



1.) [16 points] Define and explain the following terms (compare and contrast means give at least one similarity and one difference):

- a) Compare and contrast the way space is allocated through the use of the stack and the use of the heap.

**COMPARE**

Both are ways to create space for and store values for data (variables) needed during the execution of the program

**CONTRAST**

STACK: usually automatically allocated as the program enters the scope of that variable

OR freed (or released) back to be used again on the stack automatically by the program

OR size must be fixed before run time

OR a variable that is block scope will be deallocated even if you have a pointer to that space outside of the scope in which that variable was created

HEAP: needs to be explicitly allocated

OR needs to be explicitly deallocated

OR size can be determined at runtime rather than before

OR a pointer to space on the heap can continue to be accessed by a pointer that exists outside of the function/scope where the memory was allocated

- b) List two things that *should* be in header files, and two things that *shouldn't*.

1. Two things that SHOULD be in a header file:

prototype of a function who needs to be reached from more than one file

type definitions for structures, unions, etc

2. Two things that SHOULD NOT be in a header file:

definition of functions

prototypes of functions that should only be reached from one file

- c) Compare and contrast **`stderr`** and **`stdout`**.

**COMPARE**

both are output streams that can be written to using the I/O library functions of C

OR both are already open for output – they do not need to be explicitly opened before they can be used

**CONTRAST**

`stderr` is often sent down the stream faster because it isn't buffered the same way

OR `stderr` is redirected with `>&` while `stdout` is redirected with `>` or `>>`

OR `stderr` is often not redirected to a file when `stdout` is because it will contain messages you want the user to see immediately

- d) Functions may be defined with array dimensions omitted for example:

**`int func1(int arr[][5]);`** or **`int func2(int arr[]);`**

Explain why some of the dimensions (the 5 in this example) need to be specified and why others don't.

**Why the 1<sup>st</sup> is not required**

The first dimension is not required because it can calculate the position of the next argument based on the type of the array

OR the first argument is not required because it is the same as a pointer to an element of the type indicated before the array name

**Why the others are**

In order to calculate the position of the next element in the other dimensions, it needs to know how long that array is

OR the type is actually a pointer to a single dimension array not to an individual element

[15 points] Give the exact output that would be produced by the following code. You do not need to worry about the exact location of any whitespace characters in your output since none of the field width specifiers are given. You do need to make sure you include the line breaks appropriately.

<pre>#include &lt;stdio.h&gt; #define ARR_SIZE 12 typedef struct{     int size;     int *arr; }SType; int main(void){     char name[ARR_SIZE] = "Bob";     int i;     SType s1;     int *ipntr;     char *cpntr1;     char *cpntr2;     SType *spntr;      cpntr1 = name;     s1.arr = calloc(3,sizeof(int));     s1.size = 3;     for (i = 1; i &lt; s1.size; i++){         *(s1.arr + i) = i + 10;     }     printf("%d, %d-%d-%d\n",s1.size,         s1.arr[0],s1.arr[1],s1.arr[2]);</pre>	<pre>    cpntr1 = name;     cpntr2 = malloc(strlen(name)+1);     cpntr2 = name;     *cpntr1 = 'S';     printf("%s %s %s\n",         name,cpntr1,cpntr2);     printf("%s %s %s\n",         name+1,cpntr1+2,cpntr2+3);      spntr = &amp;s1;     spntr -&gt; size = 1;     *(spntr -&gt; arr) = 23;     printf("%d, %d-%d-%d\n",s1.size,         s1.arr[0],s1.arr[1],s1.arr[2]);      return 0; }</pre>
---	---

---

3, 0-11-12

---

Sob Sob Sob

---

ob b

---

1, 23-11-12

---



---



---



---



---



---



---



---

2.) [15 points] Questions about the projects.

- a) Two of the registers in project 1 were special purpose registers (R0 and R1). Describe the purpose of R1 and how it would be updated when performing different elements of the instruction set.

**Purpose of R1 is the Program counter to keep track of the next instruction to be executed**

In most cases it is just incremented to the next instruction in the list

In the base of the Branch and BNN instruction it must be updated directly

- b) Write the portion of the hash function which will look at the last four bits (rightmost 4 bits) of the unsigned integer hashval. If any of the last four bits is a 1, it should shift the hashval 4 to the right. If none of the last four bits is a 1, it should shift the hashval 4 to the left.

