6CS005 High Performance Computing

Multi-threading Problems

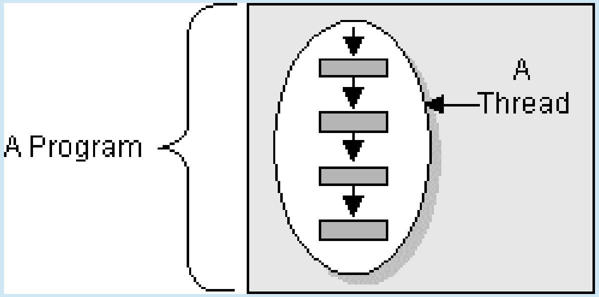
Dr Kevan Buckley

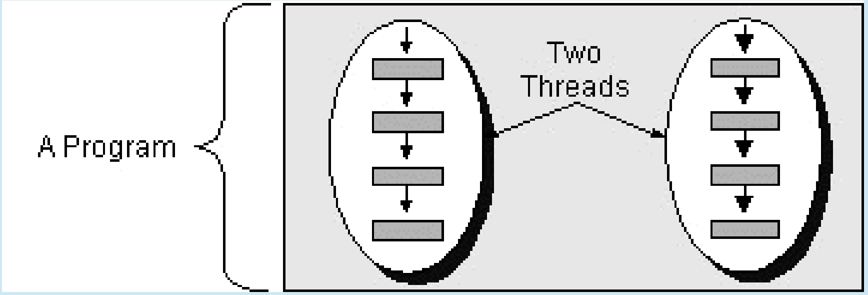
Aims

- To be able to write concurrent programs using POSIX threads.
- To understand problems that can occur in multi-thread programs.
  - Data corruption
  - Bottlenecks
- The code presented here can be found at http://www.scit.wlv.ac.uk/~in6659/hpc and on Canvas.

What is a Concurrent Program?

- A sequential program has a single thread of control.



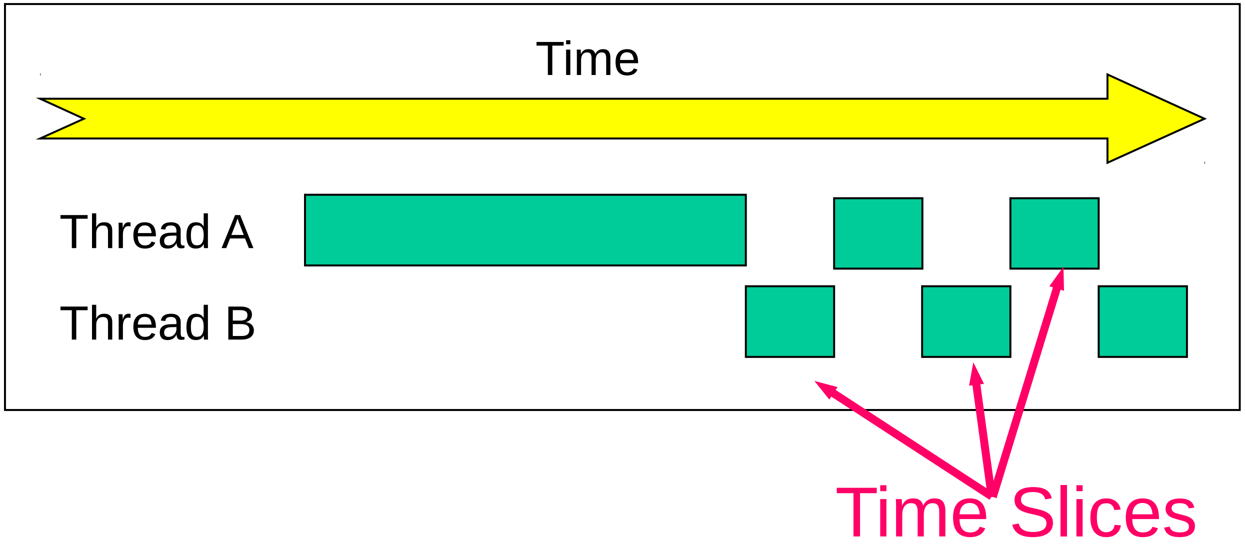


- A concurrent program has multiple threads of control allowing it to perform multiple computations in parallel and to control multiple external activities that occur at the same time.

Concurrency and Threads

- Modern operating systems run tasks concurrently by splitting tasks into smaller chunks.
- One task is executed for a small amount of time
- Then a thread is *pre-empted*, enabling another thread to run.

Execution is interleaved



If time slices are small enough it seems like several things are happening at once

On machines with more than one processor, threads might actually run simultaneously

Bank Account Program

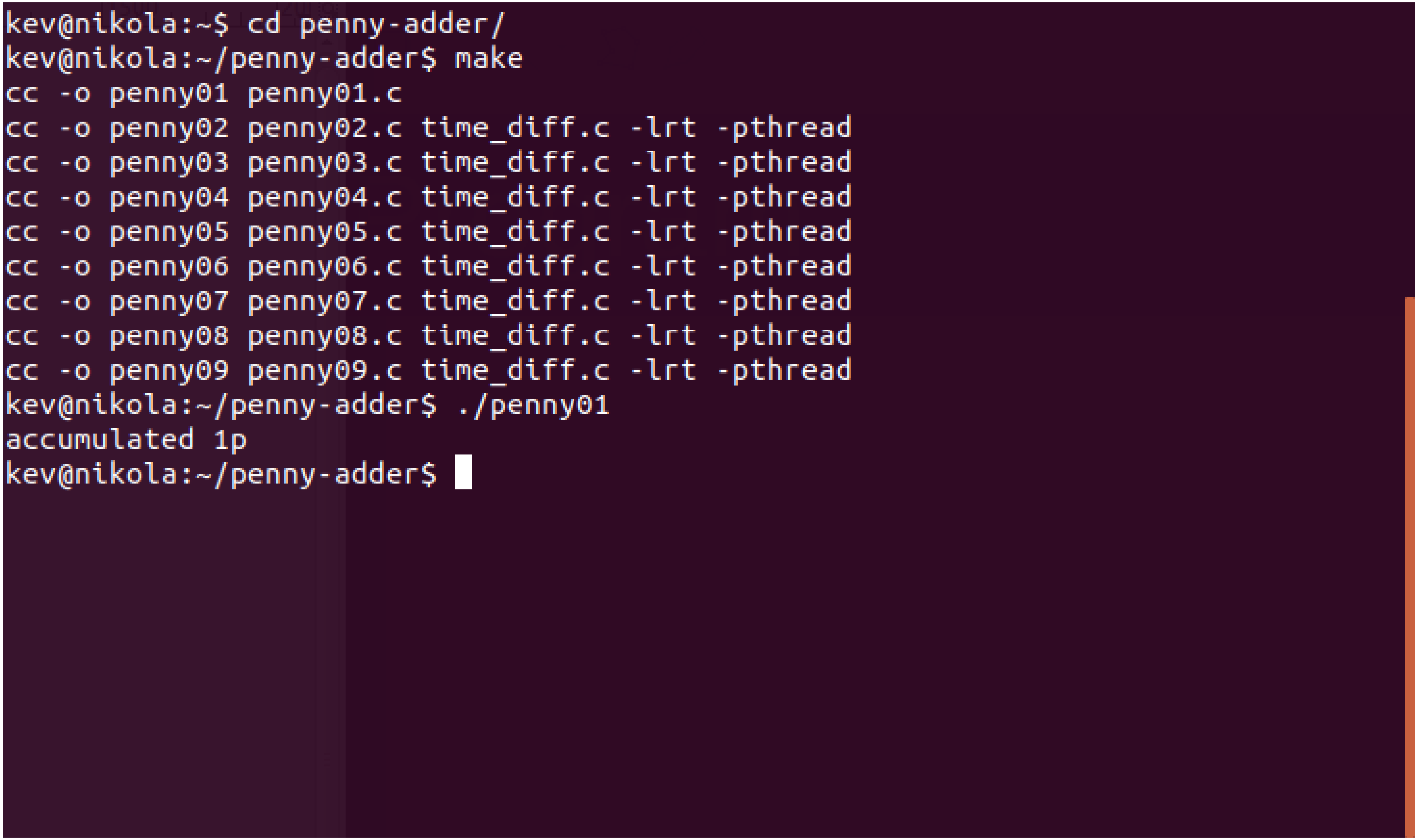
```
/****************************************************************************
  This program is the first in a series that looks at adding a penny to a
  bank account balance. It simply sets up a bank account balance as an
  integer and calls a function to add a penny to the account. This is done
  in a convoluted way (instead of just b++). This is for consistency with
  later versions that will demonstrate intermittent problems in the increment
  operation that can be accentuated when a delay is introduced.

  Compile with:

    cc -o penny01 penny01.c

  Dr. Kevan Buckley, University of Wolverhampton, 2018

*****************************************************************************/
```

Bank Account Program

```c
#include <stdio.h>
#include <unistd.h>

void add_penny(int *balance) {
  int b = *balance;
  usleep(1000000);
  b = b + 1;
  *balance = b;
}

int main(){
  int account = 0;
  add_penny(&account);
  printf("accumulated %dp\n", account);
  return 0;
}
```

How much money will the account contain?

Running the Program



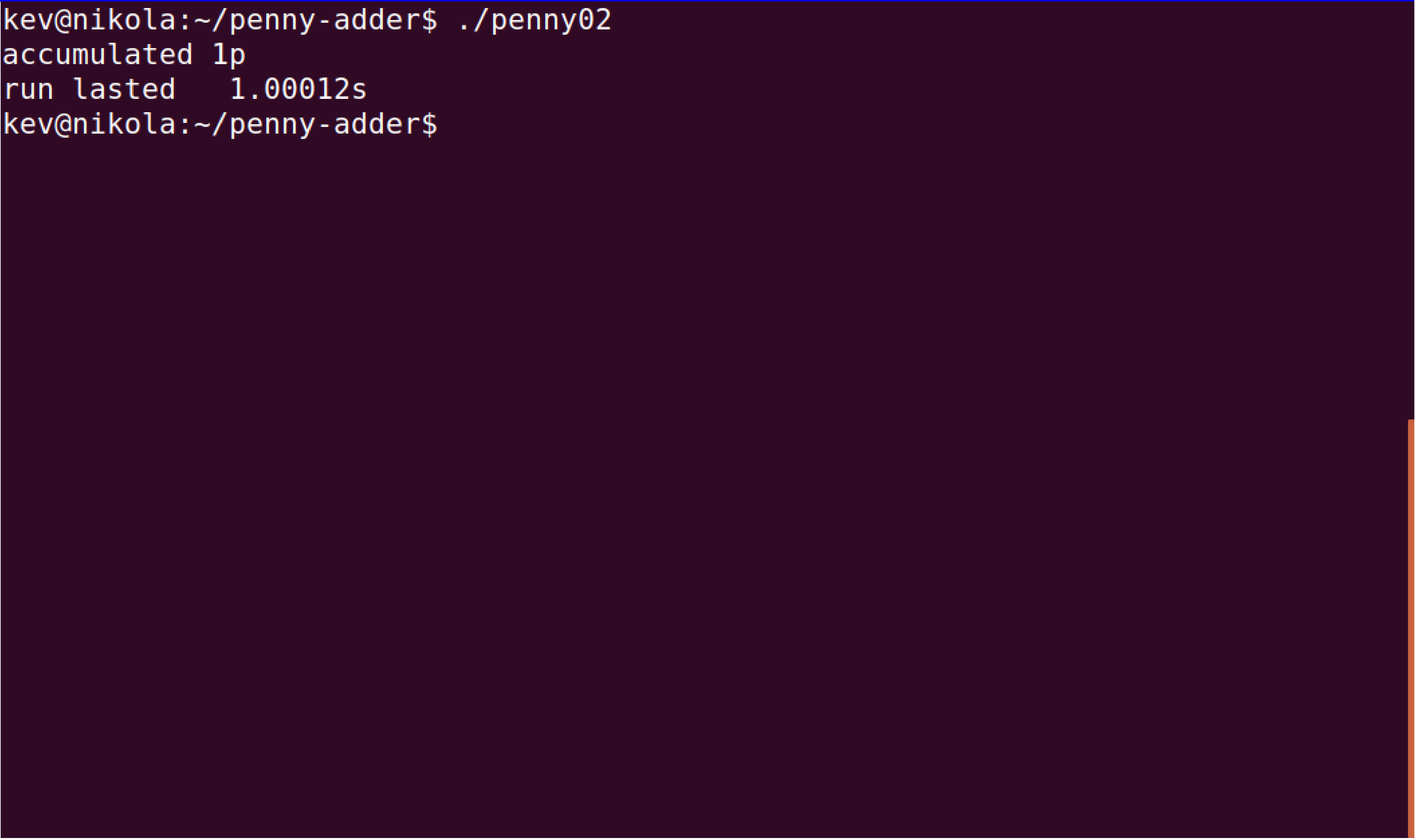
```
kev@nikola:~$ cd penny-adder/
kev@nikola:~/penny-adder$ make
cc -o penny01 penny01.c
cc -o penny02 penny02.c time_diff.c -lrt -pthread
cc -o penny03 penny03.c time_diff.c -lrt -pthread
cc -o penny04 penny04.c time_diff.c -lrt -pthread
cc -o penny05 penny05.c time_diff.c -lrt -pthread
cc -o penny06 penny06.c time_diff.c -lrt -pthread
cc -o penny07 penny07.c time_diff.c -lrt -pthread
cc -o penny08 penny08.c time_diff.c -lrt -pthread
cc -o penny09 penny09.c time_diff.c -lrt -pthread
kev@nikola:~/penny-adder$ ./penny01
accumulated 1p
kev@nikola:~/penny-adder$ 
```

Calculating The Running Time

```c
void add_penny(int *balance) {
  int b = *balance;
  usleep(1000000);
  b = b + 1;
  *balance = b;
}

int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  clock_gettime(CLOCK_MONOTONIC, &start);

  add_penny(&account);

  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %9.5lfs\n", difference/1000000000.0);
  return 0;
}
```

How long will it take?
How much money will be accoumulated?

