



1.) [16 points] Define and explain the following terms:

a) union (as a C language construct)

A C language construct that defines a set of fields that share the memory they are stored in. Unlike a structure, only one field of a union may be in-use at once.

b) Program Counter

A register in a computer that stores the address of the next instruction to be executed. Normally the program counter automatically advances to the next memory location after the current one, unless a branch instruction is used which changes the program counter to some other location.

c) Bit shift operator

The << and >> operators in C which are used to shift an arbitrary (the second operand) number of bits to the left or right. Bits at the far right of a word (for right shift) are discarded and far left bits get either 0 or copies of the sign bit (implementation dependent).

d) Function prototype

A declaration of a function's name, parameter types, and return value type that proceeds the actual function definition or is included into files that use but do not define the function. The purpose of the prototype is to allow the compiler to check the parameters and return type of function calls.

- 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 text appears on which lines.

```
#include <stdio.h>
#define ARRSIZE 12

typedef struct{
    int size;
    int arr[ARRSIZE];
}SType;

int main(void){
    char name[ARRSIZE] = "Jeff Jones";
    SType s1 = {3, {7, 2, 4}};
    int *ipntr;
    char *cpntr;
    SType *spntr;

    ipntr = s1.arr;
    printf("%d and %d\n", ipntr[1], *ipntr);

    cpntr = &name[5];
    printf("%s\n", cpntr);
    printf("%c and %c\n", *(cpntr + 3), *(cpntr - 4));

    cpntr++;
    printf("%c and %c\n", name[3], cpntr[3]);

    spntr = &s1;
    printf("%d and %d\n", spntr->size, *(spntr->arr + 2));

    return 0;
}
```

2	and	7
Jones		
e	and	e
f	and	s
3	and	4

3.) [15 points] Project 1

- a) In project 1B you needed translate a label use into the memory address where it was defined. How did you handle this (be specific about how you handled labels that were used before being defined and defined before being used)? If you didn't get this working in your project, explain how planned to do it.

Two possible answers (and a couple of hybrids):

My program made an initial pass through the file and looked for all label definitions. For each label definition, a table entry was made recording the label name and its memory location. On a second pass through the program to convert assembly to machine code, when a label use was encountered, the table built during the first pass was consulted and if the label was found, the memory location from the table was used for that instruction.

Alternate answer:

I made a single pass through the program converting assembly to machine language. When a label use was encountered I made an entry in a table noting the memory location and address where it was used. Also during this pass, I recorded all label definitions and their addresses. At the end I went through the list of label uses and looked each one up in the label definition table. If the label was found in this table, I updated the memory address field of the corresponding label use with the appropriate address.

- b) You have been assigned to extend the computer from project #1 with a new instruction that adds a constant (between 0 and 65,535) to the value stored in any one of the 16 registers and then puts the result in any of the registers 2-15. Explain how you would add this instruction to the machine. Make sure to indicate if a new op code is needed or could you extend an existing one like the two versions of Load (if you need a new op-code, recall 9 is unused). Also, explain how the fields of the instruction would be used.

I would need to create a new operand since the add operator already uses three registers, and any 16 bit value could be used in the constant addition (so there are no bits in the memory field to free to indicate the type of add we are doing). I would call my new operation Addi (and assign it op code 9). The r0 field would indicate source register to be added, the r1 field would indicate the register where to put the value, and the memory field would be the value to be added. The r2 field would be un-used.

4.) [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.

a) PersonTy p;  
p -> name[0]

**invalid: p is not a pointer**

b) HourlyTy h;  
h.numofHrs

**valid: array of integers or ptr to a integer**

c) CompanyTy x;  
\*(x.p -> name)

**valid: char**

d) PArr p;  
\*p.age

**invalid: p is an array not a structure**

e) WageTy w;  
w.numofHrs[1]

**invalid: numofHurs is not a field of a w**

f) PersonTy \*x;  
x.pay.weeklysalary

**invaidd: x is a pointer not a structure**

5.) [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) A function that calculates and returns the pay for that one week of the one person passed as the argument.

```
float CalcPay(PersonTy);
```

```
float CalcPay(PersonTy p){
    int hrcount, i;
    if (p.emptytype == 0){
        return (p.pay.weeklysalary);
    }
    else{
        for (i = 0; i < 7; i++){
            hrcount += p.pay.hrlysalary.numofHrs[i]
        }
        return (hrcount * p.pay.hrlysalary.perhourrate);
    }
}
```

- b) A function that prints firstname lastname. You may assume the name is stored (in a good null terminated string) as lastname, firstname. You may also assume there is exactly one comma stored (it should not be printed). void printname(StrTy);

```
void printname(StrTy s){
    char *commapos;
    commapos = strchr(s, ',');
    *commapos = '\0';
    commapos += 2;
    printf("%s %s\n", commapos, s);
    return;
}
```

- c) A function that gives a 3% raise for all company employees who are salaried employees and a 5% raise for all hourly employees.

```
void giveRaises(CompanyTy *);  
void giveRaises(CompanyTy *cpntr){  
    int i;  
    for (i = 0; i < cpntr->count; i++){  
        if (cpntr->p[i].emptype == 0)  
            cpntr->p[i].pay.weeklysalary *= 1.03;  
        else  
            cpntr -> p[i].pay.hrlysalary.perhourrate *= 1.05;  
    }  
}
```

6.) [15 Points] UNIX and Make

- a) Given the partial files listed below, write a Makefile to build the program application from the files file1.c, file2.c, and main.c.

```
stuff.h:                                file1.c:
    #include "helper.h"                  #include "stuff.h"
    ..                                    ..

main.c:                                  file2.c:
    #include "stuff.h"                   #include "helper.h"
    ..                                    ..
```

```
application: file1.o file2.o main.o
    $(CC) -o application file1.o file2.o main.o
```

```
file1.o: stuff.h helper.h file1.c
file2.o: helper.h file2.c
main.o: stuff.h helper.h main.c
```

- b) If you were to run the following sequence of UNIX commands, what is in the file dir1/stuff?

```
cd
mkdir dir1
cp myFile stuff
cd dir1
cp ../otherStuff stuff
```

The contents of the file otherStuff originally in the account's home directory.

- c) What is the difference between < and > when used on a UNIX command line?

< is used to redirect input from a file to a program (change standard input) and > is used to redirect output from a program to file (change standard output).

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 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.