**CODE**

```
unsigned int val = 0x000f;
if ((hashval & val) == 0)
    hashval >> 4;
else
    hashval << 4;
```

- c) Why are hash tables often resized before they are completely full? (as we did in project 2)

It is resized before it is completely full to avoid having to deal with the conflicts that are more likely to occur as the table gets closer to being full.

3.) [18 points] Use the structure on the final page for the definitions of the types used here - that page can be torn off, but make sure you write your name on it and submit it with your exam paper. For each of the following questions, the first line gives the declaration of a variable, and the second is an expression using that variable.

i) You may assume any space that would have had to be dynamically allocated has been.

ii) You may also assume that if it is a pointer it is indeed pointing to valid space

iii) You need to fill in the first blank to say if that expression would be valid or invalid

iv) If you put valid in the first column for a given item, in the second column you must then tell the type the expressions refers to. You can think of this in terms of how you would print the item – if it is something you would print using a %s, then say string, if it is something you would print using a %c then say character, etc.

	VALID/INVALID	TYPE (if valid)
a) PersonTy p; p.list->color[1]	_____VALID_____	_____char_____
b) Personty *m; m->name	_____VALID_____	_____string (char *)_____
c) InventoryTy j; j.invlst->peryard	_____VALID_____	_____float_____
d) FabricTy f; *(f.color)	_____VALID_____	_____char_____
e) InventoryTy *x; &(x.storename[0])	_____invalid_____	_____
f) PersonTy *p; p->list->color	_____VALID_____	_____string (char *)_____

4.) [21 points] Use the types defined on the last page of this exam to write each of the following functions. The last page of the exam can be torn off the exam, but make sure you put your name on that paper and submit it with your exam when you are finished. You must assume the prototype given is already present (in the .h file), and you must give the complete implementation that would appear in the corresponding .c file (including the function header). For any functions that require data to be already filled in, you may assume the space has been allocated correctly and valid information has already been filled in.

- a) A function that calculates the cost for this customer needs to return the amount of money the customer order will be based on their list of purchased fabrics. If it can't be calculated, return -1.

```
float OrderCost(PersonTy);
```

```
float ordercost(PersonTy p){
    int j;
    float total = 0;
    for (j = 0; j < p.numofitems; j++)
        total += p.list[j].numberofyards * p.list[j].peryard;
    return total;
}
```

- b) A function that copies an individual FabricTy object. Make sure the original is not harmed during the copy process, and make sure the newly created one is passed back through the return value of the function. Also, if the color of the original is "red" change it to "scarlet" in the copy. If unsuccessful, return NULL.

```
FabricTy *copyFabric(FabricTy);
```

```
FabricTy * copyFabric(FabricTy f){
    const char * col = (strcmp (f.color,"red")? f.color, "scarlet");
    FabricTy * fp = calloc(sizeof(FabricTy),1);
    if (!fp) return NULL;
    *fp = f;
    if (!(fp -> fabricname = strdup(f.fabricname))) {
        free(fp);
        return NULL;
    }
    if (!(fp -> color = strdup(col))) {
        free (fp -> fabricname);
        free(fp);
        return NULL;
    }
    return fp;
}
```

- c) Read from the input stream named to fill the FabricTy name and color. The number of yards and cost should be set to 0. The name can be multiple words such as "synthetic alpaca wool" but the color will be one and exactly one word (as separated by spaces). There will be at least two words separated by a space on the input line. The whole input line will not be more than 80 characters. You may assume the file passed is already open for reading and is not at the end of file.

