# Week 2 solutions

## Exercises — Part I

### Exercise 1

Both statements are false.

### Exercise 2: Do exercises 3-1 and 4-1 in "Beginning C"

### BC exercise 3-1

```
/*
   02635 Mathematical Software Programming
   Beginning C, exercise 3-1
*/
#include <stdio.h>

int main(void) {

  int choice=0;
  float temperature=0.0;

  printf("Temperature conversion"
         "- please select one of the following options:\n");
  printf("  1. Convert from degrees Celcius to degrees Fahrenheit\n");
  printf("  2. Convert from degrees Fahrenheit to degrees Celcius\n");
  printf("\nEnter your choice [1 or 2]: ");
  scanf("%i",&choice);

  if (choice == 1) {
    printf("Please enter temperature in degrees Celcius: ");
    scanf("%f",&temperature);
    printf("Temperature in degrees Fahrenheit: %.1f F\n",
      temperature*1.8+32);
  }
  else if (choice == 2) {
    printf("Please enter temperature in degrees Fahrenheit: ");
    scanf("%f",&temperature);
    printf("Temperature in degrees Celcius: %.1f C\n",
      (temperature-32)/1.8);
  }
  else {
    printf("Invalid choice.\n");
    return -1;
  }
```

```
  return 0;
}
```

**BC exercise 4-1**

```
/*
   02635 Mathematical Software Programming
   Beginning C, exercise 4-1
*/
#include <stdio.h>

int main(void) {

  int size=0;
  printf("Multiplication table - please enter size: ");
  scanf("%d",&size);

  // print first line with integers
  printf("      | ");
  for (int j=1;j<=size;j++)
    printf("%5d ",j);
  printf("\n-------");
  for (int j=1;j<=size;j++)
    printf("------");
  printf("\n");

  // print table
  for (int i=1;i<=size;i++) {
    printf("%5d | ",i);
    for (int j=1;j<=size;j++) {
      printf("%5d ",i*j);
    }
    printf("\n");
  }

  return 0;
}
```

**Example output**

```
Multiplication table - please enter size: 8
      |     1     2     3     4     5     6     7     8
-------------------------------------------------------
    1 |     1     2     3     4     5     6     7     8
    2 |     2     4     6     8    10    12    14    16
```

```
     3 |     3    6    9   12   15   18   21   24
     4 |     4    8   12   16   20   24   28   32
     5 |     5   10   15   20   25   30   35   40
     6 |     6   12   18   24   30   36   42   48
     7 |     7   14   21   28   35   42   49   56
     8 |     8   16   24   32   40   48   56   64
```

**Exercise 3: Do exercises 2, 3, and 4 (p. 39) in "Writing Scientific Software"**

**WSS exercise 2**

```c
/*
   02635 Mathematical Software Programming
   Writing Scientific Software, exercise 2
*/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <fenv.h>
#ifndef RMODE
#define RMODE FE_TONEAREST
#endif

#pragma STDC FENV_ACCESS ON
int main(void) {
  double x = 1.0;

  if (fesetround(RMODE)) {
    fprintf(stderr, "Failed to set rounding mode\n");
    exit(EXIT_FAILURE);
  }
  for (int k=0;k<=16;k++) {
    printf("f(10^(%-3d)) = %.16le\n", -k, (1-cos(x))/(x*x));
    x *= 0.1;
  }
  return 0;
}
```

**Output**

```
f(10^(0  )) = 4.5969769413186023e-01
f(10^(-1 )) = 4.9958347219742893e-01
f(10^(-2 )) = 4.9999583334736619e-01
f(10^(-3 )) = 4.9999995832550309e-01
f(10^(-4 )) = 4.9999999696126418e-01
f(10^(-5 )) = 5.0000004137018506e-01
```

```
f(10^(-6 )) = 5.0004445029117006e-01
f(10^(-7 )) = 4.9960036108131994e-01
f(10^(-8 )) = 0.0000000000000000e+00
f(10^(-9 )) = 0.0000000000000000e+00
f(10^(-10)) = 0.0000000000000000e+00
f(10^(-11)) = 0.0000000000000000e+00
f(10^(-12)) = 0.0000000000000000e+00
f(10^(-13)) = 0.0000000000000000e+00
f(10^(-14)) = 0.0000000000000000e+00
f(10^(-15)) = 0.0000000000000000e+00
f(10^(-16)) = 0.0000000000000000e+00
```

The limit of $f(x) = \frac{1-\cos(x)}{x^2}$ as $x \to 0$ can be found using L'Hôpital's rule, i.e.,

$$\lim_{x\to 0} f(x) = \lim_{x\to 0} \frac{\sin(x)}{2x} = \lim_{x\to 0} \frac{\cos(x)}{2} = \frac{1}{2},$$

and hence the computed value of $f(x)$ for small $x$ is quite far off. The problem is catastrophic cancellation in the numerator of $f(x)$.

