mt. Code=1 Digipen login:_____

1. **Problem** (2 pts):

   What is the output of the following program

```
int main () {
  int x = 12;
  int y = 17;
  x = x ^ y;
  y = x ^ y;
  x = x ^ y;
  printf("%i",x);
  return 0;
}
```

A) 14 B) 16 C) 15 D) none of the listed E) 17 F) 13 G) 12	_____

2. **Problem** (5 * 2 pts):

   Convert C declaration into English

   A) an array of 5 pointers to pointers to int

   B) a pointer to a function taking a pointer to array of 5 ints and returning an int

   C) an array of 5 pointers to functions taking an int and returning an int

   D) a pointer to an array of 5 pointers to functions taking an int and returning an int

   E) a pointer to a function that takes an int and returns an int

   F) a function that takes an int and returns a pointer to an array of 5 pointers to int.

   G) a pointer to an array of 5 pointers to functions that return an int

   H) legal declaration, but not in the list

   I) is a functions taking int and returning an array of 5 pointers to int

   J) a pointer to a pointer to a function that returns an int

   K) a function that takes an int and returns a pointer to an array of 5 ints

   L) a function taking an array of 5 pointers to (functions taking an int and returning an int) and returning an int

   M) a function taking a pointer to array of 5 ints and returning a pointer to an int

   N) illegal declaration

   O) a pointer to an array of 5 pointers to int

   P) a pointer to a pointer to an array of 5 int

   Q) a function that takes an int and returns a pointer to a pointer to an int

   2-1._____ `int (*(*foo)[5])(int);`

   2-2._____ `int **foo[5];`

   2-3._____ `int *(foo(int))[5];`

   2-4._____ `int foo(int (*[5])(int));`

   2-5._____ `int (*foo)(int(*)[5]);`

3. **Problem** (3 * 2 pts):

At some moment computer memory has 2 free blocks of size 2000 bytes each. The blocks are NOT adjacent. You are about to allocate a data structure that holds elements of size 16 bytes (this is the size of data). Answer the following questions, assuming sizeof(pointer)=4)

A) 166

B) 187

C) 3000

D) 125

E) 200

3-1._____ maximum number of elements you can store in an array

3-2._____ maximum number of elements you can store in a singly-linked list

3-3._____ maximum number of elements you can store in a doubly-linked list

4. **Problem** (10 * 1 pts):

For each of the following expressions determine whether it's legal.
For legal expression write down its value.
Assume non-cumulative execution, i.e. all modifications from previous lines ARE LOST.
Note that the value of assignment is value of the right-hand side, if the assignment is legal.

```
short array[] = {3, 6, 2, 4, 7, 8};
/* assume array = 1000, sizeof(int)=sizeof(pointer)=4, sizeof(short)=2 */
short *p1 = &array[1];
short *p3 = &array[3];
short *p5 = &array[5];
```

A) legal, but not listed B) 1 C) -4 D) 5 E) 6 F) 8 G) 10 H) 1000 I) 1002 J) 1003 K) 1004 L) 1006 M) illegal N) 2 O) 3 P) 4 Q) 1001 R) 7 S) 1008	4-1._____ `p1-p5` 4-2._____ `p1-p3+p5` 4-3._____ `p5-2-p1` 4-4._____ `p1 += p5-p3` 4-5._____ `p1 -= p5-p3` 4-6._____ `p5 + p1` 4-7._____ `p5 - p1--` 4-8._____ `*--p3 = 2` 4-9._____ `*(p3+2) = 6` 4-10._____ `p3+2 = &array[2]`

5. **Problem** (4 * 1 pts):

Given the following declaration,

```
short sha[]={1,2,3,4,5,6,7,8};
short* ps1 = sha+1;
short* ps3 = &sha[3];
int*   pi0 = (int*)sha;
int*   pi2 = (int*)(&sha[2]);
```

evaluate expressions. Assume sizeof(int)=4, sizeof(short)=2.

A) 1 B) 2 C) 3 D) -1 E) 6 F) -6 G) 16 H) 4 I) -2 J) -4 K) illegal L) 8	5-1._____ `sizeof(sha)` 5-2._____ `sizeof(*sha)` 5-3._____ `pi2 - pi0` 5-4._____ `pi2 - ps1`

6. **Problem** (7 * 1 pts):

Given the following declaration,

```
int f1(void) { return 16;  }
int f2(void) { return 32;  }
int f3(void) { return 64;  }

int (*pf[])(void) = {f1, f2, f3};
```

evaluate expressions.

