suppressed errors, rerun with: -v
==11161== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```

No errors on the code I did!

If I adjust the code and remove the *free_inventory* function:

```
[fiq8745@omega ~]$ vim donuts.c
[fiq8745@omega ~]$ gcc -std=c99 donuts.c
[fiq8745@omega ~]$ valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --track-origins=yes --
num-callers=20 --track-fds=yes ./a.out 2
==12169== Memcheck, a memory error detector
==12169== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==12169== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==12169== Command: ./a.out 2
==12169==

There are 2 types of donuts in our store today!!!

1.Enter donut type: Eclair
How many are we starting with today? 10
How many do we want to sell? 7

2.Enter donut type: Sprinkle
How many are we starting with today? 5
How many do we want to sell? 5

***Donut list:***
Eclair: start-10, hope to sell-7
Sprinkle: start-5, hope to sell-5
==12169==
==12169== FILE DESCRIPTORS: 3 open at exit.
==12169== Open file descriptor 2: /dev/pts/30
==12169==    <inherited from parent>
==12169==
==12169== Open file descriptor 1: /dev/pts/30
==12169==    <inherited from parent>
==12169==
==12169== Open file descriptor 0: /dev/pts/30
==12169==    <inherited from parent>
==12169==
==12169==
==12169== HEAP SUMMARY:
==12169==     in use at exit: 88 bytes in 8 blocks
==12169==   total heap usage: 9 allocs, 1 frees, 120 bytes allocated
==12169==
==12169== 8 bytes in 2 blocks are indirectly lost in loss record 1 of 4
==12169==    at 0x4A0610C: malloc (vg_replace_malloc.c:195)
==12169==    by 0x4006A5: inventory (in /home/f/fi/fiq8745/a.out)
==12169==    by 0x4008F7: main (in /home/f/fi/fiq8745/a.out)
==12169==
==12169== 8 bytes in 2 blocks are indirectly lost in loss record 2 of 4
==12169==    at 0x4A0610C: malloc (vg_replace_malloc.c:195)
==12169==    by 0x4006BE: inventory (in /home/f/fi/fiq8745/a.out)
==12169==    by 0x4008F7: main (in /home/f/fi/fiq8745/a.out)
==12169==
==12169== 40 bytes in 2 blocks are definitely lost in loss record 3 of 4
==12169==    at 0x4A0610C: malloc (vg_replace_malloc.c:195)
==12169==    by 0x400674: inventory (in /home/f/fi/fiq8745/a.out)
==12169==    by 0x4008F7: main (in /home/f/fi/fiq8745/a.out)
==12169==
==12169== 48 (32 direct, 16 indirect) bytes in 2 blocks are definitely lost in loss record 4 of 4
==12169==    at 0x4A0610C: malloc (vg_replace_malloc.c:195)
==12169==    by 0x400688: inventory (in /home/f/fi/fiq8745/a.out)
==12169==    by 0x4008F7: main (in /home/f/fi/fiq8745/a.out)
==12169==
==12169== LEAK SUMMARY:
==12169==    definitely lost: 72 bytes in 4 blocks
==12169==    indirectly lost: 16 bytes in 4 blocks
```

```
==12169==      possibly lost: 0 bytes in 0 blocks
==12169==    still reachable: 0 bytes in 0 blocks
==12169==         suppressed: 0 bytes in 0 blocks
==12169==
==12169== For counts of detected and suppressed errors, rerun with: -v
==12169== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 4 from 4)
```

See Valgrind documentation for more info!

https://valgrind.org/
https://valgrind.org/docs/manual/manual.html

Program 2

[fiq8745@omega ~]$ gcc -o german german_words.c
[fiq8745@omega ~]$ ./german germanwords.txt 7

******
Longest word is: Donaudampfschiffahrtsgesellschaftskapitaen
Length is: 42
******

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

/*this function looks for the longest word*/
void find_word(int n, char** words)
{
	int i=0, hold=0, current=0, hold_index=0;
	while(i<n)
	{
		current=strlen(words[i])-1; /*get rid of newline given by fgets*/

		if(hold<current)
		{
			hold=current;
			hold_index=i;
		}
		i++;
	}
	printf("\n******\nLongest word is: %sLength is: %d\n******\n\n", words[hold_index],hold);

}

/*this function reads in the file*/
void read_file(char ** words, char *filename)
{
```

```
        int i=0;
        FILE *fp=fopen(filename, "r");

        while(i<7) /*I hardcoded 7-you could have passed it in as a parameter (number of words)*/
       {
            words[i]=malloc(sizeof(char)*100); /*remember each element of words (words[i]) is a char pointer
so we allocate 100 bytes and pointer returned by malloc is held in words[i] (each element)*/
            if(words[i]==NULL)
            {
                printf("Note: Memory not allocated.\n");/*my program won't handle this-just showing you*/
                break;
            }

            fgets(words[i], 100, fp); /*read a line of the file into the memory allocated above (address held in
words[i])*/
            i++;

       }
        fclose(fp); /*close your file*/


}

int main(int argc, char **argv)
{


        char *w;
        int i=0, num=0;
        char **words=(char**)malloc(sizeof(char*)*7); /*malloc returns a pointer that holds the address of a
block of space for pointers-meaning each element of words is a pointer (see the read_file function)*/
        num=atoi(argv[2]);
        read_file(words, argv[1]);
        find_word(num, words);

        while(i<num)
        {
                free(words[i]); /*free all the memory we allocated in read_file*/
                i++;
        }

        free(words); /*free the pointer we allocated above*/
}
```

Note: the values of the fruit in the sample run and the fruit.txt file I am uploading may vary since the program constantly modifies the file each time it runs

Program 3

