

Carleton University
SYSC 2006 C – Foundations of Imperative Programming – Winter 2018
Midterm Exam - March 1, 2018

Name: _____

Student Number: _____

INSTRUCTIONS:

- 1. Exam questions will not be explained, and no hints will be given. If you think something is unclear or ambiguous, make a reasonable assumption (one that does not contradict the question), write it at the start of your solution, and answer the question.**
2. Do not write anything on this page other than your name and student number.
3. Answer all questions on the question paper. You can use pen or pencil.
4. This exam is closed book. Calculators are permitted. You are not permitted to use the calculator/clock apps on your cell phone. The time will be written on the blackboard and updated periodically throughout the exam.
5. For all questions, assume that all required `#include` statements are in the source code. The crib sheet on the last page of this question paper summarizes some C functions that you may find useful.
6. An attendance sheet will be distributed after the exam begins. If you finish your exam early, do not leave until you have signed the attendance sheet.
7. If you need to go to the washroom during the exam, bring your question paper to the teaching console at the front of the classroom.
8. When you have finished your exam, give it to one of the proctors. Show your campus card to the proctor, who will verify that the name and student number on your card match what is written on this page. You must leave the exam room as soon as you have turned in your paper.
9. Do not detach any sheets from the question paper. All pages of this question paper must be turned in.

Mark: _____ / 32

Question 1 [4 marks]

<pre>void do_that(int *m, int n) { *m = *m + 1; n = n + 2; printf("*m: %d, n: %d\n", *m, n); } void do_this(int x, int *y) { x = x + 3; *y = *y + 4; do_that(&x, *y); }</pre>	<pre>int main(void) { int a = 2; int b = 7; do_that(&a, b); printf("a: %d, b: %d\n", a, b); a = 2; b = 7; do_this(a, &b); printf("a: %d, b: %d\n", a, b); return 0; }</pre>
--	---

Here is an incomplete transcript of the console output from this program. On the ruled lines, write the missing parts of the output produced by the `printf` calls. (0.5 marks will be awarded for each correct part.)

*m: _____, n: _____

a: _____, b: _____

*m: _____, n: _____

a: _____, b: _____

Multiple-choice Questions [28 marks]

For Questions 2 through 14, put a check mark, ✓, or an ✗, in the box, ☐, beside the correct answer. For each question, 2 marks will be awarded for selecting the correct answer and 0 marks will be awarded for selecting an incorrect answer. 0 marks will be awarded if you select more than one answer, even if one of the answers is correct.

Question 2

The C statement:

```
double &x;
```

- ☐ declares a variable named `&x` with type `double`.
- ☐ declares a variable named `x` with type "pointer-to-double".
- ☐ declares a variable named `x` that can store the address of a variable of type `double`.
- ☐ calculates the address of variable `x` and converts it to a value of type `double`.
- ☐ causes a compilation error.

Question 3

<pre>int f(int a) { return a * 4; }</pre>	<pre>int main(void) { int x = 6; double y; y = f(x); return 0; }</pre>
---	---

This program:

- ☐ causes a compilation error (mismatch between the declared return type of `f` and the declared type of `y`).
- ☐ terminates with a run-time error when it executes `y = f(x);`
- ☐ assigns function `f` to `y`.
- ☐ assigns the expression, `a * 4`, to `y`.
- ☐ assigns 24 to `y`.
- ☐ assigns 24.0 to `y`.

Question 4

What does this program display?

<pre>void mystery(int a) { a = 10; printf("%d ", a); }</pre>	<pre>int main(void) { int a = 20; printf("%d ", a); mystery(a); printf("%d ", a); return 0; }</pre>
--	---

<input type="checkbox"/> 20 10 10	<input type="checkbox"/> 10 20 10 10
<input type="checkbox"/> 20 10 20	<input type="checkbox"/> 10 20 10 20
<input type="checkbox"/> 20 20 10	<input type="checkbox"/> 10 20 20 10
<input type="checkbox"/> 20 20 20	<input type="checkbox"/> 10 20 20 20

Question 5

A student was asked to write a function that returns the sum $a + (a + 1) + (a + 2) + \dots + b$, where a and b are integers. Here is the student's code:

```
int sum_integers(int a, int b)
{
    int sum;
    for (sum = 1; b > a; b = b - 1) {
        sum = sum + b;
    }
    return sum;
}
```

The student is provided with this sput test suite and a main function that calls `test_sum_integers`:

```
void test_sum_integers(void)
{
    sput_fail_unless(sum_integers(1, 1) == 1, "sum_integers(1, 1)");
    printf("Expected: 1, actual: %d\n", sum_integers(1, 1));
    sput_fail_unless(sum_integers(1, 5) == 15, "sum_integers(1, 5)");
    printf("Expected: 15, actual: %d\n", sum_integers(1, 5));
    sput_fail_unless(sum_integers(2, 4) == 9, "sum_integers(2, 4)");
    printf("Expected: 9, actual: %d\n", sum_integers(2, 4));
}
```

Select the correct statement:

☐ No output is displayed by sput, because Pelles C reports errors when `sum_integers` is compiled.