The cosine function can be represented by the following Taylor series

$$\cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}.$$

If $x$ is close to zero, the fourth-order approximation

$$\cos(x) \approx 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4$$

is reasonably accurate. Indeed, for $|x| \leq 10^{-4}$ the magnitude of the fourth-order term is smaller than $0.02\epsilon$ (assuming double precision floating-point arithmetic). With this approximation we have

$$f(x) = \frac{1-\cos(x)}{x^2} \approx \frac{1}{2} - \frac{1}{24}x^2$$

for $x$ sufficiently close to zero. For $x = 10^{-6}$ we have

$$\begin{aligned}
\mathrm{fl}(1 - \mathrm{fl}(\cos(x))) &\approx \left(1 - \left(1 - \frac{1}{2}x^2 + \frac{1}{24}x^4\right)(1+\delta_1)\right)(1+\delta_2) \\
&\approx \frac{10^{-12}}{2} - \frac{10^{-24}}{24} - \delta_1
\end{aligned}$$

where $|\delta_1| \leq u$ and $|\delta_2| \leq u$, and hence the relative error is approximately

$$e_{\mathrm{rel}} \approx \frac{|\frac{1}{24} \cdot 10^{-24} + \delta_1|}{|\frac{1}{2} \cdot 10^{-12}|} \lessapprox \frac{2 \cdot 10^{-16}}{10^{-12}} = 2 \cdot 10^{-4}.$$

The computed value for $x = 10^{-6}$ yields a numerator with the (approximate) absolute error

$$e_{\mathrm{abs}} \approx |0.500044 - 0.5 + \frac{1}{24} \cdot 10^{-12}| \cdot 10^{-12} \approx 4.4 \cdot 10^{-17}$$

which corresponds to approximately $0.4u$, and hence the relative error in the numerator is approximately

$$e_{\mathrm{rel}} = \frac{e_{\mathrm{abs}}}{|1-\cos(x)|} \approx \frac{4.4 \cdot 10^{-17}}{\frac{1}{2} \cdot 10^{-12}} \approx 8.8 \cdot 10^{-5}.$$

### WSS exercise 3

```c
/*
   02635 Mathematical Software Programming
   Writing Scientific Software, exercise 3
*/
#include <stdio.h>
#include <math.h>

int main(void) {

  double x,y,xt,yt,tmp;

  printf("Input x: ");
  scanf("%lf",&x);
  printf("Input y: ");
  scanf("%lf",&y);

  xt = fabs(x);
  yt = fabs(y);

  // ensure that yt >= xt by swapping xt and yt if xt > yt
  if (xt > yt) {
    tmp = yt;
    yt = xt;
    xt = tmp;
  }

  if ( (yt != 0.0) && (yt != INFINITY) ) {
    tmp = xt/yt;
    printf("sqrt(x^2 + y^2) = %le\n", fabs(yt)*sqrt(1.0+tmp*tmp));
  }
  else {
    printf("sqrt(x^2 + y^2) = %le\n", yt);
  }

  return 0;
}
```

### WSS exercise 4

The solutions of a quadratic equation $ax^2 + bx + c = 0$ are

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

If $b$ is positive and $b^2$ is **large** compared to $ac$, then $-b + \sqrt{b^2 - 4ac}$ is likely to result in catastrophic cancellation. If $x_1$ and $x_2$ denote the two solutions, it is easy to verify that

$$x_1 x2- = \frac{c}{a},$$

as stated in the exercise. Thus, if $b^2$ is large compared to $4ac$, we compute

$$x_1 = \frac{-b - \operatorname{sgn}(b)\sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{c}{a x_1}.$$