Running the Program



```
kev@nikola:~/penny-adder$ ./penny02
accumulated 1p
run lasted   1.00012s
kev@nikola:~/penny-adder$
```

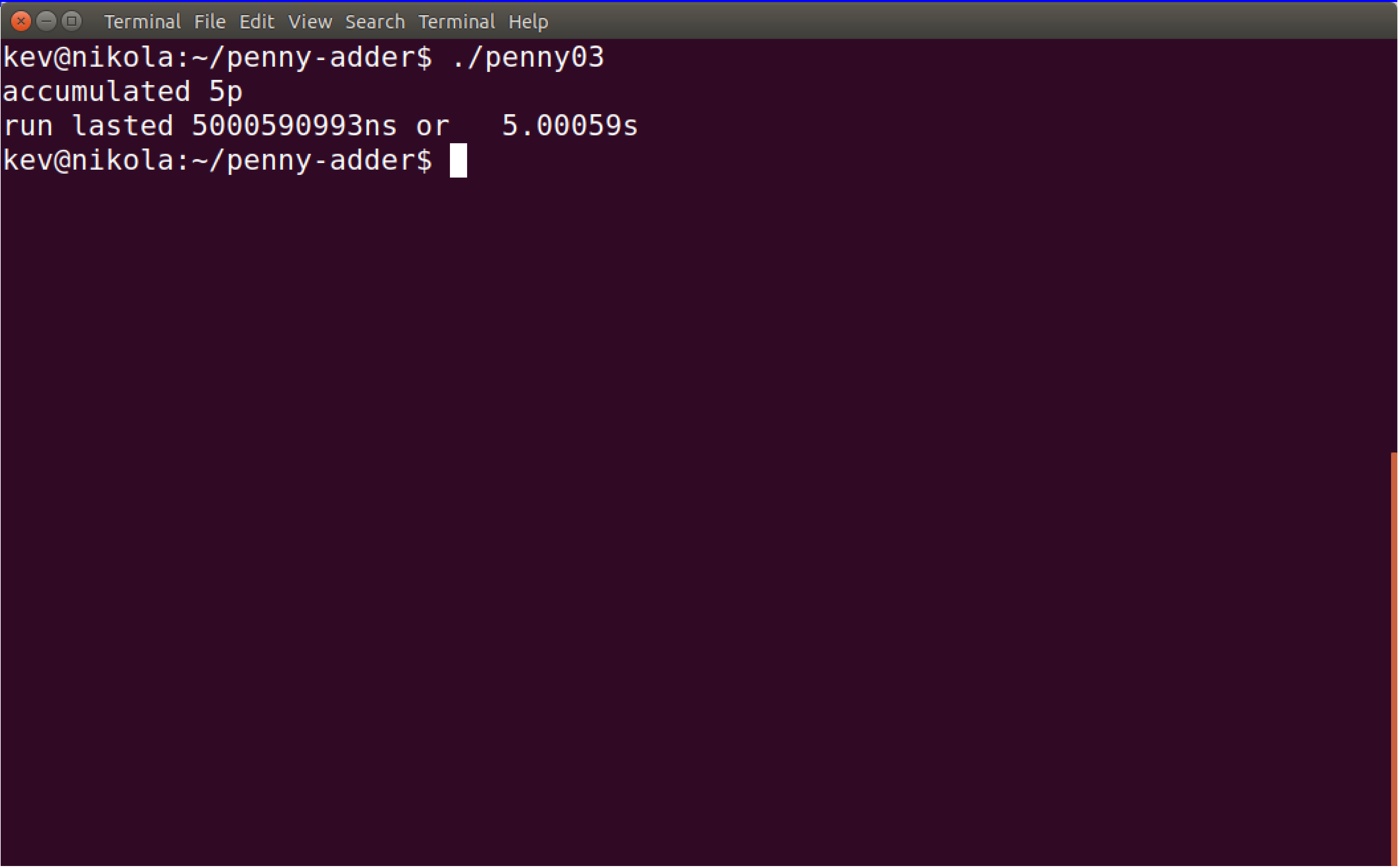
Adding Multiple Pennies

```c
void add_penny(int *balance) {
  int b = *balance;

// 1 second delay (simulating la

  usleep(1000000);

  b = b + 1;
  *balance = b;
}
```

```c
int main(){
  struct timespec start, finish;
  int i;
  long long int difference;
  int account = 0;
  clock_gettime(CLOCK_MONOTONIC, &start);

  for(i=0;i<5;i++){
    add_penny(&account);
  }

  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);

  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);
  return 0;
}
```

How long will it take?
How much money will be accoumulated?

Running the Program



```
kev@nikola:~/penny-adder$ ./penny03
accumulated 5p
run lasted 5000590993ns or    5.00059s
kev@nikola:~/penny-adder$ 
```

Using a Thread

```c
int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;

  clock_gettime(CLOCK_MONOTONIC, &start);

  pthread_t t;

  /* start a thread to call the add_penny function */
  void *add_penny();
  pthread_create(&t, NULL, add_penny, &account);

  /* wait for the thread to finish*/
  pthread_join(t, NULL);

  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n",difference,difference/1000000000.0);
  return 0;
}
```

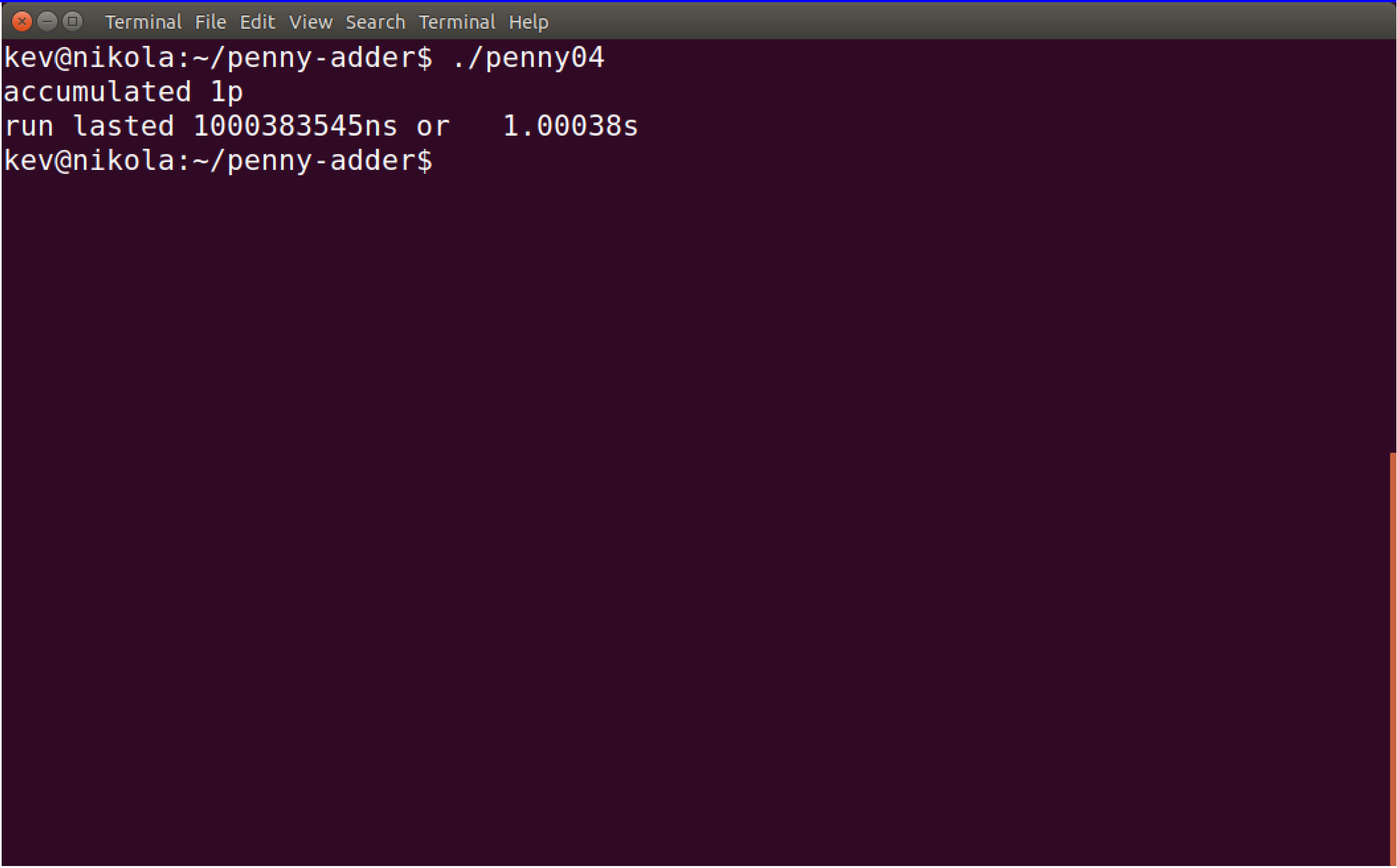
How long will it take?
How much money will be accoumulated?

Thread Function

```c
void *add_penny(void *balance) {
  int *b = balance;
  int c = *b;

// 1 second delay (simulating large calculation time)

  usleep(1000000);

  c = c + 1;
  *b = c;
}
```

How long will it take?
How much money will be accoumulated?

Running th Program



```
kev@nikola:~/penny-adder$ ./penny04
accumulated 1p
run lasted 1000383545ns or    1.00038s
kev@nikola:~/penny-adder$
```

Adding Multiple Pennies With Threads

```c
int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  int i;

  int n = 5;

  clock_gettime(CLOCK_MONOTONIC, &start);

  void *add_penny();
  pthread_t t[n];
  for(i=0;i<n;i++){
    pthread_create(&t[i], NULL, add_penny, &account);
  }
  for(i=0;i<n;i++){
    pthread_join(t[i], NULL);
  }

  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);
  return 0;
}
```