☐ The output displayed by sput is:

```
[1:1] test_sum_integers:#1 "sum_integers(1, 1)" pass
Expected: 1, actual: 1
[1:2] test_sum_integers:#2 "sum_integers(1, 5)" pass
Expected: 15, actual: 15
[1:3] test_sum_integers:#3 "sum_integers(2, 4)" pass
Expected: 9, actual: 9
```

☐ The output displayed by sput is:

```
[1:1] test_sum_integers:#1 "sum_integers(1, 1)" FAIL
!      Type:      fail-unless
!      Condition: sum_integers(1, 1) == 1
Expected: 1, actual: 2
[1:2] test_sum_integers:#2 "sum_integers(1, 5)" FAIL
!      Type:      fail-unless
!      Condition: sum_integers(1, 5) == 15
Expected: 15, actual: 20
[1:3] test_sum_integers:#3 "sum_integers(2, 4)" FAIL
!      Type:      fail-unless
!      Condition: sum_integers(2, 4) == 9
Expected: 9, actual: 15
```

There are more choices on the next page.

- ☐ The output displayed by sput is:

```
[1:1] test_sum_integers:#1 "sum_integers(1, 1)" pass
Expected: 1, actual: 1
[1:2] test_sum_integers:#2 "sum_integers(1, 5)" pass
Expected: 15, actual: 15
[1:3] test_sum_integers:#3 "sum_integers(2, 4)" FAIL
!      Type:          fail-unless
!      Condition: sum_integers(2, 4) == 9
Expected: 9, actual: 8
```

- ☐ The output displayed by sput is:

```
[1:1] test_sum_integers:#1 "sum_integers(1, 1)" FAIL
!      Type:          fail-unless
!      Condition: sum_integers(1, 1) == 1
Expected: 1, actual: 0
[1:2] test_sum_integers:#2 "sum_integers(1, 5)" FAIL
!      Type:          fail-unless
!      Condition: sum_integers(1, 5) == 15
Expected: 15, actual: 0
[1:3] test_sum_integers:#3 "sum_integers(2, 4)" FAIL
!      Type:          fail-unless
!      Condition: sum_integers(2, 4) == 9
Expected: 9, actual: 0
```

Question 6

Given an array named `arr`, consider the expression: `arr[i]`. Unlike many languages, C doesn't cause a run-time error if `i` is not valid and `arr[i]` accesses memory "outside" the array, but we can use an `assert` statement to do this.

If `n` is the array's capacity and all elements in the array have been initialized, which `assert` statement will cause a program to terminate if `i` is not valid?

- ☐ `assert(i < 0 && i >= n);`
- ☐ `assert(i >= 0 && i < n);`
- ☐ `assert(i >= 0 && i <= n);`
- ☐ `assert(i == 0 && i == n);`
- ☐ `assert(i != 0 && i != n);`

Question 7

A function contains the declaration, `int numbers[10];` and the statement:

```
numbers[5] = 2;
```

Select the equivalent statement that uses the `*` operator.

- ☐ `*numbers + 5 = 2;`
- ☐ `*(numbers + 5) = 2;`
- ☐ `(*numbers + 5) = 2;`
- ☐ `(*numbers) + 5 = 2;`

Questions 8 and 9 use these declarations:

```
typedef struct {  
    int red;  
    int green;  
    int blue;  
} RGB_colour_t;  
  
RGB_colour_t colour;
```

Question 8

The statement, `colour = (RGB_colour_t) {100, 50, 170};`

- ☐ initializes an array named `colour` that has three integer elements.
- ☐ initializes a struct named `colour` that has three integer members.
- ☐ initializes a variable named `colour` with the product of `RGB_colour_t` and `{100, 50, 170}`.
- ☐ initializes a variable named `colour` with the value returned by function `RGB_colour_t` when it is called with arguments `100, 50` and `170`.
- ☐ causes a compilation error.

Question 9

Select the statement that is equivalent to `colour.red = 100;`

- ☐ `colour[0] = 100;`
- ☐ `colour[red] = 100;`
- ☐ `colour->red = 100;`
- ☐ `(&colour)->red = 100;`
- ☐ None of the above.

Question 10

How many arrays will we have when the following code is executed?

```
int first[10];  
int *second = first;  
int *third = &first[0];  
int *fourth = malloc(10 * sizeof(int));  
int *fifth = fourth;  
int *sixth = &fourth[0];
```

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6

Question 11

This program is executed using C Tutor:

<pre>void mystery(int a[], int n) { a[1] = a[n-1] - a[2]; // Line A printf("Leaving mystery\n"); }</pre>	<pre>int main(void) { int arr[] = {3, 5, 8, 4, 18, 6, 12}; mystery(arr, 7); return 0; }</pre>
--	---

Which diagram is displayed by C Tutor immediately after the assignment statement at Line A is executed, but before printf is called?

