1. C basics

(a) What are the final values of x, y and `result` in this code snippet? [3 points]

The expression y = y - 2 will NOT be evaluated due to short circuit!

```
int x = 1;
int y = 2;
int result = 0;
if ((x = x + 1) || (y = y - 2)) {
    result = 1;
}
```

x = 2 y = 2 result = 1

(b) What is the output of the following print statements? [4 points]

```
printf("%c\t", mystery('K'));
printf("%c\t", mystery(97));
printf("%c\t", mystery(86));
printf("%c\n", mystery('i'));
```

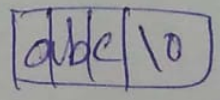
```
int mystery(int ch) {
    if (ch >= 'a' && ch <= 'z') {
        return ch - 'a' + 'A';
    } else {
        return ch;
    }
}
```

97 → 'a'
86 → 'V'
mystery → to_upper

97 - 97 + 65

OUTPUT: K A V I

(c) The code snippet below tries to create a duplicate of a given string. Can you identify the bug in this code and correct it? You may assume that the functions `strlen()`, `strcpy()` work as expected and the calling function (e.g., `main()`) frees the allocated memory. Also assume that the pointer `s` points to a valid string and is NOT NULL. [3 points]




```
char *strdup(char *s) {
    char *d = malloc(strlen(s));
    if (d == NULL)
        return NULL;
    strcpy(d,s);
    return d;
}
```

BUG: We allocate one less than the required chars for the duplicate string.

FIX: char *d = malloc(strlen(s) + 1);

Probably more safe to do.
malloc(strlen(s) * sizeof(char) + sizeof(char));

128 64 32 16 8 4 2 1

"There are only 10 types of people in the world:
those who understand binary, and those who don't."

2. Integer Representations [10 points - 1 point for each part]

Assume that the size of an **integer** in a machine is **1 byte (8 bits)**. Answer the following questions with respect to this machine.

(a) What is the maximum value of an unsigned integer (in decimal)?

$2^8 - 1 = 255$

(b) What is the maximum value of a signed integer (in decimal)?

$2^7 - 1 = +127$

(c) What is the minimum value of a signed integer (in decimal)?

$-2^7 = -128$

(d) If 8 (eight) bits are used to represent an address on this machine, how many unique addresses are there?

$2^8 = 256$

(e) What is the value of the bit pattern 1010 1010 when it is interpreted as:

1 0 1 0 1 0 1 0
128 64 32 16 8 4 2 1

i. a hexadecimal integer: AA (OR) 0xAA

ii. an unsigned decimal integer: $128 + 32 + 8 + 2 = 170$

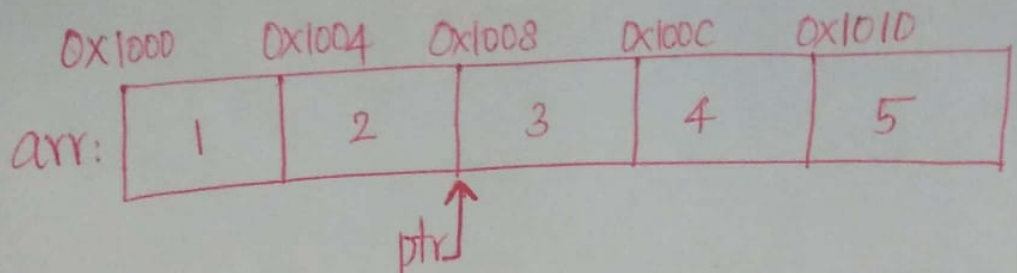
iii. a signed decimal integer: $-128 + 32 + 8 + 2 = -86$

(f) What is the binary representation of the following integer values (given in decimal) on this machine?

i. -1 (minus one) 1111 1111

ii. 42 0010 1010

iii. -128 1000 0000



3. Pointers, Dreaded Pointers! [10 points - 1 point for each part]

Answer the questions based on the following code snippet. You may assume that the **base location** (i.e., starting memory address) of the array `arr` is **0x1000**.

```
int arr[] = {1, 2, 3, 4, 5};
int *ptr = arr + 2;
```

(a) What are the values of the following expressions? In other words what is the value that will be printed if these expressions are used in a `printf()`. e.g., `printf("%d", *arr);` If the value of any expression is indeterminate, you should write "indeterminate". If any expression is illegal in C, you should write "illegal".