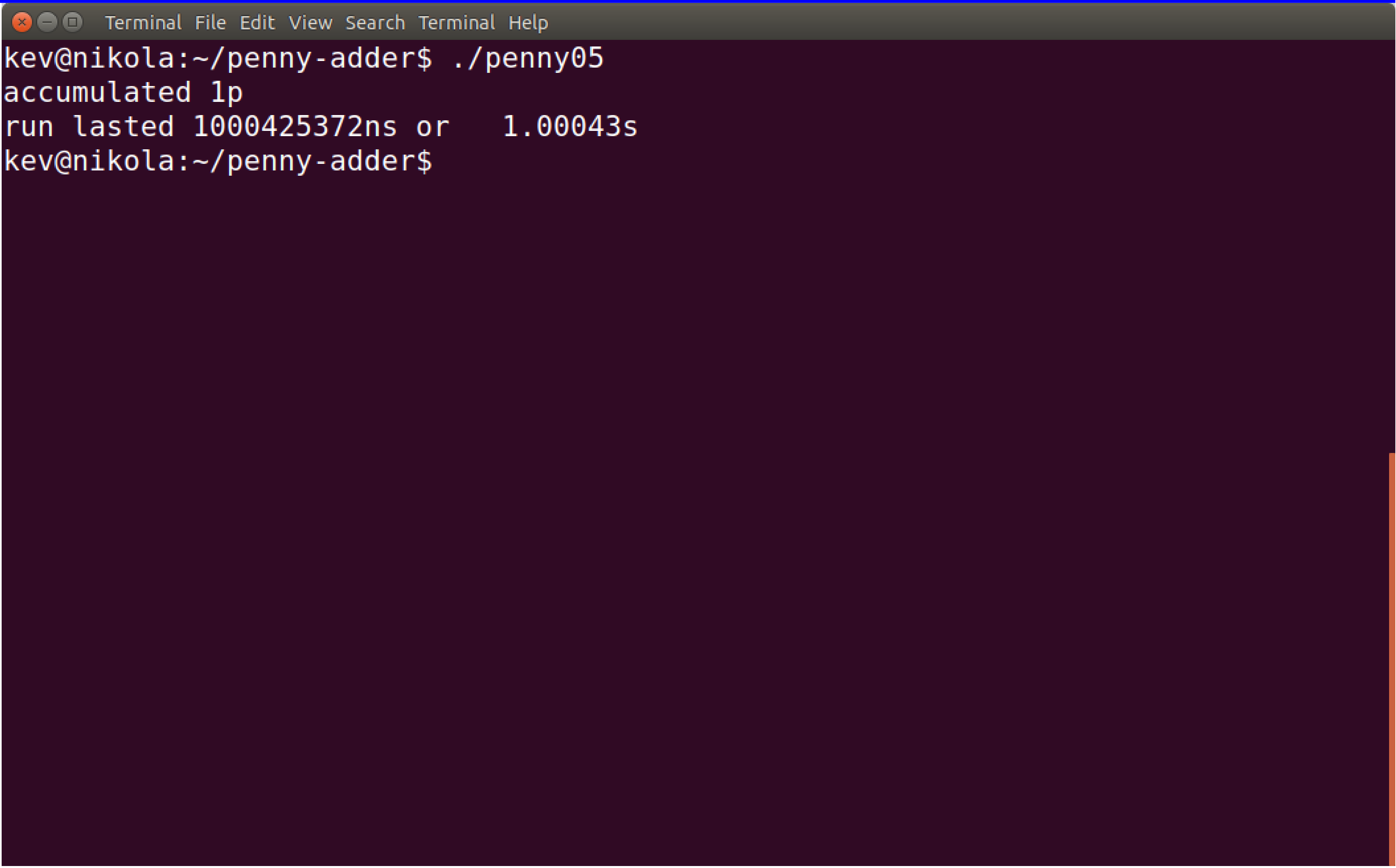
How long will it take?
How much money will be accoumulated?

Thread Function with Delay

```
void *add_penny(void *balance) {
  int *b = balance;
  int c = *b;

// 1 second delay (simulating large calculation time)

  usleep(1000000);

  c = c + 1;
  *b = c;
}
```

How long will it take?
How much money will be accoumulated?

Running the Program



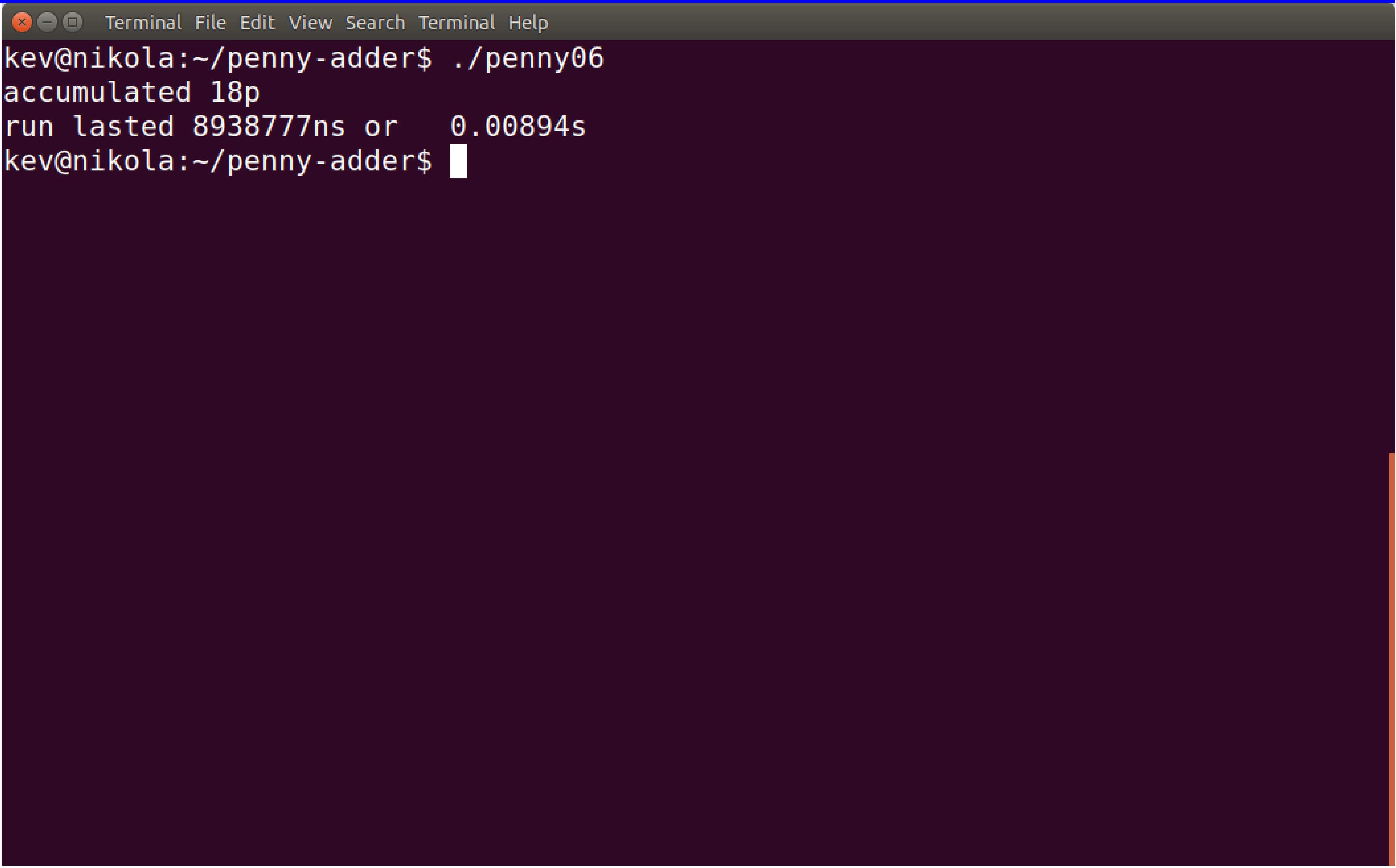
```
kev@nikola:~/penny-adder$ ./penny05
accumulated 1p
run lasted 1000425372ns or   1.00043s
kev@nikola:~/penny-adder$
```

A Shorter Delay

```c
void *add_penny(int *balance) {
  int b = *balance;
/* cause a short delay */
  int i;
  for(i=0;i<100000;i++){
  }
  b = b + 1;
  *balance = b;
}
```

How long will it take?
How much money will be accoumulated?

Running the Program



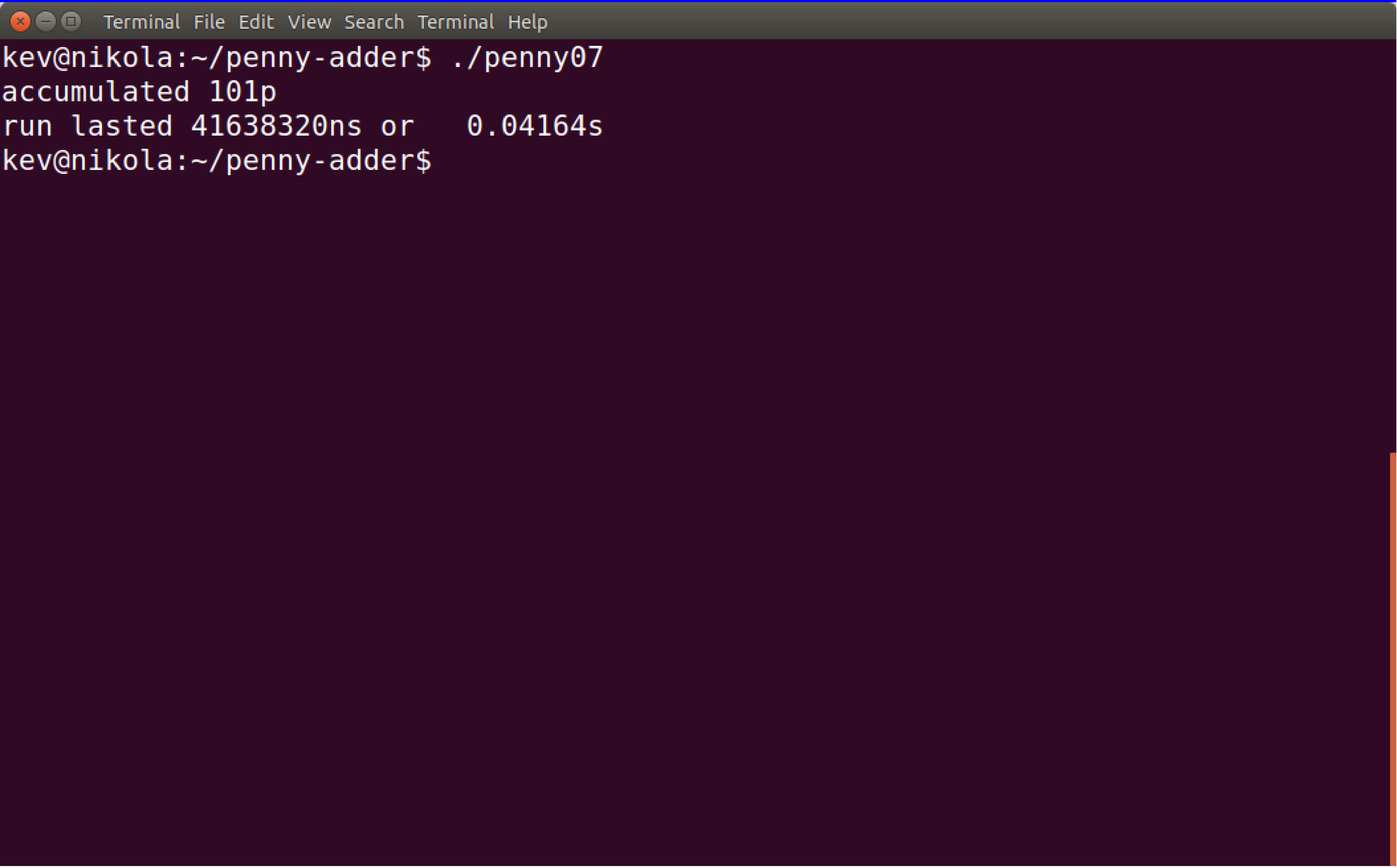
```
kev@nikola:~/penny-adder$ ./penny06
accumulated 18p
run lasted 8938777ns or   0.00894s
kev@nikola:~/penny-adder$ 
```

Initialising a Mutex

```c
pthread_mutex_t mutex;

void initialise_mutex() {
  int result =   pthread_mutex_init(&mutex, NULL);
  if(result != 0){
    printf("problem initialising mutex\n");
    exit(EXIT_FAILURE);
  }
}

int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  int i;

  int n = 1000;
  int result;

  initialise_mutex();
  clock_gettime(CLOCK_MONOTONIC, &start);

  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);

  pthread_mutex_destroy(&mutex);
```

How long will it take?
How much money will be accoumulated?