*Remark*: The determinant, $b^2 - 4ac$, may also be subject to cancellation, and hence it is generally a good idea to compute this in extended precision.

```c
/*
   02635 Mathematical Software Programming
   Writing Scientific Software, exercise 4
*/
#include <stdio.h>
#include <math.h>

int main(void) {

  double a,b,c,xm,xp,det;

  printf("Solve quadratic equation a*x^2 + b*x + c == 0\n\n");

  // prompt user to enter a,b,c
  printf("Input a: ");
  scanf("%lf",&a);
  printf("Input b: ");
  scanf("%lf",&b);
  printf("Input c: ");
  scanf("%lf",&c);

  det = b*b - 4*a*c;

  if (a == 0) {
    if (b != 0)
      printf("x = %.4le\n",-c/b);
    else
      printf("a and b are both zero.\n");
    return 0;
  }
  else { // a is nonzero
    if (det < 0) { // complex roots
      printf("Complex roots\n");
      printf("x1 = %.4le + i*%.4le, ",-b/(2*a),sqrt(-det)/(2*a));
      printf("x2 = %.4le - i*%.4le\n",-b/(2*a),sqrt(-det)/(2*a));
      return 0;
    }
```

```
    else if (b*b > 10*a*c && b > 0) { // special case 1
      xm = -b - sqrt(det);
      xm /= 2*a;
      xp = c/(a*xm);
    }
    else if (b*b > 10*a*c && b < 0) { // special case 2
      xp = -b + sqrt(det);
      xp /= 2*a;
      xm = c/(a*xp);
    }
    else { // default case
      xp = (-b + sqrt(det))/(2*a);
      xm = (-b - sqrt(det))/(2*a);
    }
    printf("Real roots\n");
    printf("x1 = %.4le, x2 = %.4le\n",xp,xm);
    return 0;
  }
}
```

**Exercise 4: Modify your code from exercise 2 in "Writing Scientific Software" so that it uses the "round toward zero" rounding mode.**

Changing the rounding mode does not mitigate catastrophic cancellation. However, it reveals that some results (e.g., at $x = 10^{-7}$) are very sensitive to rounding errors:

```
f(10^(-7 )) = 5.1070259132757200e-01
```

**Exercise 5: Table with floating-point operations**

Write a program that prints a table with values `x op y` where

- `x` and `y` are `-INFINITY`, `-1.0`, `-0.0`, `0.0`, `1.0`, `INFINITY`, or `NAN`
- `op` is one of the arithmetic operators `*`, `/`, `+`, or `-`, or one of the relational operators `==`, `!=`, `>`, or `<`

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

#define xstr(s) str(s)
#define str(s) #s
#ifndef op
#define op ==
#endif
```

```
int main(void) {
  int i,j;
  double arr[7] = {-INFINITY,-1.0,-0.0,0.0,1.0,INFINITY,NAN};

  printf("op: %2s | ", xstr(op));
  for (i=0;i<7;i++)
     printf("%6.1f ", arr[i]);
  putchar('\n');
  for (i=0;i<57;i++)
     putchar('-');
  putchar('\n');

  for (i=0;i<7;i++) {
     printf("%6.1f | ", arr[i]);
     for (j=0;j<7;j++) {
        printf("%6.0f ", (double) (arr[i] op arr[j]));
     }
     printf("\n");
  }
  return 0;
}
```

