Family Name:. . . . . **Other Names:** . . . .

Student ID:. . . . . **Signature**. . . . .

NWEN 241: Test 1

2023, October 27 ** WITH SOLUTIONS **

Instructions

- Time allowed: **90 minutes**
- Attempt **all** the questions. There are **44 marks** in total.
- Write your answers in this exam paper and hand in all sheets.
- If you think a question is unclear, ask for clarification.
- You may use unmarked paper Chinese-English translation dictionaries.
- You may write notes and workings on this paper, but make sure your answers are clear.

Questions	Marks	
1. True or False	[15]	
2. Multiple Choice	[12]	
3. Short Answer	[17]	
	TOTAL:	

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 1. True of False **[15 marks]**

For the following statements, circle "true" or "false" for each statement.

(a) **[1 mark]** `123variable` is a valid C identifier.

true [✓ false]

(b) **[1 mark]** `6.022E23L` is a valid C identifier.

[✓ true] false

(c) **[1 mark]** The statement `int c = 'A'++;` is valid, resulting in the variable `c` having a value of `66` since the numeric value of the character `'A'` is `65`.

true [✓ false]

(d) **[1 mark]** The expression `5.5 + 'X' / 8` evaluates to a value that has type `float`.

[✓ true] false

(e) **[1 mark]** Arrays in C can have a dynamic size that changes during program execution. `float`.

true [✓ false]

(f) **[1 mark]** An array name in C is a pointer to the first element of the array.

[✓ true] false

(g) **[1 mark]** The following C code will compile successfully:

[✓ true] false

```
int foo(const int *a, const int *b)
{
    (*b)++;
    return *a + *b;
}
```

(h) **[1 mark]** When you pass an array to a function in C, it is always passed by value, making a copy of the entire array.

true [✓ false]

(i) **[1 mark]** The following C code will compile successfully:

true [✓ false]

```
#include <stdio.h>
int main(void)
{
    char *str = "nWEN241";
    str [0] = 'N';
}
```

(Question 1 continued on next page)

(Question 1 continued)

(j) **[1 mark]** In the following declaration:

```
register int count;
```

the value of variable `count` is **NOT** guaranteed to be stored in a CPU register.

[✓ true] false

(k) **[1 mark]** Declaring auto variables of the same name in two different non-overlapping blocks will cause compilation issues.

true [✓ false]

(l) **[1 mark]** In C, a string is an array of characters terminated by a null character ($'\backslash 0'$)

[✓ true] false

(m) **[1 mark]** The `strlen()` function in C returns the length of a string including the null character.

true [✓ false]

(n) **[1 mark]** The arrow operator (`->`) is used to access structure members through a pointer to a structure.

[✓ true] false

(o) **[1 mark]** A pointer to a function can be used to call that function.

[✓ true] false

Question 2. Multiple choice ☑ **[12 marks]**

Hint: There might be more than one correct answer for each question

(a) **[1 mark]** Which of the following are valid integer literals in C?

- ☐ [✓ 42]
- ☐ 3.14
- ☐ [✓ 0x1A]
- ☐ 1e5
- ☐ 'A'

(b) **[1 mark]** A C program contains the following declarations:

```
int i, j;
long ix ;
short s;
float x;
char c;
```

What is the resulting data type of the expression?

```
3.5 * i + (short) (ix / s) - x * c / j
```

- ☐ [✓ float]
- ☐ double
- ☐ int
- ☐ long
- ☐ char

(c) **[1 mark]** Consider the following function-like macro:

```
#define FUNCMACRO(X,Y) X/Y
```

What value does the macro evaluate when invoked as `FUNCMACRO(1+8, 4-3)`?

- ☐ [✓ 0]
- ☐ 9
- ☐ the string "1+8/4-3"
- ☐ None of the above

(d) **[1 mark]** Which of the following is a correct way to use a function-like macro?

- ☐ #define SQUARE(x) x * x
- ☐ [✓ int result = SQUARE(5);]
- ☐ [✓ int result = SQUARE(5 + 2);]
- ☐ #define SUM(a, b) a + b

(Question 2 continued on next page)

(Question 2 continued)

(e) **[1 mark]** Consider the following statement:

```
char str[] = "Seven";
```

What is the size of the array `str`?

- ☐ 5
- ☐ 6 ☑
- ☐ 7
- ☐ None of the above

(f) **[1 mark]** Consider the following C code snippet:

```
char str1[] = "String 1";
char *str2 = "String 2";
```

Select ALL valid statements from the following:

- ☐ `str1[0] = 's';` ☑
- ☐ `str2[0] = 's';`
- ☐ `strcpy(str1, str2);` ☑
- ☐ `strcpy(str2, str1);`
- ☐ `str2 = str1;` ☑

(g) **[1 mark]** Suppose the following declarations are given:

```
int i = 5, j = 10, *ip;
ip = &i;
```

Which of the following statements use * for indirection?

- ☐ `int *x = ip;`
- ☐ `i = i * j;`
- ☐ `j = j * *ip;` ☑
- ☐ `int **y = &ip;`

(h) **[1 mark]** Consider the following code snippet:

```
int a = 2, b = 3, *x, *y;
x = &a;
y = &b;
*x = *x + *y;
```

What is the resulting value of a?

- ☐ 2
- ☐ 3
- ☐ 5 ☑
- ☐ 8

(Question 2 continued on next page)

(Question 2 continued)

(i) **[1 mark]** Consider the following C snippet:

```
int a[ ] = {2, 4, 6, 8};
int *p = a;
```

Select ALL expressions that will return the value of the third element of the array a, that is, the value 6.