Running the Program



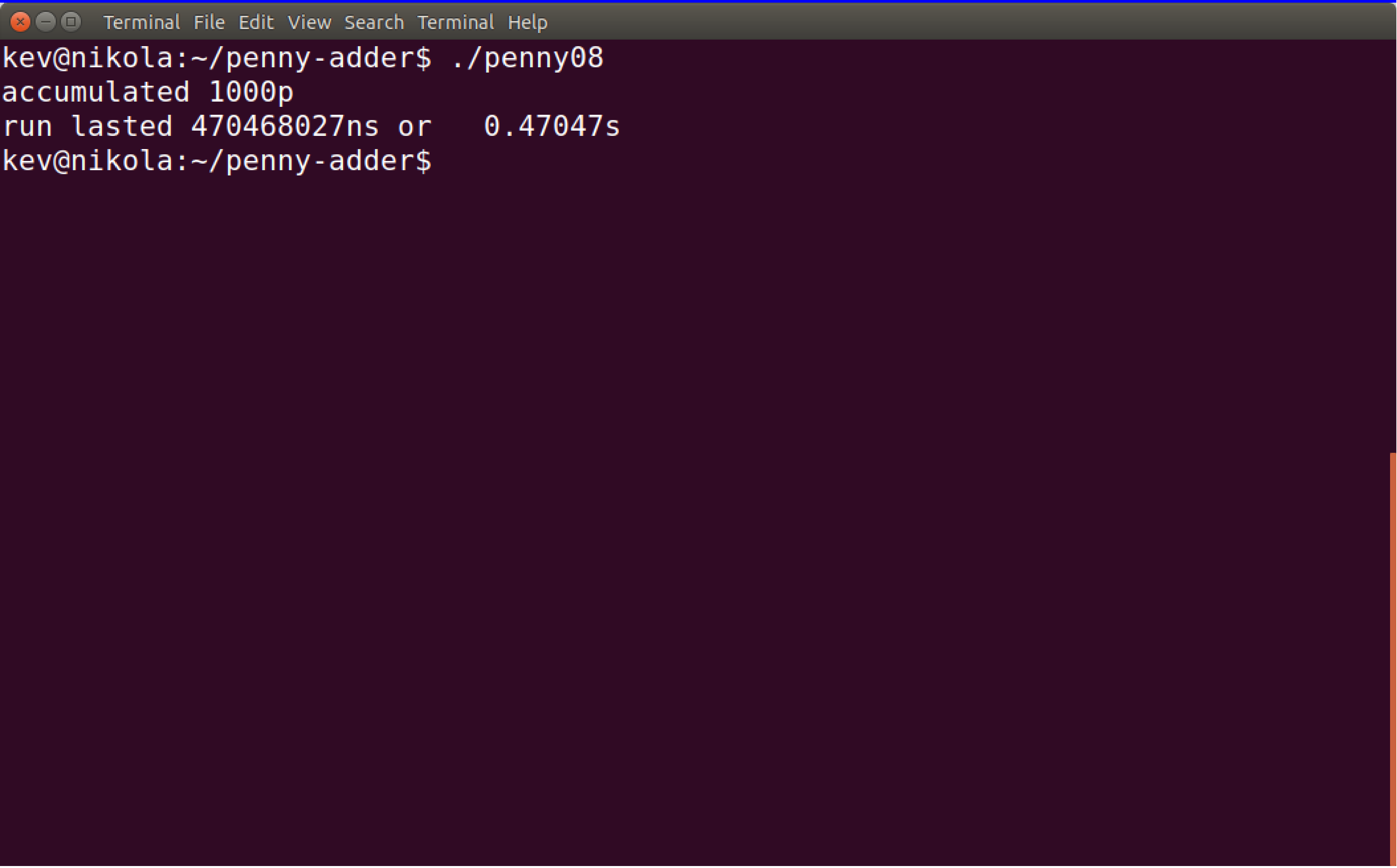
```
kev@nikola:~/penny-adder$ ./penny07
accumulated 101p
run lasted 41638320ns or   0.04164s
kev@nikola:~/penny-adder$
```

Adding Multiple Pennies With a Mutex

```
void *add_penny(void *bal) {
  int *balance = bal;

  int result = pthread_mutex_lock(&mutex);
  if(result != 0){
    printf("problem locking mutex\n");
    exit(EXIT_FAILURE);
  }

  int b = *balance;
/* cause a short delay */
  int i;
  for(i=0;i<100000;i++){
  }
  b = b + 1;
  *balance = b;

  result = pthread_mutex_unlock(&mutex);
  if(result != 0){
    printf("problem unlocking mutex\n");
```

How long will it take?
How much money will be accoumulated?

Running the Program



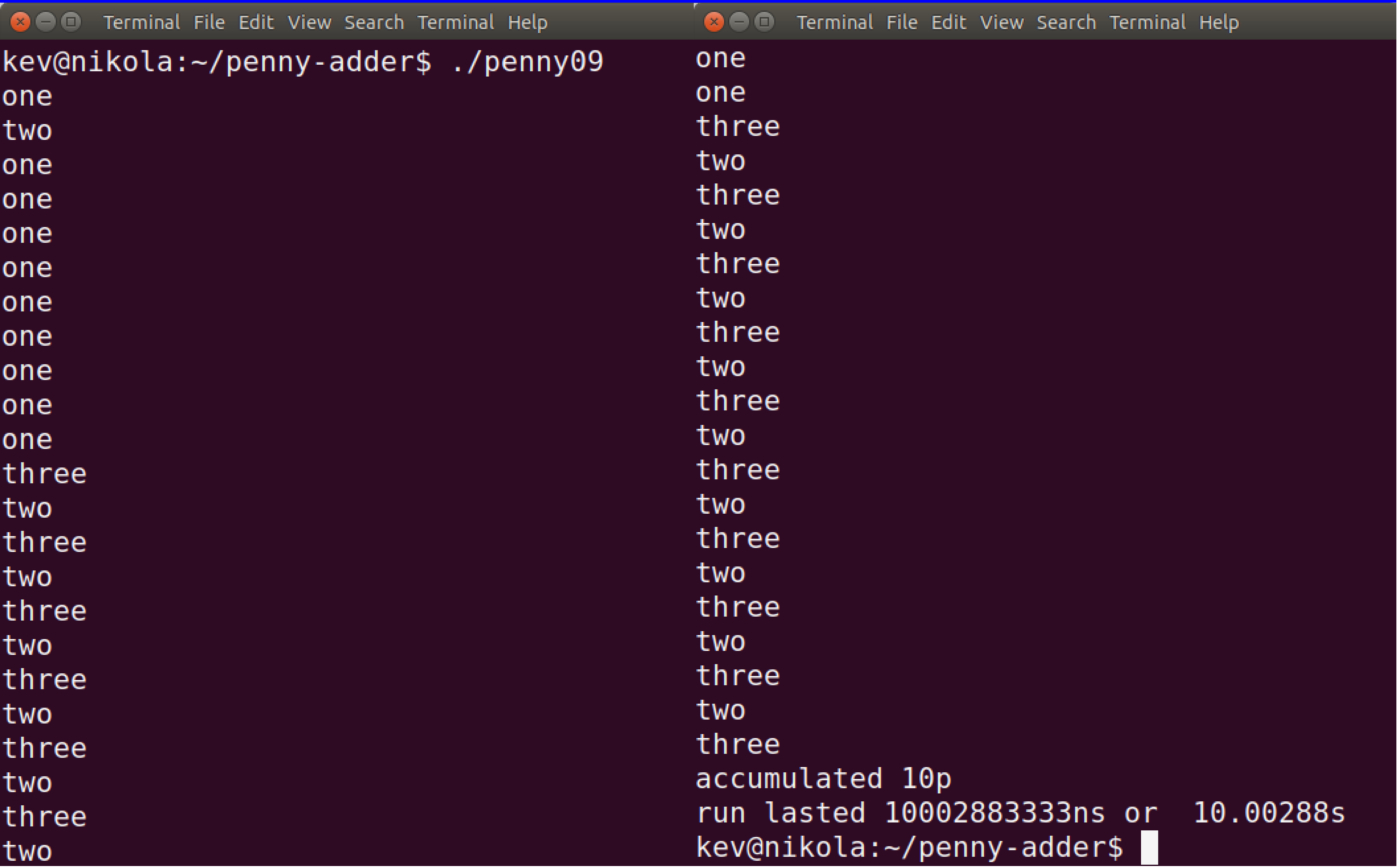
```
kev@nikola:~/penny-adder$ ./penny08
accumulated 1000p
run lasted 470468027ns or   0.47047s
kev@nikola:~/penny-adder$
```

A Mutex Combined With A Delay

```c
void *add_penny(void *bal) {
  int *balance = bal;
  printf("one\n");

  int result = pthread_mutex_lock(&mutex);
  if(result != 0){
    printf("problem locking mutex\n");
    exit(EXIT_FAILURE);
  }

  printf("two\n");

  int b = *balance;
/* cause a substantial delay */
  usleep(1000000);
  b = b + 1;
  *balance = b;

  printf("three\n");

  result = pthread_mutex_unlock(&mutex);
  if(result != 0){
    printf("problem unlocking mutex\n");
    exit(EXIT_FAILURE);
  }
}
```

How long will it take?
How much money will be accoumulated?

Running the Program

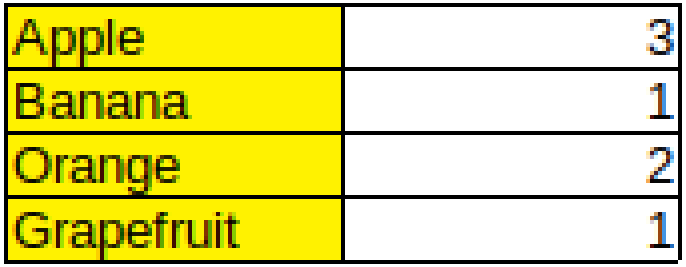

Terminal File Edit View Search Terminal Help
Terminal File Edit View Search Terminal Help

```
kev@nikola:~/penny-adder$ ./penny09
one
two
one
one
one
one
one
one
one
one
one
three
two
three
two
three
two
three
two
three
two
three
two
```

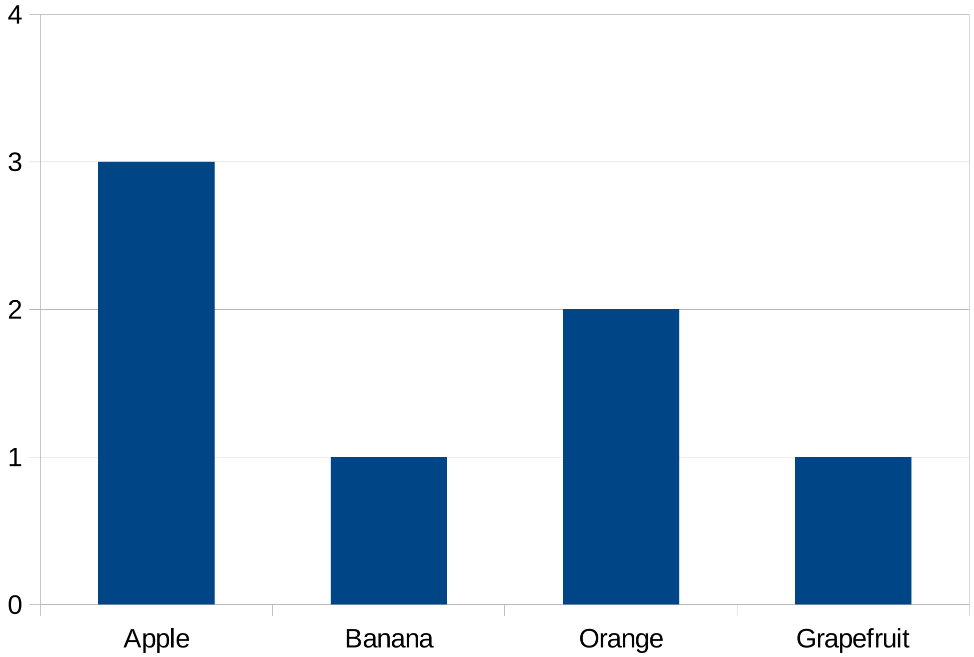
```
one
one
three
two
three
two
three
two
three
two
three
two
three
two
three
two
three
two
three
two
three
accumulated 10p
run lasted 10002883333ns or  10.00288s
kev@nikola:~/penny-adder$ 
```

