



1.) [19 points] Answer each of the following in a short paragraph or a short piece of code:

a) Compare and contrast the *struct* and the *union* as it is defined in C

A struct and a union are the same in that both are aggregate types that allow heterogeneous fields – this means that you can create a single variable that is a struct or a union and it has many fields and the fields do not all need to be of the same type.

They are different in that the space allocated for a struct is the sum of all of the individual members because all members exist in that space in the same time, but in the case of a union, there is only enough space allocated for the largest member and it holds only one of its members at a time. In other words a struct stores its fields as if there is an “and” between them, but a union stores its members as if there is an “or” between them.

b) Describe how you would use the term “extern” as a specifier for a variable in a C program. Include what it would mean, where it would likely appear, what else has to appear, and how your program would be different because it is there.

The extern specifier when used on a variable will allow that variable to extend beyond the file scope so that that same variable is available in other files. When the extern specifier is used on a variable that has file scope, it tells the compiler that the actual definition of this variable will come from another file. The actual variable (without the extern keyword) must appear in another file that will be linked in. It allows the object code to be created with the assumption that the actual definition of the variable will be provided at link time instead. If there are two variables of the same name in two different files with neither having the extern keyword, they are independent (unrelated) variables.

- c) Explain the difference between the p and the q as they appear here.

```
int j= 5;
const int *p = &j;   (this line is equivalent to int const *p = &j;)
int* const q = &j;
```

The first line makes the integer constant. It will not allow a change in the value of the integer through the pointer named p, but p can change in value. In other words `*p = 78;` will cause a compilation error but `p = (int*) calloc(sizeof(int),1);` will not cause an error.

The second line makes the pointer constant. It will not allow a change in the value of q itself, but it will allow a change in the value of the integer pointed at. In other words `*q = 78;` will not cause a compilation error but `q = (int*) calloc(sizeof(int),1);` will cause a compilation error.

- d) Write the complete program (main only) that will look at the command line arguments and do the following:
- If there aren't exactly two arguments, it should give an appropriate error message and exit with status -1.
  - If there are exactly two arguments and they are exactly the same length (same number of characters), it should give an appropriate message and exit with status 0.
  - If there are exactly two arguments but they are not exactly the same length, it should give an appropriate message and exit with status 1.

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]){
    if (argc != 3){
        printf("%s needs exactly two arguments\n", argv[0]);
        return -1;
    }
    else{
        printf("%s and %s: ", argv[1], argv[2]);
        if (strlen(argv[1]) == strlen(argv[2])){
            printf("same length\n");
            return 0;
        }
        else{
            printf("different length\n");
            return 1;
        }
    }
}
```

- 2.) [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 since none of the field width specifiers are given. You only need to worry about exactly what values/text appears on which lines.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define ARR_SIZE 12

typedef struct{
    char *name;
    int age;
}SType;

int main(void){
int nums[ARR_SIZE] = {3,5,7,9,2,4,6,8,1};
int *ipntr1,*ipntr2;
char *cpntr1,*cpntr2;
SType *spntr;

    spntr = (SType*) malloc(sizeof(SType));
    spntr->name = (char*)calloc(4,sizeof(char));
    strncpy(spntr->name,"Tim",4);
    spntr->age = 37;

    printf("%d, %d-%d-%d\n",spntr->age,
           *nums, nums[3],*(nums+4));

    cpntr1 = spntr->name;
    cpntr2 = spntr->name + 1;
    ipntr1 = nums;
    ipntr2 = nums+5;
    *cpntr1 = 'J';
    *(cpntr2+1) = 'g';
    printf("%s %s %s\n",spntr->name,cpntr1,cpntr2);
    printf("%s %d %d\n",spntr->name+1,*ipntr1,*ipntr2);
    printf("%d %d\n",ipntr1[2],ipntr2[1]);

    return 0;
}
```

---

37, 3-9-2

Jig Jig ig

ig 3 4

7 6

---

7.) [12 points] Answer the following questions.