A) address of f1 B) address of f3 C) illegal D) 16 E) 32 F) 64 G) address of array H) address of f2	6-1._____ `(*(pf+1))()` 6-2._____ `(*(pf+1))` 6-3._____ `(*pf+1)()` 6-4._____ `pf()` 6-5._____ `pf[1]()` 6-6._____ `(**(pf+1))()` 6-7._____ `f3`

7. **Problem** (8 * 1 pts):

Which of the following assignments are legal? Assume Foo is a well-defined struct. Unless declaration is provided with a question, assume

```
Foo f;
const Foo cf;
Foo* p_f;
const Foo* p_cf;
Foo* const cp_f;
```

A) illegal B) legal	7-1._____ `Foo* p_f = &f;` 7-2._____ `Foo* p_f = &cf;` 7-3._____ `Foo * const cp_f = &f;` 7-4._____ `Foo * const cp_f = &cf;` 7-5._____ `cp_f = &f;` 7-6._____ `cp_f = &cf;` 7-7._____ `Foo* p_f = cp_f;` 7-8._____ `cp_f = p_f;`

8. **Problem** (4 * 1 pts):

Given the following declarations, evaluate expressions - use decimal system and sizeof(int)=sizeof(pointer)=4.

```
int array[][4] = { {1,2,3,4}, {11,12,13,14}, {21,22,23,24}, {31,32,33,34} };
int **pp_int = (int**)array;
/* assume array is 1000 */
```

A) 1002 B) 1036 C) 1060 D) 1048 E) 1000 F) 1009 G) garbage H) 1012	8-1._____ `&array[3];` 8-2._____ `&array[3][3];` 8-3._____ `&pp_int[3];` 8-4._____ `pp_int[3][3]`

9. **Problem** (6 pts):

Write a function that accepts an integer and sets its bits 0,2,4 to 1 and bits 1,3 to 0. Points are awarded for correctness, meaningfullness, and compactness of your code.

10. **Problem** (12 pts):

Write a function (provide signature and implementation) that inserts a given node right BEFORE the last node of a singly linked list. If the list is empty – then the new node becomes the only node in the list.

Points are awarded for correctness and compactness of code.

```
struct Node {
  ... /*some data*/
  struct Node *next;
};
```

11. **Problem** (8 pts):

Write the C declaration for each of the English statements below. If a statement describes an illegal declaration, then write ILLEGAL.

a. foo is a function that takes an pointer to an int and returns a pointer to an int.

b. foo is a pointer to an array of 5 functions that take an int and returns an int

c. foo is an array of 10 pointers to arrays of 5 pointers to int

d. foo is a function that takes a pointer to (a function that takes an int and returns an int) and returns a pointer to function that (takes nothing and returns nothing)

12. **Problem** (6 pts):

Study the code below carefully. What does it print out? You should draw a diagram to help visualize the operations. (Hint: The outputted characters form a word.)

```
#include <stdio.h>
void main() {
    char *strs[] = {"In","Ithaca","socialism", "savors","relaxation",0};
    char** pps = strs;
    int i=0;

    while(i<4) {
        printf ("%c", *(*++pps + ++i) - i);
    }
}
```

13. **Problem** (10 pts):

Write a function that accepts an integer *dim* and allocates a 2-D array of doubles of size $dim \times dim$. Array elements should be initialized using formula $i/(j+1)$ for position (i, j).

Function should return the array to a client who is using the code below.

Implement a function that properly deletes array allocated by `create2Darray`.

```
int main() {
  int size = 5;
  ..... a = create2Darray( size );
  if (a) {
    printf("a[%i,%i]=%f",3,2,a[3][2]);
    clean_up( a, size );
  }
  return 0;
}
/* output of this program should be "1" */
```

Incomplete prototype - fill in the blanks and implement the function:

_____ `create2Darray ( int dim ) {`

()	Grouping	exp	N/A
()	Function call	rexp	L-R
[]	Subscript	lexp	L-R
.	Structure member	lexp	L-R
->	Structure pointer member	lexp	L-R
++	Postfix increment	rexp	L-R
--	Postfix decrement	rexp	L-R
!	Logical negate	rexp	R-L
~	One's complement	rexp	R-L
+	Unary plus	rexp	R-L
-	Unary minus	rexp	R-L
++	Prefix increment	rexp	R-L
--	Prefix decrement	rexp	R-L
*	Indirection (dereference)	lexp	R-L
&	Address of	rexp	R-L
sizeof	Size in bytes	rexp	R-L
(type)	Type conversion (cast)	rexp	R-L
*	Multiplication	rexp	L-R
/	Division	rexp	L-R
%	Integer remainder (modulo)	rexp	L-R
+	Addition	rexp	L-R
-	Subtraction	rexp	L-R
<<	Left shift	rexp	L-R
>>	Right shift	rexp	L-R
>	Greater than	rexp	L-R
>=	Greater than or equal	rexp	L-R
<	Less than	rexp	L-R
<=	Less than or equal	rexp	L-R
==	Equal to	rexp	L-R
!=	Not equal to	rexp	L-R
&	Bitwise AND	rexp	L-R
^	Bitwise exclusive OR	rexp	L-R
|	Bitwise inclusive OR	rexp	L-R
&&	Logical AND	rexp	L-R
||	Logical OR	rexp	L-R
?:	Conditional	rexp	N/A
=	Assignment	rexp	R-L
+=	Add to	rexp	R-L
-=	Subtract from	rexp	R-L
*=	Multiply by	rexp	R-L
/=	Divide by	rexp	R-L
%=	Modulo by	rexp	R-L
<<=	Shift left by	rexp	R-L
>>=	Shift right by	rexp	R-L
&=	AND with	rexp	R-L
^=	Exclusive OR with	rexp	R-L
|=	Inclusive OR with	rexp	R-L
,	Comma	rexp	L-R