Histograms

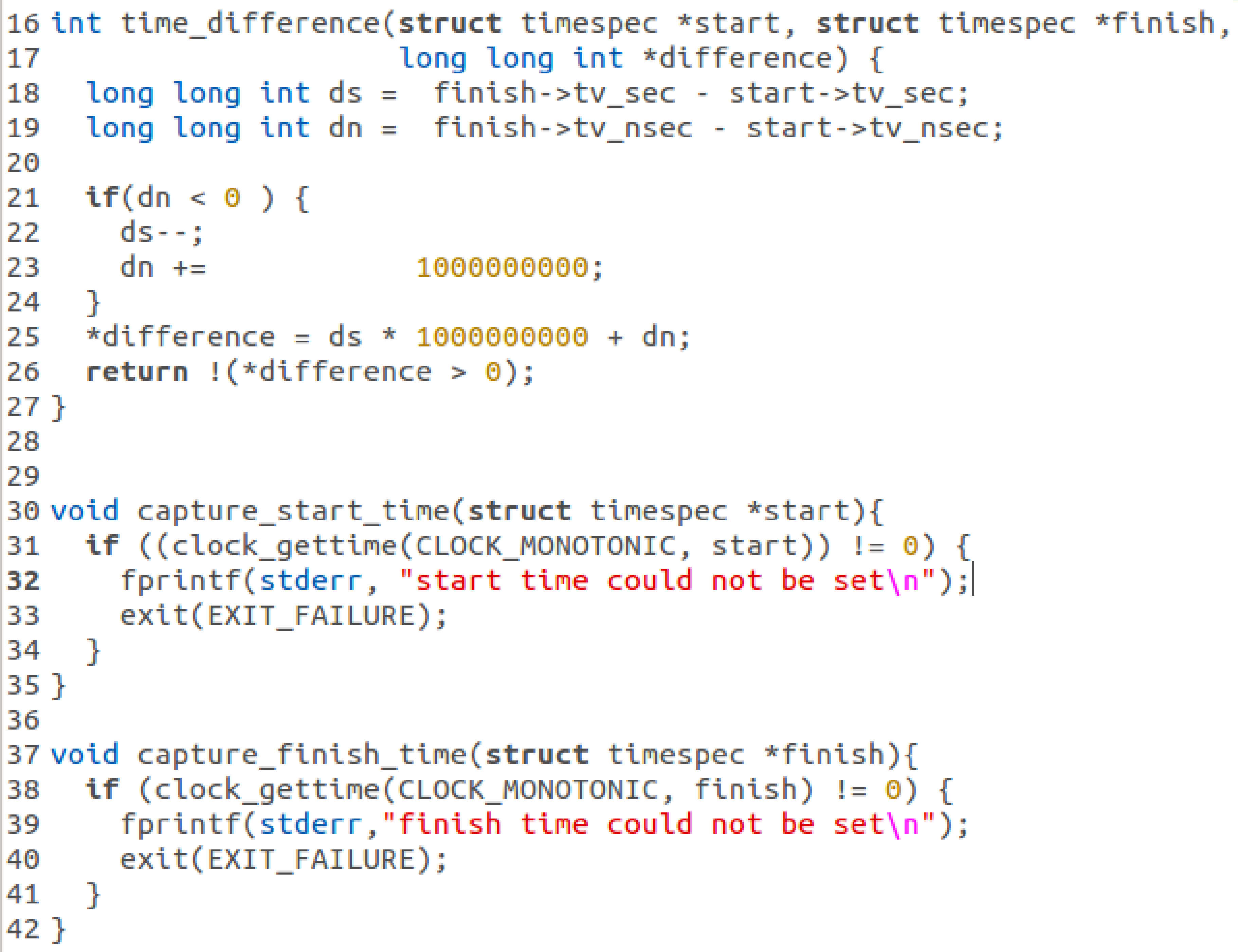
Apple
Apple
Orange
Banana
Apple
Orange
Grapefruit



Apple	3
Banana	1
Orange	2
Grapefruit	1

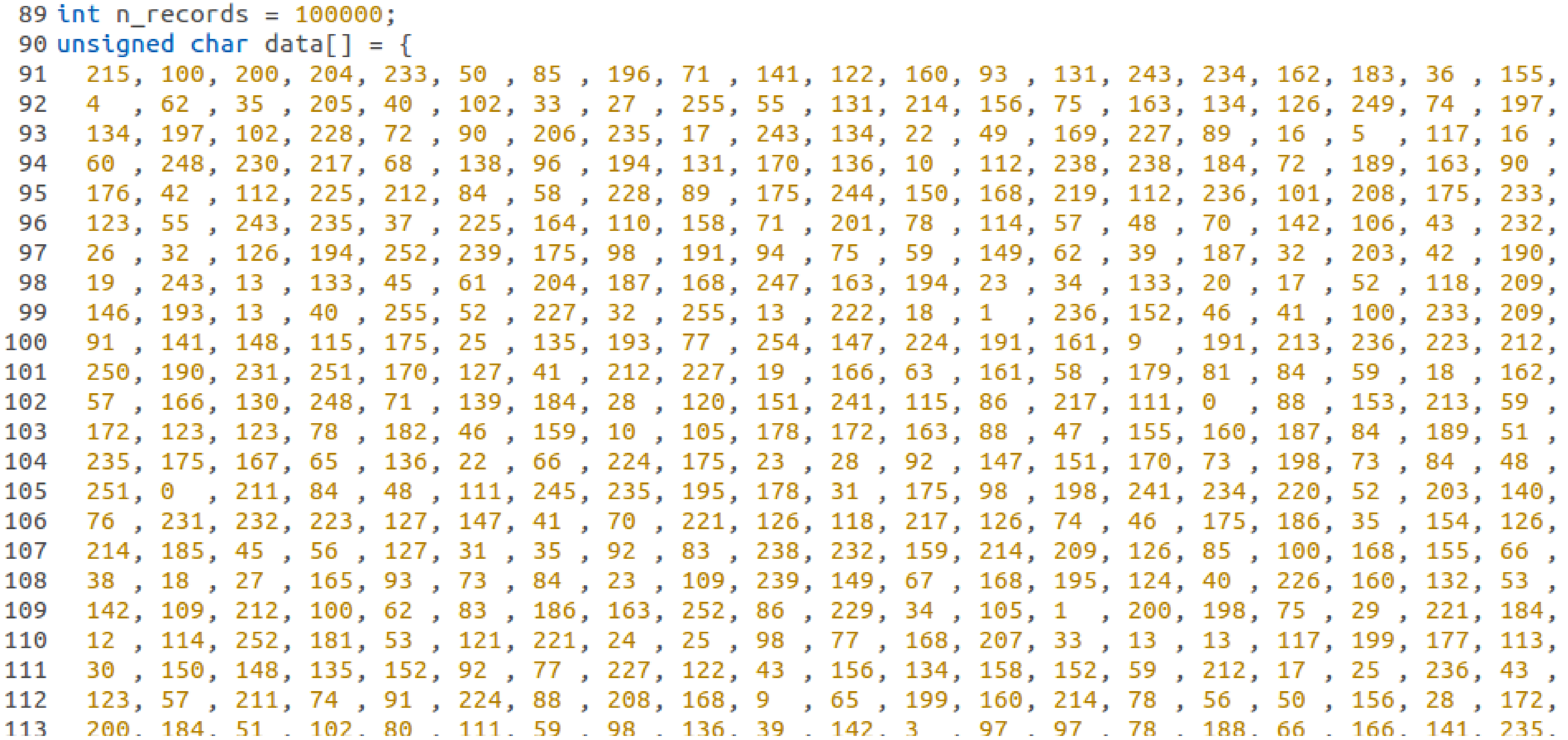


time_diff.c

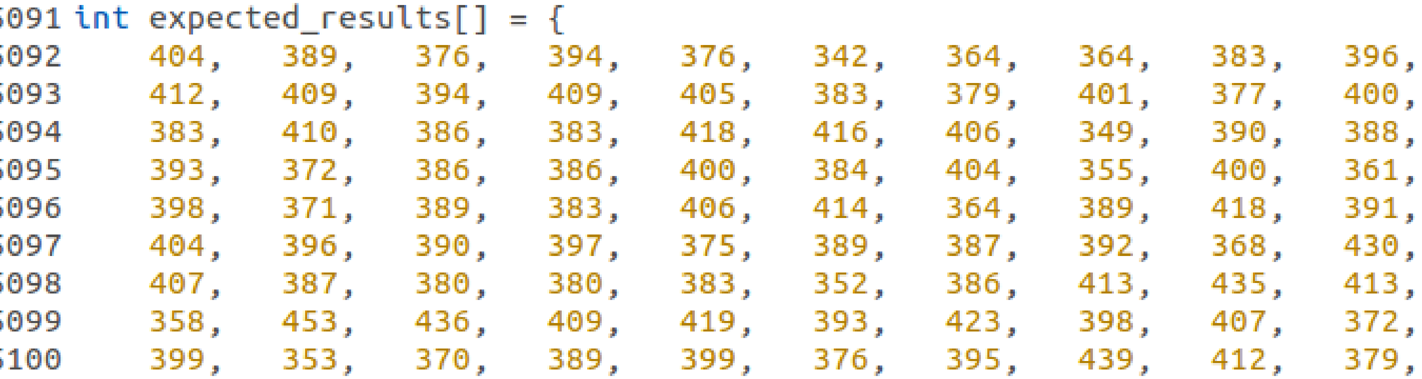


```c
int time_difference(struct timespec *start, struct timespec *finish,
                    long long int *difference) {
  long long int ds =  finish->tv_sec - start->tv_sec;
  long long int dn =  finish->tv_nsec - start->tv_nsec;

  if(dn < 0 ) {
    ds--;
    dn +=            1000000000;
  }
  *difference = ds * 1000000000 + dn;
  return !(*difference > 0);
}


void capture_start_time(struct timespec *start){
  if ((clock_gettime(CLOCK_MONOTONIC, start)) != 0) {
    fprintf(stderr, "start time could not be set\n");
    exit(EXIT_FAILURE);
  }
}

void capture_finish_time(struct timespec *finish){
  if (clock_gettime(CLOCK_MONOTONIC, finish) != 0) {
    fprintf(stderr,"finish time could not be set\n");
    exit(EXIT_FAILURE);
  }
}
```

histogram_common.c

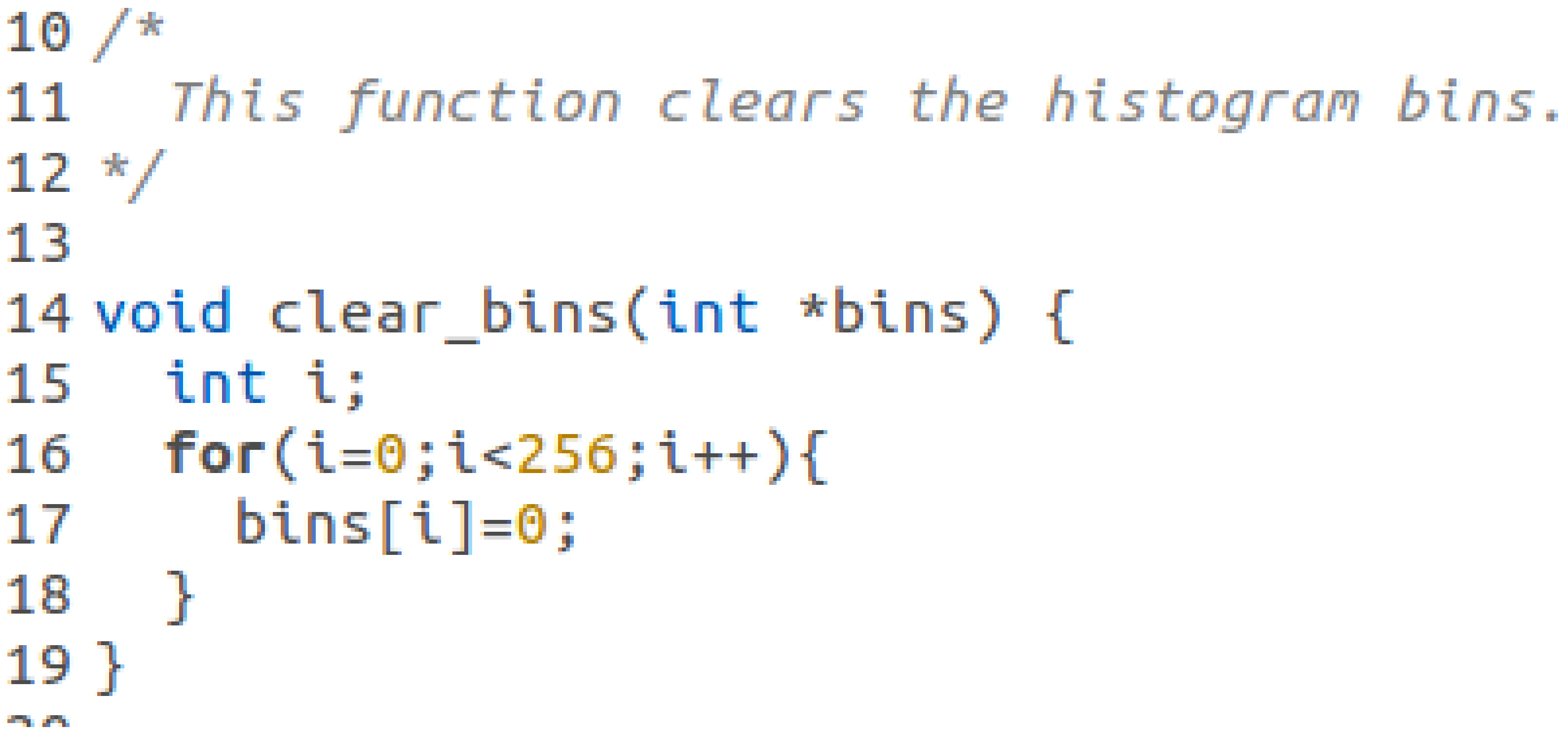


```
int n_records = 100000;
unsigned char data[] = {
  215, 100, 200, 204, 233, 50 , 85 , 196, 71 , 141, 122, 160, 93 , 131, 243, 234, 162, 183, 36 , 155,
  4  , 62 , 35 , 205, 40 , 102, 33 , 27 , 255, 55 , 131, 214, 156, 75 , 163, 134, 126, 249, 74 , 197,
  134, 197, 102, 228, 72 , 90 , 206, 235, 17 , 243, 134, 22 , 49 , 169, 227, 89 , 16 , 5  , 117, 16 ,
  60 , 248, 230, 217, 68 , 138, 96 , 194, 131, 170, 136, 10 , 112, 238, 238, 184, 72 , 189, 163, 90 ,
  176, 42 , 112, 225, 212, 84 , 58 , 228, 89 , 175, 244, 150, 168, 219, 112, 236, 101, 208, 175, 233,
  123, 55 , 243, 235, 37 , 225, 164, 110, 158, 71 , 201, 78 , 114, 57 , 48 , 70 , 142, 106, 43 , 232,
  26 , 32 , 126, 194, 252, 239, 175, 98 , 191, 94 , 75 , 59 , 149, 62 , 39 , 187, 32 , 203, 42 , 190,
  19 , 243, 13 , 133, 45 , 61 , 204, 187, 168, 247, 163, 194, 23 , 34 , 133, 20 , 17 , 52 , 118, 209,
  146, 193, 13 , 40 , 255, 52 , 227, 32 , 255, 13 , 222, 18 , 1  , 236, 152, 46 , 41 , 100, 233, 209,
  91 , 141, 148, 115, 175, 25 , 135, 193, 77 , 254, 147, 224, 191, 161, 9  , 191, 213, 236, 223, 212,
  250, 190, 231, 251, 170, 127, 41 , 212, 227, 19 , 166, 63 , 161, 58 , 179, 81 , 84 , 59 , 18 , 162,
  57 , 166, 130, 248, 71 , 139, 184, 28 , 120, 151, 241, 115, 86 , 217, 111, 0  , 88 , 153, 213, 59 ,
  172, 123, 123, 78 , 182, 46 , 159, 10 , 105, 178, 172, 163, 88 , 47 , 155, 160, 187, 84 , 189, 51 ,
  235, 175, 167, 65 , 136, 22 , 66 , 224, 175, 23 , 28 , 92 , 147, 151, 170, 73 , 198, 73 , 84 , 48 ,
  251, 0  , 211, 84 , 48 , 111, 245, 235, 195, 178, 31 , 175, 98 , 198, 241, 234, 220, 52 , 203, 140,
  76 , 231, 232, 223, 127, 147, 41 , 70 , 221, 126, 118, 217, 126, 74 , 46 , 175, 186, 35 , 154, 126,
  214, 185, 45 , 56 , 127, 31 , 35 , 92 , 83 , 238, 232, 159, 214, 209, 126, 85 , 100, 168, 155, 66 ,
  38 , 18 , 27 , 165, 93 , 73 , 84 , 23 , 109, 239, 149, 67 , 168, 195, 124, 40 , 226, 160, 132, 53 ,
  142, 109, 212, 100, 62 , 83 , 186, 163, 252, 86 , 229, 34 , 105, 1  , 200, 198, 75 , 29 , 221, 184,
  12 , 114, 252, 181, 53 , 121, 221, 24 , 25 , 98 , 77 , 168, 207, 33 , 13 , 13 , 117, 199, 177, 113,
  30 , 150, 148, 135, 152, 92 , 77 , 227, 122, 43 , 156, 134, 158, 152, 59 , 212, 17 , 25 , 236, 43 ,
  123, 57 , 211, 74 , 91 , 224, 88 , 208, 168, 9  , 65 , 199, 160, 214, 78 , 56 , 50 , 156, 28 , 172,
  200, 184, 51 , 102, 80 , 111, 59 , 98 , 136, 39 , 142, 3  , 97 , 97 , 78 , 188, 66 , 166, 141, 235,
```



