FEEG6002 Advanced Computational Methods 1:

Laboratory-Assignment 6

Contents

- 1 Training 1: Determine pi using trapezoidal integration
- 2 Training 2: Allocating an array of longs dynamically
- 3 Laboratory: Creating an array of Fibonacci numbers

Prerequisites: pointers, dynamic memory allocation

1 Training 1: Determine pi using trapezoidal integration

The function $f(x)=\sqrt{1-x^2}$ for $x \in [-1,1]$ describes a half-circle. If we integrate it from -1 to 1, we obtain $\pi/2$, and thus $\pi = 2\int_{-1}^{1} f(x)dx$.

We carry out the integration numerically (with a pretty basic integration scheme) and will thus obtain an approximation pi(n) for π which depends on n. We use the composite trapezoidal rule which for a given n and $f(x)=\sqrt{1-x^2}$ can be described through pseudo code as:

```
a = -1
b = 1
h = (b-a)/n
s = 0.5 * f(a) + 0.5 * f(b)
for i from 1 to n-1
   x = a + i*h
   s = s + f(x)
end-of-for-loop
pi = s * h * 2
```

pi.c

```
#include<stdio.h>
/* TIMING CODE BEGIN (We need the following lines to take the timings.) */
#include<stdlib.h>
#include<math.h>
#include <time.h>
clock_t startm, stopm;
#define RUNS 1
#define START if ( (startm = clock()) == -1) {printf("Error calling clock");exit(1);}
#define STOP if ( (stopm = clock()) == -1) {printf("Error calling clock");exit(1);}
#define PRINTTIME printf( "%8.5f seconds used .", (((double) stopm-
startm)/CLOCKS_PER_SEC/RUNS));
```

```
/* TIMING CODE END */

int main(void) {
   /* Declarations */



   /* Code */
   START;           /* Timing measurement starts here */
   /* Code to be written by student, calling functions from here is fine
     if desired
   */



   STOP;            /* Timing measurement stops here */
   PRINTTIME;         /* Print timing results */
   return 0;
}
```

- Write a C program given above with name pi.c to compute the approximation of pi for a given n (use a symbolic constant for n). If you expand the template provided in pi.c, then this template will measure and print the execution time for you. In more detail:
- Write a function double f(double x) that $f(x) = \sqrt{1-x^2}$ returns for a given *x*.
- Write a function double pi(long n) which computes the approximation pi(n) of π as described in the pseudo code above, and returns this approximation as a double.
- Compile your code with -ansi -pedantic –Wall. If you use n=5, you should find an approximation for pi of 2.84767343
    - For n=10,000,000:
    - If you compare the numerical approximation of pi with the correct answer, how many digits after the decimal point are correct?
    - How long does the execution take?

<u>2 Training 2: Allocating an array of longs dynamically</u>

Create a file training6.c in which you

- Define a function long* make_long_array(long n) which takes a long integer n, dynamically allocates an array of n longs, and returns the pointer to the first element of this array.
- If the memory cannot be allocated, your function make_long_array should print "Memory allocation failed" and return the special pointer NULL.
- Here is some code you can use to test the function make_long_array:

```
void use_make_long_array(long n) {
  int i;
  long *p = make_long_array(n);

  printf("In use_make_long_array(%ld)\n", n);
```

```
/* if p is not NULL, we could allocate memory, and we proceed
   with testing: */
 if (p != NULL) {

/* write some data to the array -- if the allocated memory
     is too short, this might trigger a segfault */
    for (i=0; i<n; i++) {
     p[i]=i+42;          /* just write some data */
    }

  /* free array -- if the allocated array is too short, we may
     have corrupted malloc/free metadata when writing the i+42 data
     above, and this may show when we call the free command: */

  free(p);

  /* if the program does not crash, it is a good sign [but
     no proof for correctness]. The other way round: if the
     program segfaults or crashes, this is not a good sign. */
 }
 else { /* we get here if memory allocation didn't work for
         some reason. */
  printf("Error - it seems that the memory allocation failed.\n");
 }
}

int main(void) {
 int n;
 for (n=0; n<20; n++) {
  use_make_long_array(n);
 }
 return 0;
}
```

Email your file training6.c attached to an email with subject line training 6 to feeg6002@soton.ac.uk.

3 Laboratory: Creating an array of Fibonacci numbers

- Save your file training6.c under the new name lab6.c and add the following function: long* make_fib_array(long n) which takes an long integer and returns an array of long with n elements for which it uses dynamic memory allocation. (You may want to re-use and call the function make_long_array here.)
- The array of long integers should be populated with the Fibonnacci numbers.

  You can use this algorithm (shown as a Python function) to compute the array entries:

```
def fibs(n):
   """Given an integer number n, return a list with
```

```
    the first n fibbonnaci numbers. Assume that n>=2"""

    # create list fibs with n elements
    fibs = [0] * n

    # populate with data
    fibs[0] = 0
    fibs[1] = 1
    for i in range(2, n):
        fibs[i] = fibs[i - 1] + fibs[i - 2]
    return fibs
```

- If the function make_fib_array cannot allocate the memory for the fibonacci array, it should return the NULL pointer instead of a pointer to the array.
- You can use this C code and main function for some testing of your function make_fib_array:

```
void use_fib_array(long N) {
 /* N is the maximum number for fibarray length */
  long n;     /* counter for fibarray length */
  long i;     /* counter for printing all elements of fibarray */
 long *fibarray;  /* pointer to long -- pointer to the fibarray itself*/

  /* Print one line for each fibarray length n*/
  for (n=2; n<=N; n++) {
    /* Obtain an array of longs with data */
    fibarray = make_fib_array(n);

    /* Print all elements in array */
   printf("fib(%2ld) : [",n);
    for (i=0; i<n; i++) {
      printf(" %ld", fibarray[i]);
    }
    printf(" ]\n");

    /* free array memory */
    free(fibarray);
  }
}

int main(void) {
  use_fib_array(10);
  return 0;
}
```

Email your file lab6.c attached to an email with subject line lab 6 to feeg6002@soton.ac.uk.