- a) Give the exact output of the following assembly code (same assembly as used for projects 1 and 3). Assume the input available is 2 5 1 8 9 10 12 (with one value per line and assume you may not need to use all of the input that is available).

Program:	Output:
li R2 3	
read R3	2
write R3	
neg R2	
write R2	-3
add R3 R3 R0	
add R2 R3 R4	
write R4	-1
read R5	
add R5 R5 R5	10
write R5	

- a) In our application explain the purpose two dedicated registers (R0 and R1). In addition to their purpose, what special restrictions are there to using these registers in instructions.

Register 0 and Register 1 are both special purpose registers in that there is a restriction on if and/or how their values can be changed. R0 is the zero register and it always contains the value 0 – no instruction is allowed to change the value of that register. Instructions are allowed to write to R0, but its value never is allowed to change. R1 is the program counter. It keeps track of which assembly statement will be executed next. The only assembly statements that are allowed to change the register 1 are the branch statements. Other assembly statements are considered an error if they attempt to change register 1.

10.) [18 points] Use the structure on the last 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 need to fill in the blank first to say if that expression would be valid or not valid

ii) If the expression does use the variable name declared in a legal way, you must then also tell the type the expression is or if the expression contains something that is invalid, you must say exactly what make the expression invalid.

**Circle One**

**Either tell the type or tell why it is invalid**

a) PArr p; *((*p).name)	(Valid) / Invalid	char
b) PersonTy h; h.pay.hrlysalary.numofHrs	(Valid) / Invalid	pointer to an integer or int * or array of integers or int[]
c) CompanyTy x; x.p[1].emptye	(Valid) / Invalid	int
d) CompanyTy y; (y.p)->name	(Valid) / Invalid	StrTy
e) PArr z; z.p.pay.weeklysalary	Valid / (Invalid)	Since z is a pointer not a structure or union - it can't be followed by a period
f) PersonTy *x; x.pay	Valid / (Invalid)	Since x is a pointer, it can't be followed by a period

11.) [15 Points] UNIX and Make

- a) Given the partial files listed below, write a complete Makefile to build the program application from the files shown here.

```
main.c:
    #include "helper.h"
    ..

first.c:
    #include "first.h"
    ..

second.c:
    #include "second.h"
    #include "helper.h"
    ..
```

```
application: main.o first.o second.o
    gcc -o application main.o first.o second.o
```

```
main.o: main.c helper.h
    gcc -c main.c
```

```
first.o: first.c first.h
    gcc -c first.c
```

```
second.o: second.c second.h helper.h
    gcc -c second.c
```

- b) What is the purpose of the `-o` when used with the gcc compiler?

The `-o` option allows you to name the output of that compilation.

- c) Give the complete UNIX command line needed to do each of the following tasks.

Display the contents of a file named "this.file": that is stored in your home directory to the screen one screenful at a time	<code>more ~/this.file</code>
Run the executable file named "this" where the argument to the command is "that" and standard input is redirected from the file named "other". Standard output should go to the screen. You know that the file named "this" is in the current directory which is not on your command search path.	<code>./this that &lt; other</code>
Compare the two ASCII text files to see if there are any differences. The names of the two files are "one" and "two". They are both in the current directory.	<code>diff one two</code>
Copy the file named "newone.c" from the 212files space as you have set it up on grace. The new copy should be placed in your home and it should be named "newerone.c".	<code>cp ~/212files/newone.c ~/newerone.c</code>

12.) [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.