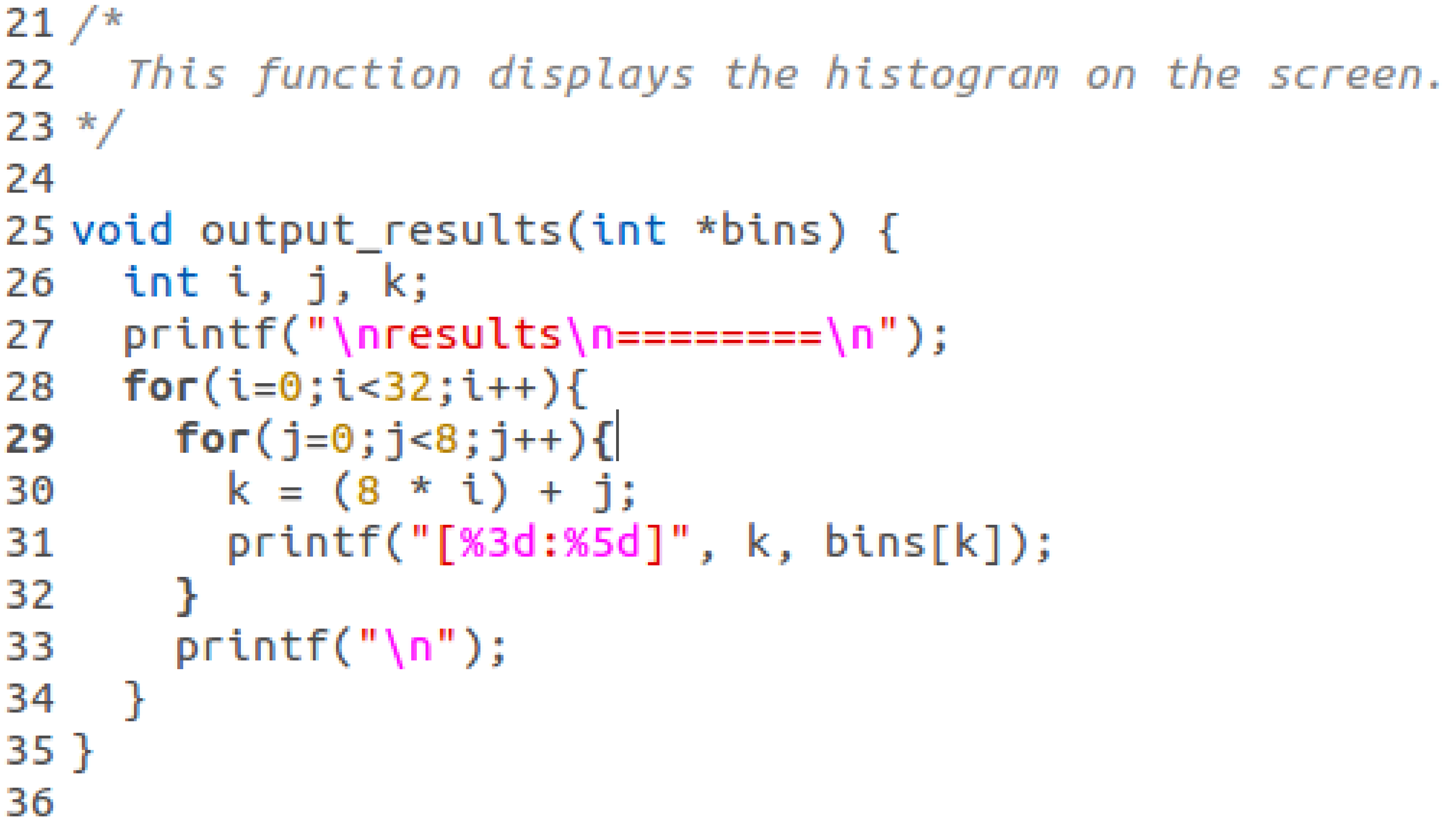
```
int expected_results[] = {
    404,    389,    376,    394,    376,    342,    364,    364,    383,    396,
    412,    409,    394,    409,    405,    383,    379,    401,    377,    400,
    383,    410,    386,    383,    418,    416,    406,    349,    390,    388,
    393,    372,    386,    386,    400,    384,    404,    355,    400,    361,
    398,    371,    389,    383,    406,    414,    364,    389,    418,    391,
    404,    396,    390,    397,    375,    389,    387,    392,    368,    430,
    407,    387,    380,    380,    383,    352,    386,    413,    435,    413,
    358,    453,    436,    409,    419,    393,    423,    398,    407,    372,
    399,    353,    370,    389,    399,    376,    395,    439,    412,    379,
```

histogram_common.c



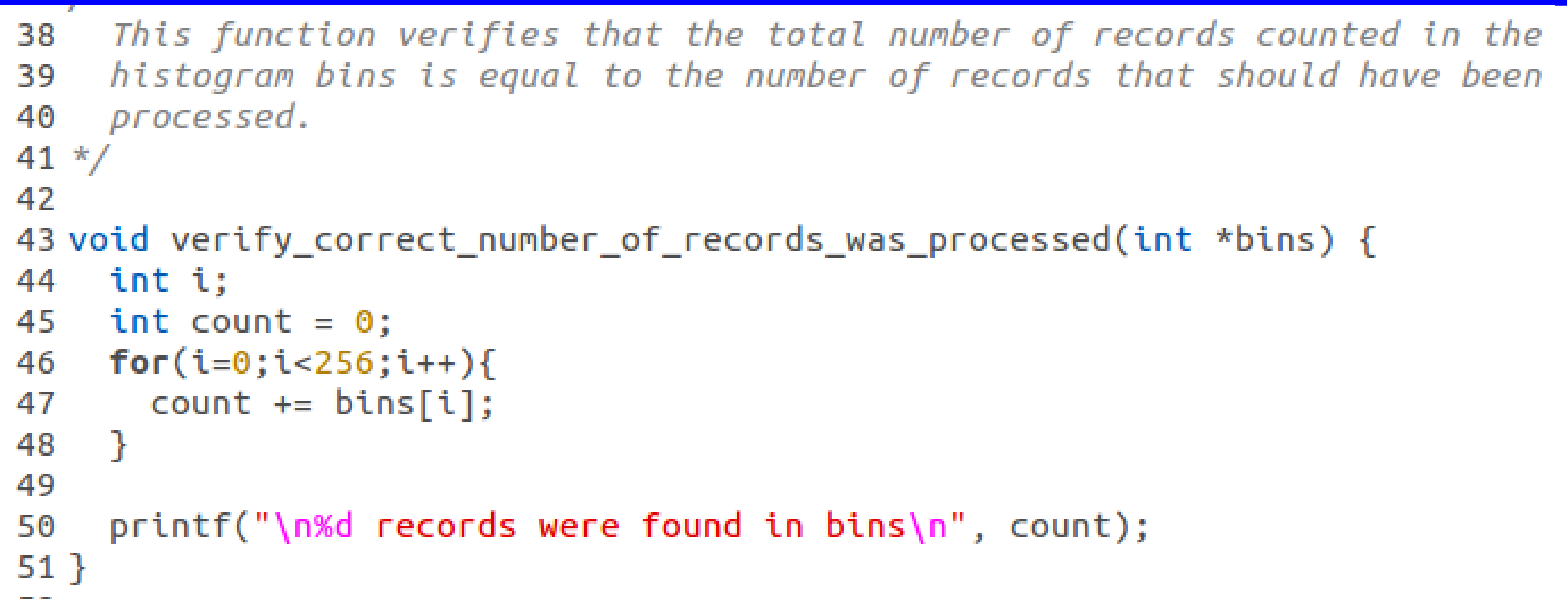
```c
/*
  This function clears the histogram bins.
*/

void clear_bins(int *bins) {
  int i;
  for(i=0;i<256;i++){
    bins[i]=0;
  }
}
```

