Information Processing 2

introduction to C programming

end of semester projects (weeks 13–15)

You can earn up to 50 points towards your final grade from projects. Choose, complete, and present any combination of projects that you like. Ask the instructors or TAs if you need help.

Your project work **must** be graded **in-class** by an instructor. We do not accept project submissions electronically.

Each project can be completed as a single '`.c`' source file or (if it makes sense) as two or more source files with any header files that are needed for them to work properly together.

When grading your project we will ask you questions about how it works. You will earn some of the available points for demonstrating a working project and the rest for your ability to explain its operation.

Contents

1 Prime number generator [10 points]

Write a prime number generator using the *Sieve of Eratosthenes*.[1] Your program should require exactly one command-line argument, an integer (no smaller than 2) specifying the largest prime number to print. An example session might be as follows:

```
$ ./primes
usage: ./primes max-prime
$ ./primes 0
./primes: max-prime 0 is too small (must be at least 2)
$ ./primes 20
2 3 5 7 11 13 17 19
```

To find the prime numbers up to some maximum n, an array $p[i]$ with indices $2 \leq i \leq n$ is used. Each element $p[i]$ is *true* if i is prime and *false* if it is not. Initially every element in p is set to *true*. For each i between 2 and n, the element $p[i]$ is tested. If $p[i]$ is *false* then the number is not prime and can be ignored. If $p[i]$ is *true* then then i is prime and all multiples of it $p[n \times i]$ (for $n \geq 1$) in the array can be set to *false*, since they are multiples of a prime factor i. When the process completes, the elements $p[i]$ that are *true* identify prime numbers i. As pseudo-code:

```
1 let $P$ be an array of boolean values indexed by integers 2 to $N$
2 set all elements of $P$ to $true$
3 for $i$ in $2 \ldots N$ do
4     if $P[i]$ is $true$ then
5         for $j$ in $2i \ldots N$ do
6             set $P[j]$ to $false$
7 for $i$ in $2 \ldots N$ do
8     if $P[i]$ is $true$ then
9         print $i$
```

Note that two worthwhile optimisations are possible. First, when a new prime i is discovered (at line 4) all composite (non-prime) numbers with factors $< i$ have already been removed from p. The loop on line 5 can therefore begin removing multiples of i starting with i^2 (rather than $2i$). Second, since every composite number n must have at least one factor $f \leq \sqrt{n}$, it is sufficient that the main loop on line 3 step i from 2 to $\sqrt{N}$ to guarantee that every non-prime number is removed from P.

Test your program by generating and printing the prime numbers between 2 and 100 (inclusive).

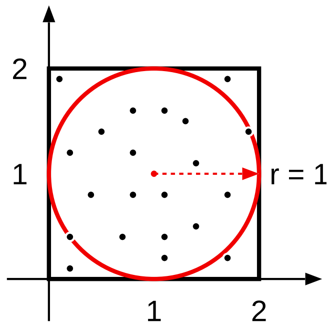
1.1 Deliverable

Present the entire program which we will test with various command-line arguments.

[1] `https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`

2 Monte Carlo simulation to calculate π [10 points]

Write a program to approximate π. You can do this by generating random points inside a 2×2 square and then counting what fraction of the points land inside a unit circle that is concentric with the square. The test for landing inside the circle is easy: the distance between the point and the centre of the square, calculated using Pythagoras' Theorem, must be < 1.



A graphical illustration of the simulation is shown on the right. Random points in the square bounded by $(0,0)$ and $(2,2)$ are generated. The number of points lying within the circle are tallied. The 2×2 square has an area of 4. The circle has an area of $\pi r^2 = \pi$ (since $r = 1$). The number of points lying inside the circle should therefore be $\frac{\pi}{4}$ of the total number of points. If the total number of points is (for example) $4,000,000$ then the number lying inside the circle should be approximately $1,000,000 \times \pi$.

The `<stdlib.h>` function `int rand(void)` returns random integers in the range $[0 \ldots \texttt{RAND\_MAX}]$. Convert these to random doubles in the range $[0 \ldots 1]$ and then multiply them by 2.0 to make x and y coordinates of random points within the square. For each point use Pythagoras' Theorem to calculate its distance d from $(1,1)$. If $d < 1$ then the point is inside the circle. Use the ratio of points inside the circle to the total number of points to calculate and print π.

Your program should accept one command-line argument, the total number of points to generate and test. Using `int` counters you can achieve no more than 9 significant digits of accuracy (why?) and so printing 8 digits after the decimal point is appropriate. An example session might be as follows:

```
$ ./pi
usage: ./pi n-points
$ ./pi 4000000
3142106 out of 4000000 points inside the circle
pi is approximately 3.14210600
```

2.1 Deliverable

Present the entire program which we will test with various command-line arguments.

3 Improved RPN calculator [5 points]

The slides for Week 8 show a reverse-Polish notation calculator. An example session, that calculates $1+2\times3+4$, might be as follows:

```
$ ./calc
1 2 3 * 4 + +
        11
```

Modify the calculator so that it uses `scanf()` to read numbers and operators. This should allow you to omit the space after numbers in your expressions, but if you find that you still need to read one character too much from the input then also use `ungetc()` to push the 'one character too much' back onto the standard input.