- a) This function will write a warning so you know which of your hourly employees have gone over the 33 hours per week (this is when you have to start giving them benefits). The function will traverse the list of employees and if the sum of their hours is over 33, then the function should write their first name followed by their last name on a single line. If the employee is a salary employee or if the hours are not over 33, the function does nothing for that employee. If there are no employees that require the warning, the function should say "none". The function should also return the number of warnings it printed.

```
int overtimewarn(CompanyTy x);

int overtimewarn(Company x){
    int numover = 0, sum = 0;
    for (int j = 0; j < x.count; j++){
        if (x.p[j].emptytype != 0){
            sum = 0;
            for (int k = 0; k<7; k++){
                sum += x.p[j].pay.hrlysalary.numofHrs[k];
            }
            if (sum >33){
                printf("%s %s\n",
                    x.p[j].lname, x.p[j].name);
                numover++;
            }
        }
    }
    if (numover == 0)
        printf("None\n");
    return numover;
}
```

- b) This function will read from an input file named NewSalEmp which should exist in the current directory (make sure you open and test for success exiting with -1 on error). Close the file as appropriate. It will then add that new employee to the list of employees if there is room. If there is no room to add this new employee, the function should print an error message and return -1. The structure of the input file will have the last name on a line by itself and the first name on a line by itself and then the age of the employee on a line by itself (you can assume no extraneous characters anywhere). The weekly salary of this employee should be set to 0 (later to be set by his/her supervisor). If all was successful, the function should return a 0.

```
int addSalEmp(CompanyTy *x);  
  
int addSalEmp(Company *x) {  
    FILE *fp;  
    fp = fopen("NewSalEmp", "r");  
    if (!fp) {  
        perror("NewSalEmp");  
        exit(-1);  
    }  
    if (x->count == 20) {  
        printf("no more room\n");  
        fclose(fp);  
        return -1;  
    }  
    if (!fgets(x->p[x->count].lname, 20, fp) return -1;  
    if (!fgets(x->p[x->count].name, 20, fp) return -1;  
    char line[20];  
    if (!fgets(fp, line, 20) return -1;  
    x->p[x->count].age = atoi(line);  
    x->p[x->count].pay.weeklysalary = 0;  
    x->p[x->count].emptytype = 0;  
    x->count += 1;  
    fclose(fp);  
    return 0;  
}
```

- c) Write the function that through the parameters will have access to two different structures of type PersonTy. If both employees are hourly employees, you want to give all of the hours worked to the first employee without modifying the contents of the second employee. You combine the hours by adding the corresponding places in the numofHrs array (The Sunday hours for both employees get put into the Sunday hours of the first employee, the Monday hours for both employees get put into the Monday hours of the first employee, etc.) The function returns a -1 if it could not make any changes because at least one of the employees was not an hourly worker. Otherwise, the function returns a non-negative number telling the total number of hours moved.

```
int movehours(PersonTy *, PersonTy);
```

```
int movehours(PersonTy *fpntr, PersonTy second){
    int numhours = 0, dhours;
    if (fpntr -> emptytype == 0 || second.emptytype == 0){
        return -1;
    }
    for (int j = 0; j < 7; j++){
        dhours = second.pay.hrlysalary.numofHrs[j];
        numhours += dhours;
        fpntr->pay.hrlysalary.numofHrs[j] += dhours;
    }
    return numhours;
}
```

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[ARRSIZE];

typedef struct{
    int numofHrs[7]; /*number of hours worked each day of the 7 in a week*/
    float perhourrate; /* how much the person gets paid per hour */
} HourlyTy;

typedef union{
    float    weeklysalary; /*weekly salary of a salaried worker*/
    HourlyTy  hrlysalary; /*pay for hourly worker as defined above*/
} WageTy;

typedef struct {
    StrTy name;          /*person's first name */
    StrTy lname;        /*person's last name */
    int  age;           /* age of person in years */
    int  emptytype;    /* 0 for employee on weekly salary or
                       non-zero for hourly employee*/
    WageTy pay;        /* pay as defined above */
} PersonTy;

typedef PersonTy PArr[ARRSIZE];

typedef struct{
    StrTy name;          /* the name of the company the employee list is for */
    PArr  p;            /* the array of employees for this company */
    int  count;         /* the number of employees in this company */
} CompanyTy;
```

This page intentionally blank.