histogram_common.c



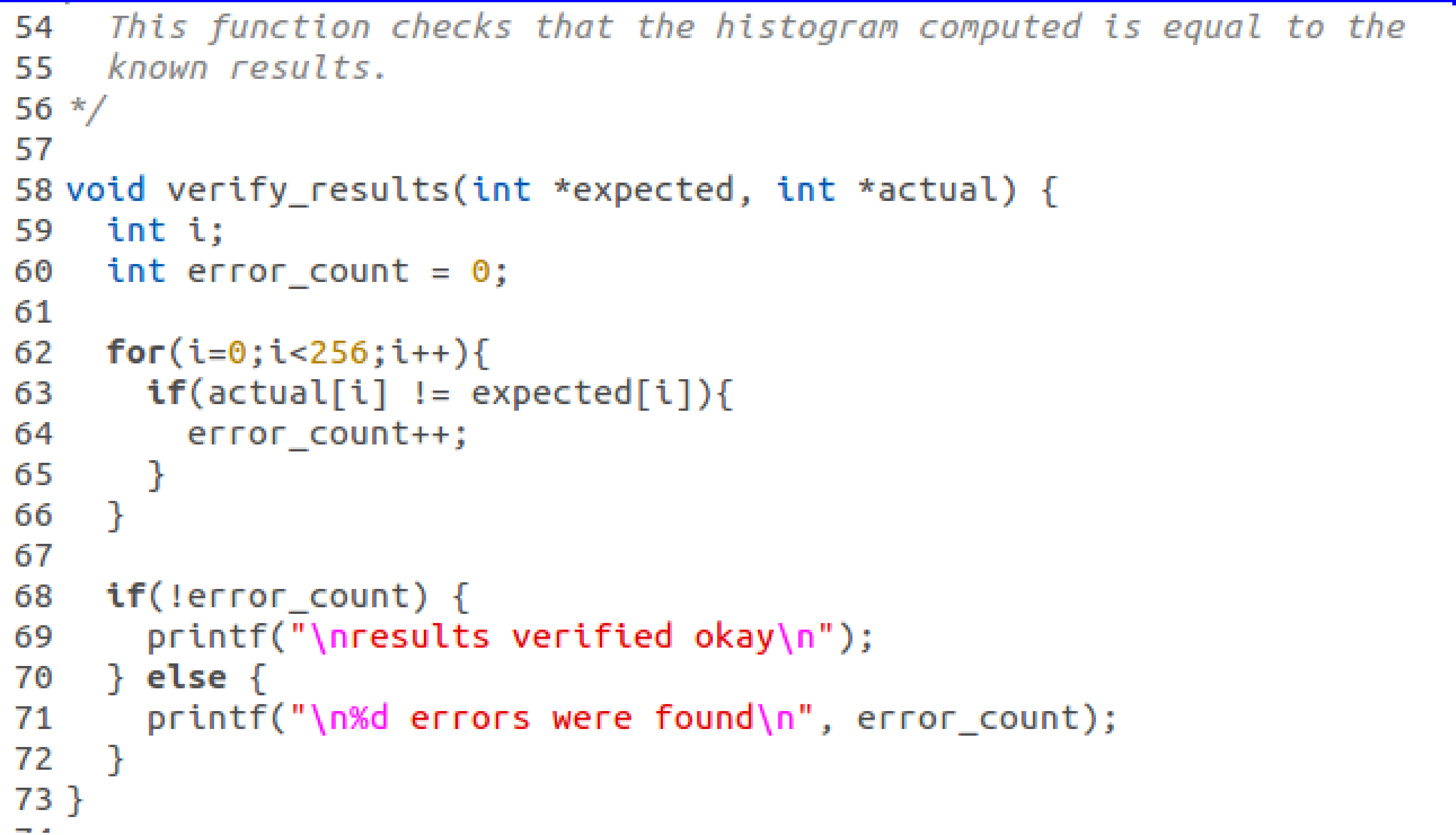
```
/*
  This function displays the histogram on the screen.
*/

void output_results(int *bins) {
  int i, j, k;
  printf("\nresults\n========\n");
  for(i=0;i<32;i++){
    for(j=0;j<8;j++){
      k = (8 * i) + j;
      printf("[%3d:%5d]", k, bins[k]);
    }
    printf("\n");
  }
}

```

histogram_common.c



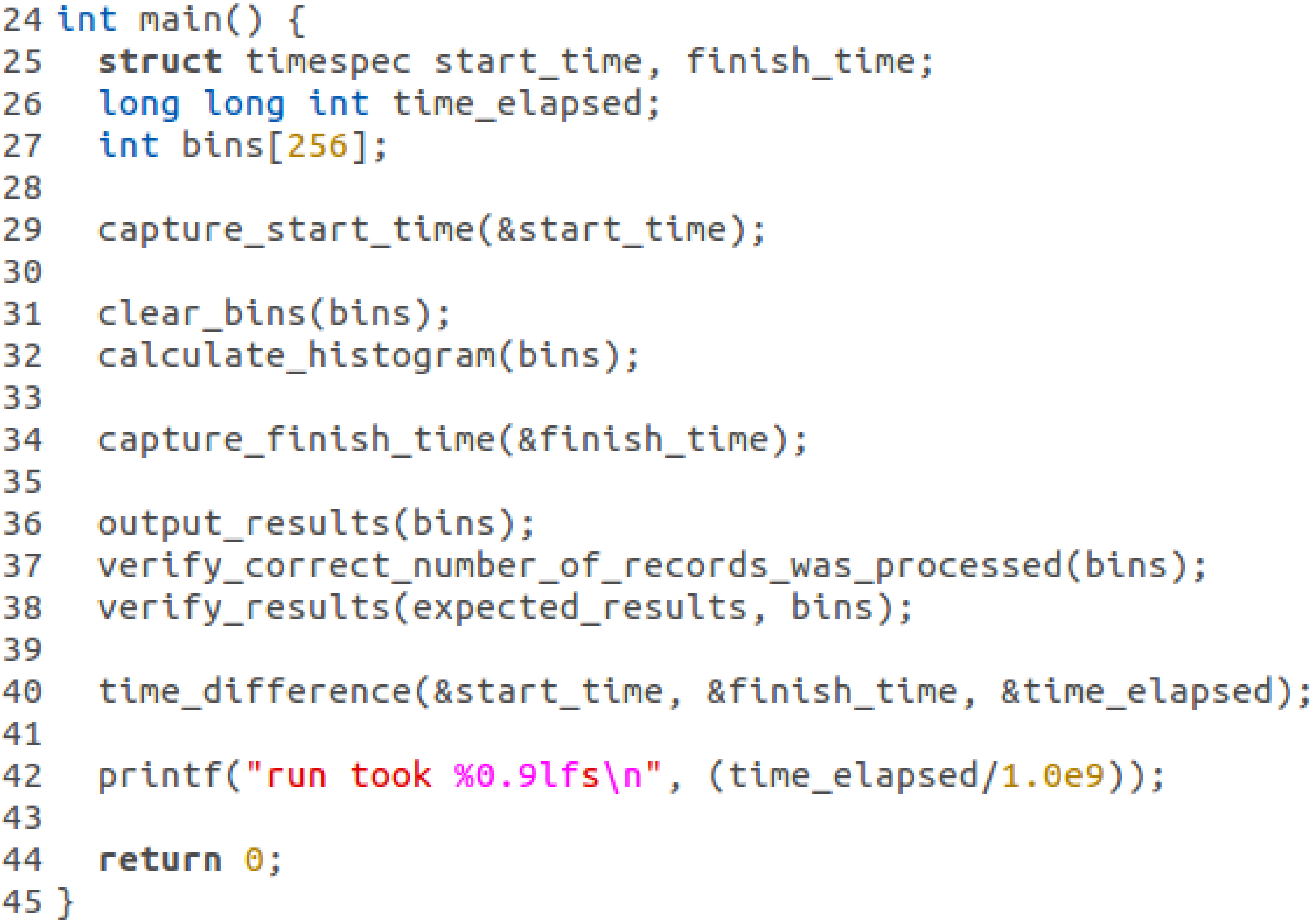
```c
  This function verifies that the total number of records counted in the
  histogram bins is equal to the number of records that should have been
  processed.
*/

void verify_correct_number_of_records_was_processed(int *bins) {
  int i;
  int count = 0;
  for(i=0;i<256;i++){
    count += bins[i];
  }

  printf("\n%d records were found in bins\n", count);
}
```

histogram_common.c



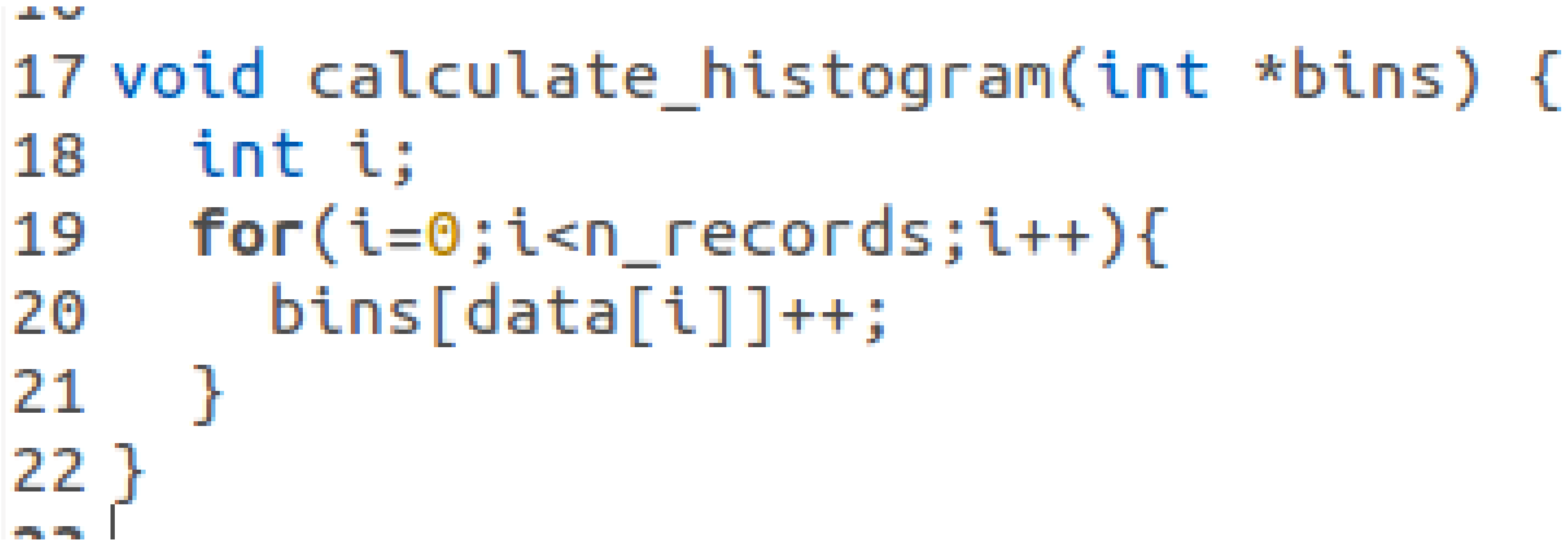
```c
  This function checks that the histogram computed is equal to the
  known results.
*/

void verify_results(int *expected, int *actual) {
  int i;
  int error_count = 0;

  for(i=0;i<256;i++){
    if(actual[i] != expected[i]){
      error_count++;
    }
  }

  if(!error_count) {
    printf("\nresults verified okay\n");
  } else {
    printf("\n%d errors were found\n", error_count);
  }
}
```

histogram00

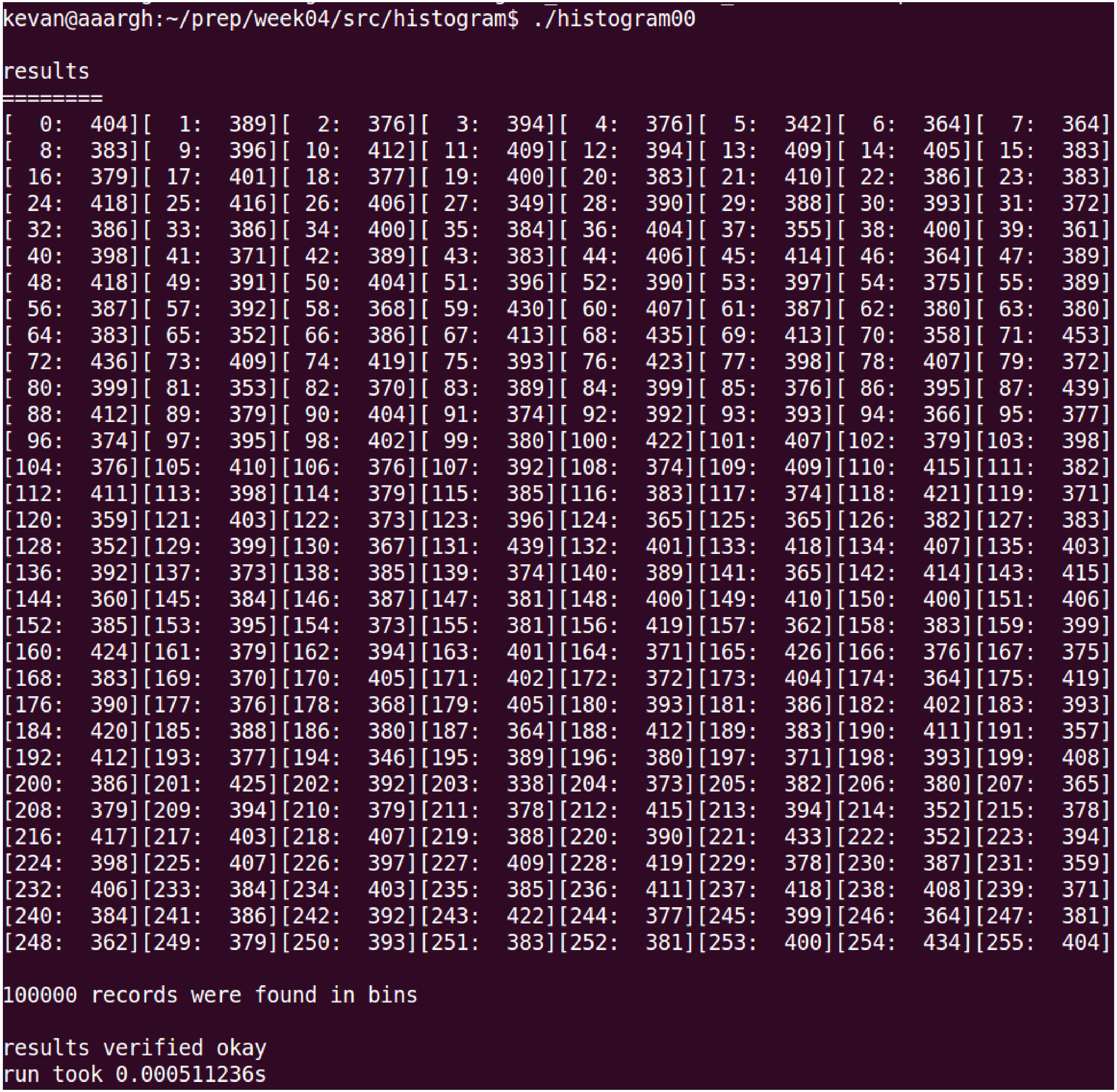


```c
int main() {
  struct timespec start_time, finish_time;
  long long int time_elapsed;
  int bins[256];

  capture_start_time(&start_time);

  clear_bins(bins);
  calculate_histogram(bins);

  capture_finish_time(&finish_time);

  output_results(bins);
  verify_correct_number_of_records_was_processed(bins);
  verify_results(expected_results, bins);

  time_difference(&start_time, &finish_time, &time_elapsed);

  printf("run took %0.9lfs\n", (time_elapsed/1.0e9));

  return 0;
}
```