- ☐ a[2] [✓]
- ☐ *a+2
- ☐ *(p+2) [✓]
- ☐ p[2] [✓]
- ☐ p+2

(j) **[1 mark]** Consider the following code snippet:

```
int n[ ] = {1, 2, 3, 4, 5, 6, 7, 8};
int *p = n + *n;
```

What is the value of `*(n + *p)`?

- ☐ 2
- ☐ 3 [✓]
- ☐ 4
- ☐ 5

(k) **[1 mark]** Consider the following C code snippet:

```
enum loudness { moderate, defeaning = 2, painful };
```

What is the value of `painful`?

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ 3 [✓]

(l) **[1 mark]** Consider the following C code snippet:

```
union {
    char c;
    short s;
    int i;
    long l;
} u;

u.i = 4;
```

What is the size of the variable u equal to?

- ☐ sizeof(char)
- ☐ sizeof(short)
- ☐ sizeof(int) [✓]
- ☐ sizeof(long)

Question 3. Short Answer questions **[17 marks]**

(a) **[1 mark]** Consider the following C program:

```
#include <stdio.h>

int foo( int a, int b)
{
    return ++b / a;
}

int main(void)
{
    int i = 4;
    int j = 2 * foo(1+2, i+1);
    printf("%d %d", i, j);
    return 0;
}
```

What is the output of the program?

44

(b) **[2 marks]** Re-write `foo(int a, int b)` from the program in the previous question into a function-like `FOO(A, B)`. This will ensure that replacing the call to `foo(1+2, i+1)` with `FOO(1+2, i+1)` will result in the same output.

#define FOO(A,B) (((B)+1)/(A))

(Question 3 continued on next page)

(Question 3 continued)

(c) **[2 marks]** Consider the following declaration:

```
struct point {
    int x;
    int y;
};
```

Write a single statement declaring a variable `p1` of type `struct point` with the members `x` and `y` initialised to 10 and 20, respectively.

```
struct point p1 = {10, 20};
//or
//struct point p1 = { x: 10, y: 20}; // x and y can be in any order
//struct point p1 = { .x = 10, .y = 20}; // x and y can be in any order
```

(d) **[2 marks]** What will be the output of the following program?

```
#include <stdio.h>

void swap(int*, int*);

int main(void) {

    int a = 10;
    int b = 12;
    swap(&a, &b);
    printf("%d : %d\n", a, b);

}
void swap(int* a, int* b) {

    int temp = *a;
    *a = *b;
    *b = temp;

}
```

```
12 : 10
```

(Question 3 continued on next page)

(Question 3 continued)

(e) **[5 marks]** What will be the output of the following program?

Note: Suppose that a `short` occupies 2 bytes in memory. The array a is at memory address 100, while `ip` is at memory address 200 (all addresses are in decimal).

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

int main(void) {
    short a[ ] = {1, 2, 3, 4, 5, 6};
    short *ip = a;

    printf("1: %d\n", a);
    printf("2: %d\n", ip+1);
    printf("3: %d\n", &a[2]);
    printf("4: %d\n", *(ip+2));
    printf("5: %d\n", *++ip);

    return 0;
}
```

1: 100

2: 102

3: 104

4: 3

5: 2

(Question 3 continued on next page)

(Question 3 continued)

(f) **[5 marks]** Consider the following C program:

```
1 #include<stdio.h>
2
3 int a;
4
5 int func(int i)
6 {
7      int b;
8      static int c = 10;
9      b = c;
10     if(i == 0) c = c+b;
11     else if(i < 0) c--;
12     else c++;
13
14     return c;
15 }
16
17 int main(void)
18 {
19     int d = -1, e;
20     func(d);
21     d++;
22     func(d);
23     e = func(++d);
24     printf("%d", e);
25     return 0;
26 }
```

i. **[1 mark]** What is storage class of variable a?

extern

ii. **[1 mark]** In which memory segment is the variable b stored?

stack

iii. **[1 mark]** What is the lifetime of variable c?

static

iv. **[1 mark]** Until what line is variable e allocated space in memory?

Until line 26 or last line of the program.

v. **[1 mark]** What is the output of the program?

```
19
```

* * * * * * * * * * * * * * *

Student ID: . . . .

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

C Operator Precedence and Associativity

This page lists all C operators in order of their precedence (highest to lowest). Their associativity indicates in what order operators of equal precedence in an expression are applied.

Operator	Description	Associativity

`()`
`[]`
`.`
`->`
	Parentheses (grouping)
	Brackets (array subscript)
	Member selection via object name
	Member selection via pointer
		left-to-right

`++ --`
`+ -`
`! ~`
(*type*)
`*`
`&`
`sizeof`
	Unary preincrement/predecrement
	Unary plus/minus
	Unary logical negation/bitwise complement
	Unary cast (change *type*)
	Dereference
	Address
	Determine size in bytes
		right-to-left

`* / %`	Multiplication/division/modulus	left-to-right
`+ -`	Addition/subtraction	left-to-right
`<< >>`	Bitwise shift left, Bitwise shift right	left-to-right

`< <=`
`> >=`
	Relational less than/less than or equal to
	Relational greater than/greater than or equal to
		left-to-right

`== !=`	Relational is equal to/is not equal to	left-to-right
`&`	Bitwise AND	left-to-right
`^`	Bitwise exclusive OR	left-to-right
`|`	Bitwise inclusive OR	left-to-right
`&&`	Logical AND	left-to-right
`||`	Logical OR	left-to-right
`?:`	Ternary conditional	right-to-left

`=`
`+= -=`
`*= /=`
`%= &=`
`^= |=`
`<<= >>=`
	Assignment
	Addition/subtraction assignment
	Multiplication/division assignment
	Modulus/bitwise AND assignment
	Bitwise exclusive/inclusive OR assignment
	Bitwise shift left/right assignment
		right-to-left

`,`	Comma (separate expressions)	left-to-right