☐

Stack

Heap

main

array

0	1	2	3	4	5	6
int	int	int	int	int	int	int
3	5	8	4	18	6	12

arr

mystery

n

int

7

array

0	1	2	3	4	5	6
int	int	int	int	int	int	int
3	4	8	4	18	6	12

a

☐

Stack

Heap

main

array

0	1	2	3	4	5	6
int	int	int	int	int	int	int
3	4	8	4	18	6	12

arr

mystery

n

int

7

array

0	1	2	3	4	5	6
int	int	int	int	int	int	int
3	4	8	4	18	6	12

a

☐

Note: the number partly obscured by the arrowhead is 3

Stack

Heap

main

array

0	1	2	3	4	5	6
int	int	int	int	int	int	int
3	4	8	4	18	6	12

arr

mystery

pointer

a

n

int

7

☐

None of these solutions correspond to the diagram displayed by C Tutor.

Question 12

Consider this code fragment, which calls `mystery`:

```
int arr[] = {1, 2, 3, 4, 5};
mystery(arr, sizeof(arr)/sizeof(arr[0]));
printf("Finished!");

void mystery(int a[], int n)
{
    int temp;
    for (int i = 0; i < n / 2; i += 1) {
        temp = a[i];
        a[i] = a[n - (i + 1)];
        a[n - (i + 1)] = temp;
    }
}
```

After `mystery` returns, what does `arr` contain?

- ☐ {1, 2, 3, 4, 5}
- ☐ {2, 1, 4, 3, 5}
- ☐ {3, 2, 1, 4, 5}
- ☐ {2, 3, 4, 5, 1}
- ☐ {5, 1, 2, 3, 4}
- ☐ {5, 4, 3, 2, 1}

Question 13

Function `find_max` contains a logic error in one of the four lines indicated by comments:

```
/* This function returns the position of the maximum element in the
 * subsection of the array arr, starting at position "first" and
 * ending at position "last".
 */
int find_max(int arr[], int first, int last)
{
    int best_so_far = first;           // line 1

    for (int i = first + 1; i <= last; i += 1) {
        if (arr[i] > best_so_far) {    // line 2
            best_so_far = i;           // line 3
        }
    }
    return best_so_far;                // line 4
}
```

Which one of the four lines indicated by the comments contains the logic error?

- ☐ line 1
- ☐ line 2
- ☐ line 3
- ☐ line 4

Question 14

Students were asked to finish `main` by adding one or more statements before the `printf` call, so that `point2` is a copy of `point1`.

<pre>typedef struct { int x; int y; } point_t;</pre>	<pre>int main(void) { point_t point1, point2; point_t *src, *dest; point1.x = 100; point1.y = 200; // One or more missing statements _____ _____ printf("(%d, %d)\n", point2.x, point2.y); // Outputs (100, 200) return 0; }</pre>
--	---

Here are the solutions written by several students:

Jack's solution:	<code>point2 = point1;</code>
------------------	-------------------------------

Gwen's solution:	<code>src = &point1;</code> <code>point2 = *src;</code>
------------------	--

Owen's solution:	<code>dest = &point2;</code> <code>*dest = point1;</code>
------------------	--

Toshiko's solution:	<code>src = &point1;</code> <code>dest = &point2;</code> <code>dest = src;</code>
---------------------	---

Ianto's solution:	<code>dest = malloc(sizeof(point_t));</code> <code>*dest = point1;</code> <code>src = dest;</code> <code>point2 = *src;</code> <code>free(dest);</code>
-------------------	---

How many solutions are correct?

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

Question 15

This question is worth 2 marks. For each part, put a check mark, ✓, or an ✗, in the box, ☐, beside the correct answer. For each part, 0.5 marks will be awarded for selecting the correct answer and 0 marks will be awarded for selecting an incorrect answer.

Assume that `point_t` is the typedef'd name of a struct "type". A program contains this code:

```
point_t *p1 = malloc(sizeof(point_t));
point_t *p2 = malloc(sizeof(point_t));
```

(a) The statements: <code>free(p1);</code> <code>free(p2);</code> correctly deallocate the structs.	<input type="checkbox"/> True <input type="checkbox"/> False
(b) The statements: <code>free(p2);</code> <code>free(p1);</code> correctly deallocate the structs.	<input type="checkbox"/> True <input type="checkbox"/> False
(c) The statements: <code>p1 = NULL;</code> <code>p2 = NULL;</code> correctly deallocate the structs.	<input type="checkbox"/> True <input type="checkbox"/> False
(d) The statements: <code>free(&p1[0]);</code> <code>free(&p2[0]);</code> correctly deallocate the structs.	<input type="checkbox"/> True <input type="checkbox"/> False

Crib Sheet

`void *malloc(int size);`

Allocates *size* bytes from the heap and returns a pointer to the memory. The memory is not initialized. On error, the function returns NULL.

`void free(void *ptr);`

Frees the memory pointed to by *ptr*, which must have been returned by a previous call to `malloc`. Otherwise, or if `free(ptr)` has already been called, undefined behaviour occurs. If *ptr* is NULL, no operation is performed.