```
[fiq8745@omega ~]$ gcc -o fruit fruit.c
[fiq8745@omega ~]$ ./fruit 6 fruits.txt

Options:
1. Sell/Get Shipment
2. Exit
1
Are you selling (s) or getting a shipment (g)? s
Which fruit are you updating? grape
How many are you selling? 1

Updated grape: 1

Options:
1. Sell/Get Shipment
2. Exit
1
Are you selling (s) or getting a shipment (g)? g
Which fruit are you updating? grape
How many are in the shipment? 20

Updated grape: 21

Options:
1. Sell/Get Shipment
2. Exit
2

Exiting...
```

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

typedef struct fruit{
        char *name; /*this is JUST A POINTER.  We will need to allocate memory to hold the name*/
        int inventory;
}fruit;

/*read the file*/
void read_fruits(fruit*f, int size, char*filename)
{

        FILE *fp=fopen(filename,"r+");
        char *token;
        char *line=malloc(40); /*we will just use this to hold each line of the file-before we would use something like char line[100]*/
```

```
	int i;


	for(i=0;i<size;i++)
	{
		fgets(line, 40, fp); /*hold a line of the file, for example: banana,2*/
		token=strtok(line,",");
		f->name=malloc(20); /*set aside space for the name-remember all we have in the struct is a
pointer*/
		strcpy(f->name,token);

		token=strtok(NULL,"\n");
		f->inventory=atoi(token);
		f++;

	}
	free(line); /*free line-all we needed it for was to hold each line of the file before we used strtok*/
	fclose(fp);
}

/*This function will update our fruit inventory*/
/*Note: not checking if letter is valid OR taking away too much*/
void update(fruit *f, char c, int size)
{
	char *answer=malloc(20);
	printf("Which fruit are you updating? ");
	scanf("%s", answer);
	int i;

	for(i=0;i<size;i++)
	{

		if(strcmp(f->name, answer)==0) /*found the right struct to update*/
		{
			break;
		}
		f++;
	}

	if(c=='s') /*selling means taking away from inventory-subtraction*/
	{
		printf("How many are you selling? ");
		scanf("%s", answer);
		f->inventory-=atoi(answer);
		printf("\nUpdated %s: %d\n", f->name, f->inventory);
	}

	else /*adding in*/
	{
		printf("How many are in the shipment? ");
		scanf("%s", answer);
		f->inventory+=atoi(answer);
```

```
            printf("\nUpdated %s: %d\n", f->name, f->inventory);
        }

        free (answer); /*we just used answer for this function-we're done now*/

}

/*when we exit the program, we want to write to file our current state (current inventory of each fruit) and
also free up the memory we allocated in read file for the names of the fruits since we are done with them after
we write them*/
void exit_prog(fruit*f, int size, char*filename)
{
        FILE *fp=fopen(filename,"r+");
        int i;
        for(i=0;i<size;i++)
        {
                fprintf(fp, "%s,%d\n", f[i].name,f[i].inventory); /*write to file current status*/
                free(f[i].name); /*free memory after*/
        }

        fclose(fp); /*close file*/
}

int main (int argc, char **argv)
{
        int num=atoi(argv[1]);
        int check=1;
        fruit* fruit_stand=malloc(sizeof(fruit)*num); /*this will set aside enough space for the number of fruits
we give on the command line*/
        char *answer=malloc(20);

        if(fruit_stand==NULL)
        {
                printf("Error getting you some memory!  Bye.\n");
        }

        else
        {
                read_fruits(fruit_stand, num, argv[2]);

                while(check)
                {
                        printf("\nOptions:\n");
                        printf("1. Sell/Get Shipment\n");
                        printf("2. Exit\n");

                        scanf("%s", answer);
                        int n=atoi(answer);

                        if(n==1)
                        {
```

```
                        printf("Are you selling (s) or getting a shipment (g)? "); /*not checking if valid
here*/
                        scanf("%s", answer);
                        update(fruit_stand, answer[0], num);


                }

                else if(n==2)
                {
                        printf("\nExiting...\n");
                        check=0;
                        exit_prog(fruit_stand, num, argv[2]);
                        free(fruit_stand);
                        free(answer);
                }

                else
                {
                        printf("Not a valid choice.");
                }


        }

    }



}
```

USING VALGRIND:

```
[fiq8745@omega ~]$ valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --track-origins=yes -
-num-callers=20 --track-fds=yes ./fruit 6 fruits.txt
==26495== Memcheck, a memory error detector
==26495== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==26495== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==26495== Command: ./fruit 6 fruits.txt
==26495==

Options:
1. Sell/Get Shipment
2. Exit
1
Are you selling (s) or getting a shipment (g)? s
Which fruit are you updating? grape
How many are you selling? 1

Updated grape: 20
```

```
Options:
1. Sell/Get Shipment
2. Exit
1
Are you selling (s) or getting a shipment (g)? g
Which fruit are you updating? grape
How many are in the shipment? 1

Updated grape: 21

Options:
1. Sell/Get Shipment
2. Exit
2

Exiting...
==26495==
==26495== FILE DESCRIPTORS: 3 open at exit.
==26495== Open file descriptor 2: /dev/pts/11
==26495==    <inherited from parent>
==26495==
==26495== Open file descriptor 1: /dev/pts/11
==26495==    <inherited from parent>
==26495==
==26495== Open file descriptor 0: /dev/pts/11
==26495==    <inherited from parent>
==26495==
==26495==
==26495== HEAP SUMMARY:
==26495==     in use at exit: 0 bytes in 0 blocks
==26495==   total heap usage: 13 allocs, 13 frees, 1,452 bytes allocated
==26495==
==26495== All heap blocks were freed -- no leaks are possible
==26495==
==26495== For counts of detected and suppressed errors, rerun with: -v
==26495== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```