The University of New South Wales

Faculty of Science – School of Physics

PHYS2020 – Computational Physics

Sample final exam 2004

Time allowed : 2 hours.

Total number of questions : 15.

All questions are of equal value.

Answer all questions.

Candidates may not bring their own calculators.

Candidates may not retain this exam paper.

All sections must be written in black or blue ink.

Question 1

Given the following `typedef` describing a sphere, of radius r, centered at coordinate (x, y, z),

```
typedef struct {
    double x, y, z, r;
} sphere;
```

write a function with prototype

```
int spheresIntersect(sphere *s1, sphere *s2);
```

That returns true if and only if the two spheres intersect. You should use `assert` to handle the case where an illegal sphere (i.e., one with a negative radius) is specified.

Question 2

Write a function that satisfies the following definition:

```
int isSorted(int *array, int nElements) {
  /*
     array is a pointer to an array of ints.
     nElements is the number of ints in the array.

     This function returns true if the array elements are sorted
     in ascending numerical order, and false otherwise.

     Example: suppose array contains {5, 9, 13}, and nElements
     is 3. Then this function returns true since 5, 9, and 13 are in
     ascending numerical order.
   */
}
```

Question 3

Write a function that satisfies the following definition:

```
void findPositives(double *array, int *nElements) {
  /*
     array is a pointer to an array of doubles.
     *nElements is the number of doubles in the array.

     This function removes all the non-positive elements from
     array, and returns, in *nElements, the number of elements remaining.

     *nElements must be greater than or equal to zero. assert() is
     used to enforce this.

     Example: suppose array contains {1.0, 2.0, -1.0, 3.0}, and *nElements
     is 4. Then, after calling findPositives, the first three elements of
     array will be {1.0, 2.0, 3.0} and *nElements will be 3.
  */
}
```

Question 4

Why was `int nElements` used in question 2 and `int *nElements` used in question 3?

Question 5

Concisely describe three techniques for reducing the execution time of a program.

Question 6

When running the following program:

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

int main(void) {
  printf("%f\n", asin(1.1));
  return 0;
}
```

the result displayed on the computer screen is:

```
nan
```

What does this mean?

Question 7

What thing(s) is/are wrong with the following code fragment? [Note: there may be one thing wrong].

```
int array[] = {1, 2, 3};
int i, sum = 0;

for (i = 0; i <= 3; i++) {
    sum += array[i];
}
```

Question 8

Suppose that you need to generate over 1 billion sequences of random numbers for a Monte Carlo simulation. What are two potential problems in using the time in microseconds as the seed of the random number generator, as per the following code fragment?

```
    struct timeval t;

    assert(0 == gettimeofday(&t, NULL));

    printf("The time is %ld.%06ld seconds since the Unix epoch\n",
   t.tv_sec, t.tv_usec);

    // Use the number of microseconds as the seed for the system
    // random number generator.

    srandom(t.tv_usec);
```

Question 9

Describe the various deficiencies of using the equation

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

to calculate the roots of the quadratic equation $ax^2 + bx + c = 0$.

How would you overcome these difficiencies?

Question 10

In the calculation of the variance of a distribution, what is the purpose of the second term in the following equation? From a mathematical point of view (i.e., without considering the limited precision of a computer calculation) what is the numerical value of this term?

$$\text{var} = \frac{1}{n-1}\left(\sum_{j=0}^{n-1}(x_j - \overline{x})^2 - \frac{1}{n}\left(\sum_{j=0}^{n-1}(x_j - \overline{x})\right)^2\right)$$

where

$$\overline{x} = \frac{1}{n}\sum_{j=0}^{n-1} x_j.$$

Question 11

With the aid of graphs, compare the Euler, midpoint approximation, and monte carlo techniques for numerically integrating a function.

Question 12

Give a concise description of what it occuring in the following program fragment:

```
    int *p;
    assert(NULL != (p = (int *)malloc(100000 * sizeof(int))));
```

Question 13

What is wrong with the following function that is supposed to return the average of two integers? [Note: there may be more than one thing wrong].

```
int average(int i, int j) {
   return (i + j)/2;
}
```

How would you fix the function?

Question 14

Assuming a `char` can range from -128 to $+127$, an `int` from -32768 to $+32767$, and an `unsigned int` from 0 to $+65535$, how many times will "`statement`" be executed in each of these four separate loops?

```
unsigned int i;
```

```
for (i = 0; i > -1; i--) {
    statement;
}
```

```
int i;
for (i = 0; i > -1; --i) {
    statement;
}
```

```
char i;
for (i = 0; i++; i < 1) {
    statement;
}
```

```
char i;
for (i = 0; i < 255; i++) {
    statement;
}
```

Question 15

Under what conditions can the following fragment lead to program failure?

```
char input[80];
scanf("%s", input);
```