Indian Institute of Technology Kharagpur
Programming and Data Structures (CS10001)
Autumn 2017-18: Class Test 2

Time: 1 Hr. **Full Marks**: 20

Section:	**Roll:**	**Name:**

Q. 01	Q. 02	Q. 03	Q. 04	Q. 05	Q. 06	Q. 07	**Total**

Please write the answers within the boxes provided in questions 1 through 5. Fill up the marked blanks in questions 6 and 7. Any answer written elsewhere will not be evaluated.

---

1. What is the output of the following C Program? **[1 + 1 = 2]**

```
#include <stdio.h>

int main() {
    int a = 27;
    int *p = &a;

    printf("%d\n", (*p)++);

    printf("%d\n", a);

    return 0;
}
```

2. In the context of: **[0.5 * 4 = 2]**

```
int a[] = { 1, 3, 5, 7, 9 };
int *p = a + 1;
```

evaluate the expressions below when `&a[2]` = `0x00BDFA68` and `sizeof(int) = 4`.

(a) `(p)` (b) `(p + 1)`

(c) `(*(p + 2))` (d) `(*p + 2)`

3. What is the output of the following C Program? **[0.5 + 1 + 0.5 = 2]**

```
#include<stdio.h>

int *func(int x, int *y) {
    x = 5;
    *y = 7;
    return y;
}

int main() {
    int a = 2, b = 3, c = 0;

    c = *func(a, &b);
    printf("%d %d %d\n", a, b, c);
    return 0;
}
```

4. What is the output of the following C Program? **[2]**

```
#include<stdio.h>

int main() {
    char *s[] = { "football", "cricket", "tennis", "hockey", "kabbadi" };
    char **ptr[] = { s + 4, s + 3, s + 2, s + 1, s }, **p;

    p = ptr[1];
    printf("%s", *p + 1);
    return 0;
}
```

5. What is the output of the following C Program? **[0.5 * 4 = 2]**

```
#include<stdio.h>

struct Student {
    unsigned int roll;
    struct Name { char first[20], last[20]; } name;
    double height;
};

int main() {
    struct Student s1 = { 15, { "Ramesh", "Gaur" }, 1.62 };
    struct Student s2 = { 23, { "Ram", "Das" } };
    s2.height = 1.68;

    printf("%d %lf\n", s1.roll, s2.height);
    printf("%s %s\n", s2.name.first, s1.name.last);

    return 0;
}
```

6. Function `int *Search(int k, int *lt, int *rt)` takes an integer search key `k` and two pointers `lt` and `rt` containing the addresses of two elements of an array of distinct integers sorted in increasing order. For a proper call to `Search`, we need `lt` $\leq$ `rt`. Otherwise, the array is considered empty. If `k` matches any element in the array between elements pointed to by `lt` and `rt`, `Search` returns the pointer to the element in the array where `k` is found. Otherwise, it returns a null pointer. Function `Search` carries out searching using the Binary Search strategy.

   Function `main` reads a key `k` and searches for it in a local array `a` using `Search`. It prints the position (array index) of `k` in `a` if it is found.

   Fill up the missing codes to complete `Search` and `main` functions. **[1 * 5 = 5]**

```
#include <stdio.h>

int *Search(int k, int *lt, int *rt) {
    int *m = 0;

    if (                         ) {    // Check for non-empty array
        -------------------------

        m =                         ;   // Compute pointer to middle element
            -------------------------
        if (*m == k)                    // Do we have a match with key?
            return m;                   // Return the pointer to match
        else if (*m > k)                // Decide which sub-array to search

            return Search(                              ); // Search one half
                          ------------------------------
        else

            return Search(                              ); // Search other half
                          ------------------------------
    }
    return m; // Return that there is no match
}
int main() {
    int a[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31 };
    int k = 0, *p = a, *q = 0;
    const int n = sizeof(a) / sizeof(int) - 1;

    scanf("%d", &k);
    if (q = Search(k, p, p + n))
                                                        // Compute the index for print
        printf("%d is in Array at index %d\n", k,                );
                                                ---------------
    else
        printf("%d is not in Array\n", k);

    return 0;
}
```

7. Function `void Reverse(char *str, <param2>, <param3>)` takes a C string `str` and reverses it. The reversed string is returned through <`param2`>. It also computes the length of the string `str` and returns it through <`param3`>.

   Fill up the missing codes to complete `Reverse` and `main` functions.

   [0.5 * 2 + 1 * 4 = 5]

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

void Reverse(char *str,

                   rstr,                n) {
    ---------------      ---------------

    char *lt = 0, *rt = 0;
    *n = strlen(str);
    *rstr = (char *)malloc((*n + 1)*sizeof(char));
    lt = *rstr;                             // Pointer to first character

    rt =                              ;  // Pointer to last character
        ------------------------------

    strcpy(*rstr, str);
    while (lt <= rt) { // Swap characters between two ends
        char t = *lt;

                        = *rt;        // Change at left end
        ---------------

                        = t;          // Change at right end
        ---------------
    }
    return;
}

int main() {
    char s[] = "wonderful";  // Input string
    char *rs;                // For Output (reversed) string
    unsigned int n;          // Length of string

    Reverse(                    );
            --------------------
    printf("Reverse of %s = %s\n", s, rs);
    free(rs);
    return 0;
}
```

Solutions

Q. 1:

```
27
28
```

Q. 2

```
00BDFA64
00BDFA68
7
5
```

Q. 3

```
2 7 7
```

Q. 4

```
ockey
```

Q. 5

```
15 1.680000
Ram Gaur
```

Q. 6

```
#include <stdio.h>

int *Search(int k, int *lt, int *rt) {
    int *m = 0;

    if (lt <= rt) {                       // Check for a valid array
        ------
        m = lt + (rt - lt) / 2;            // Compute pointer to middle element
            ---------------
        if (*m == k)                     // Do we have a match with key?
            return m;                    // Return the pointer to match
        else if (*m > k)                 // Decide which sub-array to search
            // Search one half of the array

            return Search(k, lt, m - 1);
                          -----------
        else
            // Search the other half of the array

            return Search(k, m + 1, rt);
                          -----------
    }
    return m; // Return that there is no match
}
int main() {
    int a[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31 };
    int k = 0, *p = a, *q = 0;
    const int n = sizeof(a) / sizeof(int) - 1;

    scanf("%d", &k);
    if (q = Search(k, p, p + n))
                                                  // Compute the index for print
        printf("%d is in Array at index %d\n", k, q - p);
                                                  -----
    else
        printf("%d is not in Array\n", k);

    return 0;
}
```

Q. 7

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

void Reverse(const char *str, // <param1>
    char **rstr,              // <param2>
    int *n) {                 // <param3>
    char *lt = 0, *rt = 0;
    *n = strlen(str);
    *rstr = (char *)malloc((*n + 1)*sizeof(char));
    lt = *rstr;             // Pointer to first character

    rt = *rstr + *n - 1;    // Pointer to last character

    strcpy(*rstr, str);
    while (lt <= rt) {
        char t = *lt;
        *lt++ = *rt;        // Change at left end
        *rt-- = t;          // Change at right end
    }
    return;
}

int main() {
    char s[] = "wonderful";  // Input string
    char *rs;                // For Output (reversed) string
    unsigned int n;          // Length of string

    Reverse(s, &rs, &n);
    printf("Reverse of %s = %s\n", s, rs);
    free(rs);
    return 0;
}
```