```
$ gcc -Wall -lm -D op='*' fpops.c -o fpops
$ ./fpops
op:  * |   -inf   -1.0   -0.0    0.0    1.0    inf    nan
---------------------------------------------------------
  -inf |    inf    inf    nan    nan   -inf   -inf    nan
  -1.0 |    inf      1      0     -0     -1   -inf    nan
  -0.0 |    nan      0      0     -0     -0    nan    nan
   0.0 |    nan     -0     -0      0      0    nan    nan
   1.0 |   -inf     -1     -0      0      1    inf    nan
   inf |   -inf   -inf    nan    nan    inf    inf    nan
   nan |    nan    nan    nan    nan    nan    nan    nan
```

# Exercises — Part II

## Exercise 8

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

#define RECTANGLE 1
#define TRAPEZOIDAL 2

int main(void) {
```

```
  double a, b, h, val = 0, x;
  int n, method;

  // Print welcome message
  printf("Computes an approximation of the definite integral\n\n"
         "   int_a^b exp(-x^2) dx\n\n"
         "using numerical integration.\n\n");

  // Prompt user to enter integration limits
  printf("Please enter the integration limit a: ");
  scanf("%lf", &a);
  printf("Please enter the integration limit b: ");
  scanf("%lf", &b);
  // Check that a < b
  if (a>=b) {
    printf("error: a must be less than b\n");
    return -1;
  }

  // Prompt user to enter number of subintervals
  printf("Please enter the number of subintervals: ");
  scanf("%d",&n);
  // Check that n is positive
  if (n <= 0) {
    printf("error: n must be positive\n");
    return -1;
  }

  // Prompt user to choose method
  printf("Please select integration rule (%i. rectangle rule,"
         " %i. trapezoidal rule): ", RECTANGLE, TRAPEZOIDAL);
  scanf("%d",&method);
  // Check user input
  if (!((method == RECTANGLE) || (method == TRAPEZOIDAL))) {
    printf("error: unknown method\n");
    return -1;
  }

  // Compute approximation to definite integral and print result
  h = (b-a)/n;
  if (method == RECTANGLE) {
    for (int i=0; i<n; i++) {
      x = a + (i+0.5)*h;
      val += h*exp(-x*x);
    }
  }
  else if (method == TRAPEZOIDAL) {
```

```
    val = 0.5*h*(exp(-a*a) + exp(-b*b));
    for (int i=1; i<n-1; i++) {
      x = a+i*h;
      val += h*exp(-x*x);
    }
  }
  printf("Approximate value of definite integral: %.8e\n", val);

  return 0;
}
```

**Exercise 9**

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

int main(void) {

  double a, b, h, val1, val2, x;
  int n;

  // Print welcome message
  printf("Computes an approximation of the definite integral\n\n"
         "   int_a^b exp(-x^2) dx\n\n"
         "using numerical integration.\n\n");

  // Prompt user to enter integration limits
  printf("Please enter the integration limit a: ");
  scanf("%lf", &a);
  printf("Please enter the integration limit b: ");
  scanf("%lf", &b);
  // Check that a < b
  if (a>=b) {
    printf("error: a must be less than b\n");
    return -1;
  }

  // Prompt user to enter number of subintervals
  printf("Please enter the number of subintervals: ");
  scanf("%d",&n);
  // Check that n is positive
  if (n <= 0) {
    printf("error: n must be positive\n");
    return -1;
  }

  // Compute results and print table
```

```
  printf("Parameters:\n\n  a = %.3e\n  b = %.3e\n  n = %i\n\n",a,b,n);
  printf("Results:\n\n");
  printf("%3s  %-14s  %-14s\n","n","Rectangle","Trapezoidal");
  printf("-----------------------------------\n");
  for (int i=1;i<=n;i++) {
    h = (b-a)/i;

    // Rectangle rule
    val1 = 0.0;
    for (int j=0; j<i; j++) {
      x = a + (j+0.5)*h;
      val1 += h*exp(-x*x);
    }

    // Trapezoidal rule
    val2 = 0.5*h*(exp(-a*a) + exp(-b*b));
    for (int j=1; j<i; j++) {
      x = a+j*h;
      val2 += h*exp(-x*x);
    }

    // Print row
    printf("%3i  %.8e  %.8e\n",i,val1,val2);
  }

  return 0;
}
```

**Optional exercise: Monto Carlo integration**

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

int main(void) {

  double a, b, val, u;
  int n;

  // Initialize random number generator
  srand(time(NULL));

  // Print welcome message
  printf("Computes an approximation of the definite integral\n\n"
         "   int_a^b exp(-x^2) dx\n\n"
         "using Monte Carlo integration.\n\n");
```

```c
  // Prompt user to enter integration limits
  printf("Please enter the integration limit a: ");
  scanf("%lf", &a);
  printf("Please enter the integration limit b: ");
  scanf("%lf", &b);
  // Check that a < b
  if (a>=b) {
    printf("error: a must be less than b\n");
    return -1;
  }

  // Prompt user to enter number of samples
  printf("Please enter the number of samples: ");
  scanf("%d",&n);
  // Check that n is positive
  if (n <= 0) {
    printf("error: n must be positive\n");
    return -1;
  }

  // Compute result
  val = 0.0;
  for (int i=1;i<=n;i++) {
    u = a + (b-a)*rand()/RAND_MAX;
    val = (1.0-1.0/i)*val + exp(-u*u)/i;
  }
  printf("Approximate value of definite integral: %.8e\n", val*(b-a));

  return 0;
}
```