Expression	Value
`*arr`	1
`ptr[0]`	3
`ptr[2]`	5
`ptr[4]`	indeterminate (OR) illegal - OK
`arr[-1]`	indeterminate illegal - OK
`ptr[-1]`	2
`3[arr]`	4
`3[ptr]`	indet. OR illegal.

+2 for WHOLE CLASS. ENJOY! ☺

arr[3]
*((arr)+3)
↕
*(3+(arr))
3[arr]

(b) Are the following expressions legal or illegal in C? If an expression is legal, write "LEGAL"' and write the **value** of the expression (when it is used in a print statement like `printf("%p", ptr++);`). If an expression is illegal, write "ILLEGAL" and explain **why**.

i. `ptr++;` ✓ 0x1008

ii. `++arr;` ✗ arithmetic ops - invalid on array. can't change the base of an array.

4. Structures

(a) What is the size of `struct node` in the code snippet below? In other words, what is the output of the print statement? [3 points]

```
struct array {
        int block[1000];
};
struct node {
        struct node *prev;
        struct array data[1000];
        struct node *next;
};
printf("sizeof(struct node) = %d\n", sizeof(struct node));
```

4
1000 × 1000 × 4 = 4000000
4

OUTPUT: 4000008

(b) Is the following structure valid? If it's valid, write the **size** of this structure. If not, explain **why**. [2 points]

```
struct employee {
        char name[100];
        unsigned int id;
        struct employee manager;
};
```

ANSWER: X struct employee not a complete type.

(c) For the code snippet below, which of the given expressions are valid to print the real part of the first element (i.e., `c[0]`) in the array of complex numbers? [5 points]

```
struct complex {
        double real;
        double imag;
};
struct complex c[10];
struct complex *pc = c;
```

Valid (V) OR Invalid (I)	
i. `c[0].real`	V
ii. `c[0]->real`	I
iii. `pc[0].real`	V
iv. `*pc.real`	I
v. `pc->real`	V

(*pc).real ✓

5. Where am I allocated? - Code, Data, Stack, Heap?
[10 points - 2 points for each part]

Answer the following questions based on the code snippet below.

```
#include <stdlib.h>

#define N 100

int count = 0;

int addOne(int num)
{
        int next;
        next = num + 1;
        count++;
        return next;
}

int main()
{
        int num = 0;
        int *parray = malloc(sizeof(int) * N);
        int i;
        for (i = 0; i < N; ++i)
                parray[i] = addOne(num);
        return 0;
}
```

(a) In which part of the program memory (code, data, stack, and heap) are the following variables stored? If a variable is stored on the stack, write the name of the function in paranthesis (e.g., Stack (main)).

i. Integer variable `count`: data

ii. Integer variable `next`: stack (addOne)

iii. Pointer variable `parray`: stack (main)

iv. The 400 bytes of memory allocated using `malloc`: heap

v. Loop index variable `i`: stack (main)

6. Linked Lists

(a) How does the enigma function modify the linked list shown below? [8 points]

LINKED LIST: head -> 10 -> 20 -> 30 -> 40 -> 50 -> NULL

```
/* Link list node */
struct node
{
        int data;
        struct node* next;
};

void enigma(struct node** head_ref)
{
        struct node* prev   = NULL;
        struct node* current = *head_ref;
        struct node* next;
        while (current != NULL)
        {
                next  = current->next;
                current->next = prev;
                prev = current;
                current = next;
        }
        *head_ref = prev;
}
```

ANSWER: head → 50 → 40 → 30 → 20 → 10 → NULL

(b) How will you call the enigma() function from the main() function? [2 points]

```
int main() {

        /* Start with the empty list */
        struct node* head = NULL;

        /* assume the linked list is created here */

        /* TODO: call  enigma() using the list pointed to by head */
        enigma(&head);

        return 0;
}
```

"Low-level programming is good for the programmer's soul."