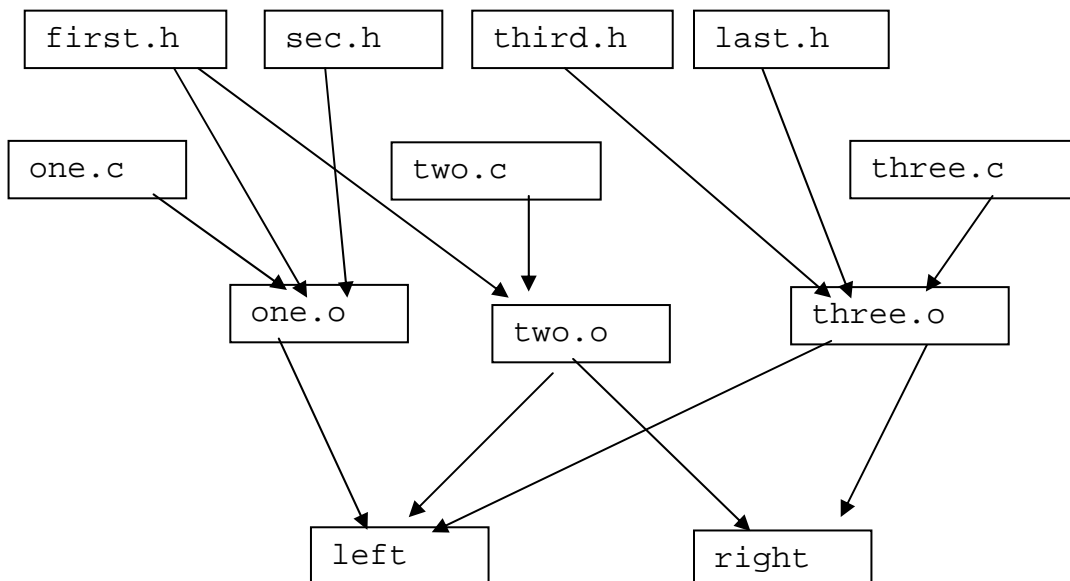
```
FabricTy *fillFabric(FILE *);
```

```
FabricTy * fillFabric(FILE * f){  
    char line[90], *delim;  
    FabricTy *fp = calloc(1, sizeof(FabricTy));  
    if (!fp) return NULL;  
    fgets(line, 80, f);  
    delim = strchr(line, ' ');  
    *delim = '\0';  
    delim++;  
    if (!fp->fabricname = strdup(line)) return NULL;  
    if (*fp -> color = strdup(delim)) return NULL;  
    return fp;  
}
```

5.) [15 Points] UNIX and Make

- a) Given the diagram below, write the Makefile such that it follows all of the following rules:
1. It creates both executables if just the command make is typed.
  2. It uses macros where appropriate.
  3. It uses implied dependencies or rules someplace
  4. It compiles so that all warnings are treated as errors and the ansi standards are enforced.

CC = gcc
CFLAGS = -wall -werror -ansi
all: left right
right: two.o three.o
\$(CC) \$(CFLAGS) -o right two.o three.o
left: one.o two.o three.o
\$(CC) \$(CFLAGS) -o left one.o two.o three.o
one.o: first.h sec.h
two.o: first.h
three.o: third.h last.h



b) Briefly describe the purpose of each of the following UNIX commands

1. **ls** \_\_\_\_\_ **list the children of the current or a named directory**\_\_\_\_\_
2. **cp** \_\_\_\_\_ **copy the contents of a file or directory**\_\_\_\_\_
3. **ln** \_\_\_\_\_ **create a link to a file or directory**\_\_\_\_\_
4. **cd** \_\_\_\_\_ **change to a different current working directory**\_

c) What is the difference between <x> and the "x" when used on a #include line?

the <> indicate the header file is in the standard header location known by the compiler

the " " indicate it is in the location named between the quotation marks

d) What is the purpose of the -o option on the cc compiler?

The -o allows you to name the output of any compilation command

Often used to name the executable file, but it can name the assembly (when used with -E), the object (when used with -c), etc.

This page intentionally blank - You may use it for scratch or to continue an answer. If it is used to continue an answer, make sure you clearly mark it both here and where that question appears in the exam.

Name: \_\_\_\_\_

The types defined on this page will be used for two of the questions on this exam. This page can be torn off the exam paper to make it easier for you to see the definitions while you answer the questions. Make sure you write your name on this paper, put it inside your exam, and submit it with your exam paper when you are finished.

```
#define ARRSIZE 20

typedef char *StrTy;
    /*string dynamically allocated to the space needed at the time*/

typedef struct{
    StrTy fabricname;    /*type of fabric, i.e. wool, silk, etc. */
    StrTy color;        /* color of the fabric, i.e. red, blue, etc. */
    int numberofyards; /* the number of yards purchased or on the bolt */
    float peryard;     /* how much this fabric costs per yard */
} FabricTy;

typedef struct {
    int size;          /* number of different kinds of fabric in inventory */
    FabricTy invlist[ARRSIZE]; /*array of fabric types */
    StrTy storename;  /* name of this store */
}InventoryTy;

typedef struct{
    StrTy name;        /* the name of the customer */
    int numofitems;   /* the number of items this person is purchasing */
    FabricTy *list;   /* the list of items this person is purchasing */
} PersonTy;
```