3.1 Deliverable

Present the entire program which we will test with various computations.

4 Add variables to a RPN calculator [10 points]

Prerequisite: an updated RPN calculator using `scanf` (and possibly `ungetc`) (see Project 3) will make a better starting point for this project than the version shown in the slides for Week 8.

Project 3 (and the Week 8 slides) show a reverse-Polish notation calculator. An example session, that calculates $1+2\times3+4$, might be as follows:

```
$ ./calc
1 2 3 * 4 + +
        11
```

Modify the calculator so that it supports 26 single-letter variables `a` through `z`. (Variable names should ignore case: `a` and `A` should both refer to the variable `a`. Hint: the <`ctype.h`> library function `int tolower(int c)` can help you achieve this.) A typical session might now look like this:

```
$ ./calc
a = 1 2 3 * 4 + +
        11
b = a a +
        22
20 b +
        42
```

Fewer than 20 additional lines of code are needed to implement variables, provided you make some simplifying assumptions:

- only one assignment is allowed per input line;
- no matter where the variable name and = symbol appear on a line, the assignment is always performed at the end of the line when a newline character is encountered; and
- `getop(char s[])` could handle a variable by printing its value as text into the array `s` used to communicate a `NUMBER` back to the main program, even though this might introduce some small rounding errors.

4.1 Deliverable

Present the entire program which we will test with various computations.

5 A 3×3 matrix data type [5 points]

Create a 3×3 matrix type that supports initialisation from constants, printing, and matrix multiplication.

5.1 Data type

Define a type Mat3 that holds a 3×3 matrix of double values. Make sure you can initialise your matrices using constants, like this.

```
Mat3 mat3 = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

5.2 Printing

Write a function void Mat3_print(Mat3 M) that prints the contents of the matrix M. At this point you should be able to run this program.

```
int main()
{
  Mat3 mat3 = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
  Mat3_print(mat3);
  return 0;
}
```

The output could look like an initialiser for a Mat3, for example:

```
{ { 1.000000, 2.000000, 3.000000 },
  { 4.000000, 5.000000, 6.000000 },
  { 7.000000, 8.000000, 9.000000 } }
```

5.3 Arithmetic

Write a function void Mat3_mulMat3(Mat3 A, Mat3 B, Mat3 P) that multiplies the matrix A by the matrix B, placing the result in matrix P. This program

```
  Mat3 A = { { 2, 1, 5 }, { 2, 10, 5 }, { 3, 1, 4 } };
  Mat3 B = { { 8, 7, 1 }, { 4,  2, 7 }, { 2, 3, 5 } };
  Mat3 P;
  Mat3_mulMat3(A, B, P);
  Mat3_print(P);
```

should print the following result:

```
{ { 30.000000 31.000000 34.000000 }
  { 66.000000 49.000000 97.000000 }
  { 36.000000 35.000000 30.000000 } }
```

5.4 Deliverable

Present the program (using the declarations, computations, and printing described above) which we will verify.

6 3-element vector data type [10 points]

Prerequisite: a working `Mat3` data type (see Project 5).

Create a 3-element vector data type that supports initialisation from constants, printing, and multiplication by a 3×3 matrix. The function `void Mat3_mulVec3(Mat3 M, Vect3 V, Vec3 P)` multiplies the matrix `M` by the vector `V` and stores the result in the vector `P`. You can check your implementation with the following program. (The expected output is shown after each print operation.)

```
    Vec3 V = { 3, 4, 1 };
    Vec3_print(V);

{ 3.000000, 4.000000, 1.000000 }

    Mat3 S = { { 3, 0, 0 }, {  0, 2, 0 }, { 0, 0, 1 } };
    Vec3 SV;
    Mat3_mulVec3(S, V, SV);
    Vec3_print(SV);

{ 9.000000, 8.000000, 1.000000 }

    Mat3 T = { { 1, 0, 5 }, {  0, 1, 7 }, { 0, 0, 1 } };
    Vec3 TV;
    Mat3_mulVec3(T, V, TV);
    Vec3_print(TV);

{ 8.000000, 11.000000, 1.000000 }

    Mat3 TS;
    Mat3_mulMat3(T, S, TS);

    Vec3 TSV;
    Mat3_mulVec3(TS, V, TSV);
    Vec3_print(TSV);

{ 14.000000, 15.000000, 1.000000 }
```

6.1 Deliverable

Present your completed program (using the declarations, computations, and printing described above) which we will verify.

6.2 Just for fun

If you consider the vector `V` to represent the (2-dimensional) point $(3,4)$ then, based on the results of the multiplications, can you guess what the matrices `S` and `T` might represent? If you can then test your theory by modifying the vector and matrices to see if the results are what you expect.

Can you now guess what the matrix product $T \times S = TS$ represents when applied to a vector `V`? If you have a theory about this then test it by modifying `S` and `T` and comparing the product $TS \times V$ to your predicted result.