7. Bitwise Operations

Consider the following code snippet. Answer the following questions based on this code snippet. You should also assume that an **integer (signed and unsigned)** takes **1 byte** of memory on this machine.

```
int snum = -1;
unsigned int unum = 1;
```

snum: 1111 1111
unum: 0000 0001

(a) What are the values of the following expressions? In other words what is the value that will be printed if these expressions are used in a print statement. e.g., `printf("0x%x", snum & unum);` You may assume that the right shift on a **signed** integer will be an **arithmethic** right shift and that on an **unsigned** integer will be a **logical** right shift. Write all values in hexadecimal. `[6 points]`

Expression	Value
`snum & unum`	0x 01
`snum ^ unum`	0x FE
`snum && unum`	0x 01
`(snum << 7) | unum`	0x 81
`(snum << 7) >> 7`	0x FF
`(unum << 7) >> 7`	0x 01

1000 0000
0000 0001
1000 0001

1111 1111

(b) What is the output of the following print statement?
`printf("%d\t%d\t%d\t", nameMe(1), nameMe(0), nameMe(-1));`
Assuming that the valid inputs to this function will only be in the range `[-127, 127]`, can you give this function a suitable name based on what it is doing? `[3 + 1 = 4 points]`

```
int nameMe(int x) {
        int y = ~x + 1;
        return y;
}
```

→ 2's complement

OUTPUT: −1 0 +1

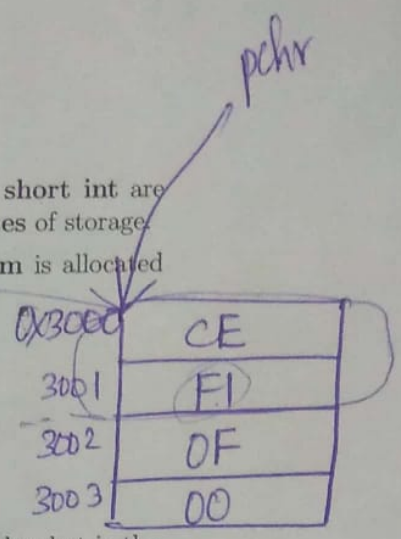
Suitable **name** for this function: negate

8. More low-level programming! How low are we going to go?!

Assume a **little-endian** machine in which the size of a **char** and **short int** are **1 byte** and **2 bytes** respectively. An **int** and a **pointer** take **4 bytes** of storage.

Consider the following code snippet. Assume that the variable **num** is allocated starting from memory address **0x3000**.

```
int num = 0x0FF1CE;
int *pnum = &num;
char *pchr = (char *)pnum;
```



(a) What are the values of the following expressions? In other words what is the value that will be printed if these expressions are used in a print statement. e.g., `printf("0x%x", *pnum);` Write all values in hexadecimal. [7 points]

Expression	Value
`*pnum`	0x OFF1CE
`(short)num`	0x F1CE
`pchr[1]`	0x FI
`(char)num`	0x CE
`++(*pchr)`	0x CF
`++pchr`	0x 3001
`++pnum`	0x 3004

(b) The function `isStrLonger()` compares the length of two strings `s` and `t` and should return:

i. 1 (one), if strlen(s) > strlen(t)

ii. 0 (zero), otherwise

The function signature of strlen is: `unsigned int strlen(char *s);` Will the function `isStrLonger()` work as expected? If yes, just write "WORKS". If not, mention the issue and provide a fix for the same. You may assume that the pointers `s` and `t` point to valid strings and are NOT NULL. [3 points]

```
int isStrLonger(char *s, char *t) {
        return strlen(s) > strlen(t);
}
```

ANSWER: WORKS.

9. Recursion and Preprocessor fun! :)

(a) What is the output of the following code snippet? [5 points]

```
#include <stdio.h>
int func(int n)
{
        if (n == 0)
                return 0;
        return (n % 10 + func(n / 10));
}
int main()
{
        int num = 12345;
        int result = func(num);
        printf("result = %d\n", result);
        return 0;
}
```

OUTPUT: 5+4+3+2+1 = 15

(b) What is the output of the following code snippet? Assume **malloc** is successful and the contents in the malloc'ed memory are all initialized to **0 (zero)**. [5 points]

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

#define ARRAY_LENGTH(A) (sizeof(A) / sizeof(A[0]))

int main(int argc, char *argv[]) {
        int *p = malloc(sizeof(int) * 5);
        for (i = 0; i < ARRAY_LENGTH(p); ++i) {
                printf("p[%d] = %d\n", i, p[i]);
        }
        int a[] = {1, 2, 3, 4, 5};
        int i;
        for (i = 0; i < ARRAY_LENGTH(a); ++i) {
                printf("a[%d] = %d\n", i, a[i]);
        }
        return 0;
}
```

OUTPUT: p[0] = 0

a[0] = 1

a[1] = 2

a[4] = 5

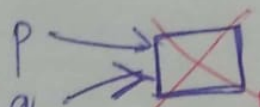
10. Bugs, Bugs, Bugs Everywhere There Are Bugs
[10 points - 2 points for each part]

The following code snippets have a few errors that are commonly encountered in C programs. If a code snippet has any issue(s), underline the issue and **explain the issue(s)** in a few words. Note that you need NOT provide a fix for the issue. If there are no issues in a code snippet, just write "OK".

Code Snippet	Issue

`int *func() {`
`  int result = 42;`
`  return &result;`
`}`
	returning pointer to a local variable is ~~invalid~~ incorrect because locals get destroyed when fn returns

`#define N (10)`
`int i = N, array[N];`
`for( ; i >= 0; i--)`
`  array[i] = i;`
	array[10]

`int *p = malloc(sizeof(int));`
`int *q = p;`
`p[0] = 3;`
`free(p);`
`q[0] = 42;`
	 accessing freed memory

`struct user {`
`  char name[100];`
`};`
 	
`// Assume malloc succeeds!`
`struct user *puser = malloc(sizeof(puser));`
 	
`// Assume a name with 50 chars`
`// is assigned to puser->name`
	struct user. ✗

`// Check if the value in memory`
`// pointed to by ptr is n.`
`bool isN(int* ptr, int n){`
`  return ptr && *ptr == n;`
`}`
	bool ✓
	OK ✓

"It is what it is. Accept it and move on."

11. Move Instructions

(a) Assume three variables x, y, and z that are in memory locations 0x2008, 0x2004, and 0x2000 respectively. The contents of the register %ebx is 0x200C. Given the following x86 assembly code, write the complete C code (including variable initializations) that was used to generate this assembly code. [8 points]

```
movl    $1, -4(%ebx)
movl    $2, -8(%ebx)
movl    -4(%ebx), %eax          /* line 3 */
movl    %eax, -12(%ebx)         /* line 4 */
movl    -8(%ebx), %eax
movl    %eax, -4(%ebx)
movl    -12(%ebx), %eax
movl    %eax, -8(%ebx)
```

```
int main(int argc, char *argv[]) {
        int x, y, z;

        // TODO: Write C code corresponding to the above assembly code.
        int x = 1;  int y = 2;  int z;
        z = x;
        x = y;
        y = x;
        return 0;
}
```

swap!

(b) In the assembly code shown above, can we replace lines 3 and 4 with the following line to achieve the same effect? If yes, write "YES"; If no, write "NO" and explain why. [2 points]

```
movl    -4(%ebx), -12(%ebx)
```

S D

X

BOTH cannot be memory locations.

"Memories warm you up from the inside. But they also tear you apart."

12. Memory and Control Structures in Assembly

Consider the following memory layout.

Address	Value
0x4000	0x3
0x4004	0x5
0x4008	0x7
0x400C	0x0
0x4010	0x0

The following are the initial values in the registers.

Address	Value
%eax	0x0
%ebx	0x4010
%ecx	0x400C
%edx	0x4000

Consider the following assembly code.

```
        jmp expr
top:
        movl    (%ebx), %eax
        movl    (%edx,%eax,4), %eax
        addl    %eax, (%ecx)
        addl    $1, (%ebx)
expr:
        cmpl    $2, (%ebx)
        jle top
```

What are the final values in the following memory locations after the above assembly instructions are executed?

Address	Value (in hexadecimal)	
0x400C	0xF	[3 points]
0x4010	0x3	[3 points]

Explain what this piece of assembly code does (e.g., It computes the maximum of two values at addresses 0x4008 and 0x400C and stores it at address 0x4010). You need NOT write any corresponding C code. [4 points]

Adds the values at 0x4000, 0x4004, and 0x4008 and stores it at 0x400C.

Optional: 0x4010 is used as a counter.