histogram00

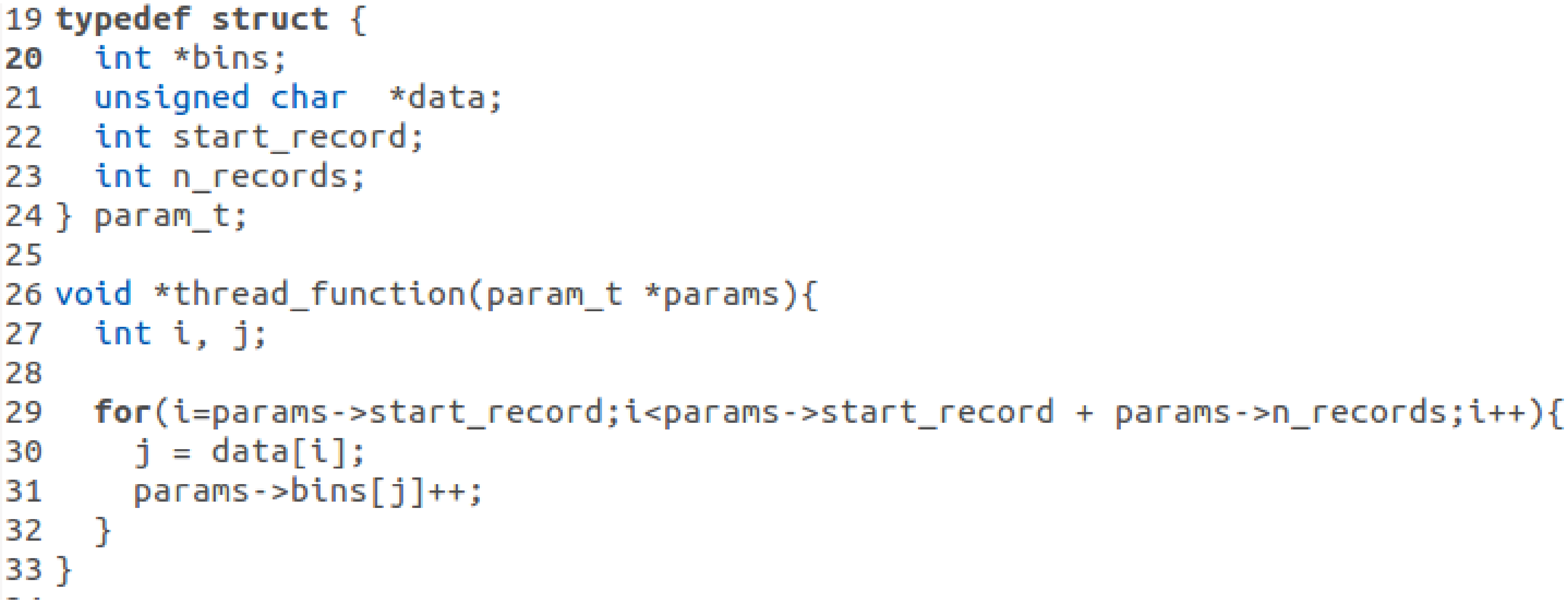


```
void calculate_histogram(int *bins) {
  int i;
  for(i=0;i<n_records;i++){
    bins[data[i]]++;
  }
}
```



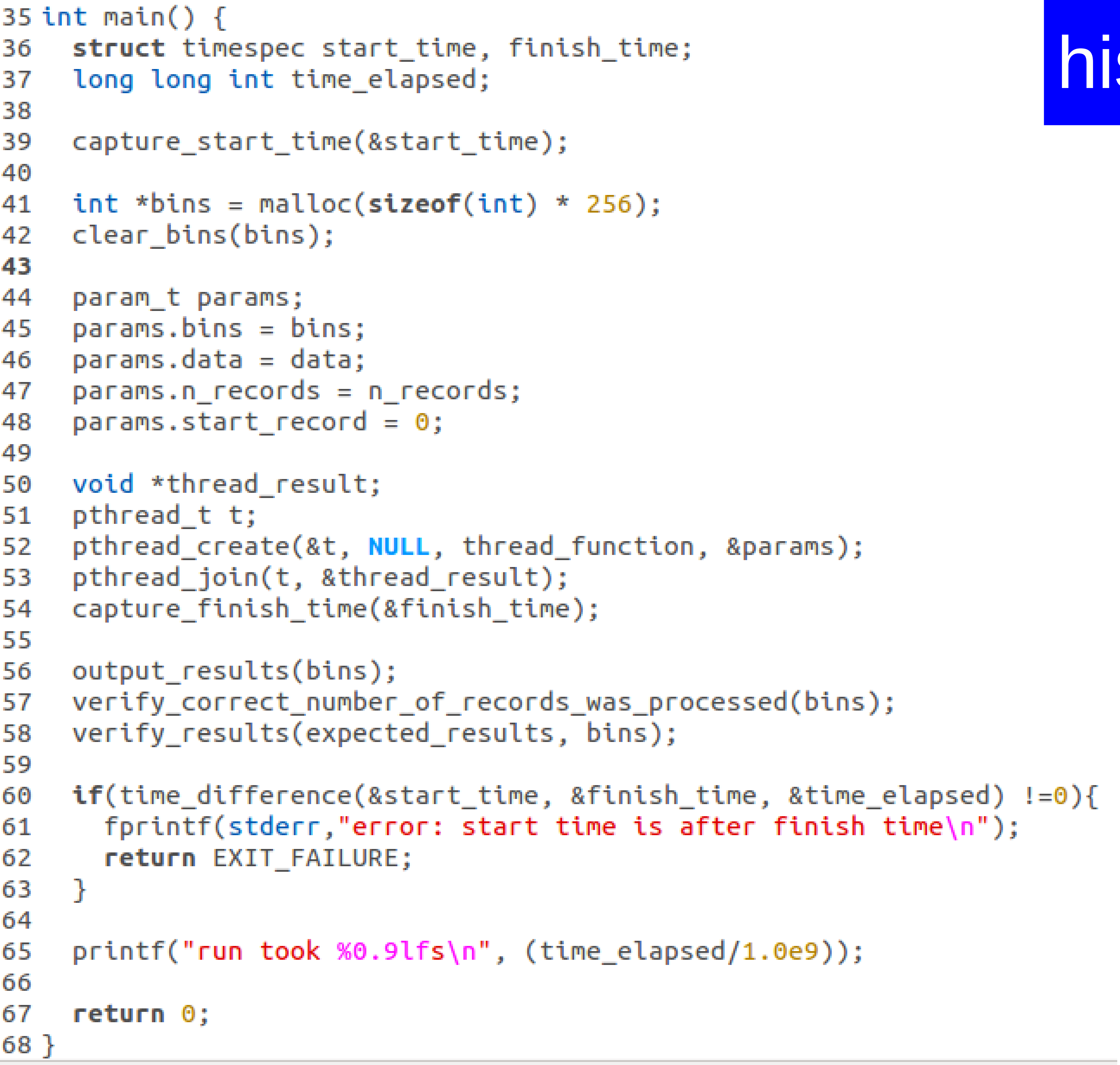
```
kevan@aaargh:~/prep/week04/src/histogram$ ./histogram00

results
=========
[  0:  404][  1:  389][  2:  376][  3:  394][  4:  376][  5:  342][  6:  364][  7:  364]
[  8:  383][  9:  396][ 10:  412][ 11:  409][ 12:  394][ 13:  409][ 14:  405][ 15:  383]
[ 16:  379][ 17:  401][ 18:  377][ 19:  400][ 20:  383][ 21:  410][ 22:  386][ 23:  383]
[ 24:  418][ 25:  416][ 26:  406][ 27:  349][ 28:  390][ 29:  388][ 30:  393][ 31:  372]
[ 32:  386][ 33:  386][ 34:  400][ 35:  384][ 36:  404][ 37:  355][ 38:  400][ 39:  361]
[ 40:  398][ 41:  371][ 42:  389][ 43:  383][ 44:  406][ 45:  414][ 46:  364][ 47:  389]
[ 48:  418][ 49:  391][ 50:  404][ 51:  396][ 52:  390][ 53:  397][ 54:  375][ 55:  389]
[ 56:  387][ 57:  392][ 58:  368][ 59:  430][ 60:  407][ 61:  387][ 62:  380][ 63:  380]
[ 64:  383][ 65:  352][ 66:  386][ 67:  413][ 68:  435][ 69:  413][ 70:  358][ 71:  453]
[ 72:  436][ 73:  409][ 74:  419][ 75:  393][ 76:  423][ 77:  398][ 78:  407][ 79:  372]
[ 80:  399][ 81:  353][ 82:  370][ 83:  389][ 84:  399][ 85:  376][ 86:  395][ 87:  439]
[ 88:  412][ 89:  379][ 90:  404][ 91:  374][ 92:  392][ 93:  393][ 94:  366][ 95:  377]
[ 96:  374][ 97:  395][ 98:  402][ 99:  380][100:  422][101:  407][102:  379][103:  398]
[104:  376][105:  410][106:  376][107:  392][108:  374][109:  409][110:  415][111:  382]
[112:  411][113:  398][114:  379][115:  385][116:  383][117:  374][118:  421][119:  371]
[120:  359][121:  403][122:  373][123:  396][124:  365][125:  365][126:  382][127:  383]
[128:  352][129:  399][130:  367][131:  439][132:  401][133:  418][134:  407][135:  403]
[136:  392][137:  373][138:  385][139:  374][140:  389][141:  365][142:  414][143:  415]
[144:  360][145:  384][146:  387][147:  381][148:  400][149:  410][150:  400][151:  406]
[152:  385][153:  395][154:  373][155:  381][156:  419][157:  362][158:  383][159:  399]
[160:  424][161:  379][162:  394][163:  401][164:  371][165:  426][166:  376][167:  375]
[168:  383][169:  370][170:  405][171:  402][172:  372][173:  404][174:  364][175:  419]
[176:  390][177:  376][178:  368][179:  405][180:  393][181:  386][182:  402][183:  393]
[184:  420][185:  388][186:  380][187:  364][188:  412][189:  383][190:  411][191:  357]
[192:  412][193:  377][194:  346][195:  389][196:  380][197:  371][198:  393][199:  408]
[200:  386][201:  425][202:  392][203:  338][204:  373][205:  382][206:  380][207:  365]
[208:  379][209:  394][210:  379][211:  378][212:  415][213:  394][214:  352][215:  378]
[216:  417][217:  403][218:  407][219:  388][220:  390][221:  433][222:  352][223:  394]
[224:  398][225:  407][226:  397][227:  409][228:  419][229:  378][230:  387][231:  359]
[232:  406][233:  384][234:  403][235:  385][236:  411][237:  418][238:  408][239:  371]
[240:  384][241:  386][242:  392][243:  422][244:  377][245:  399][246:  364][247:  381]
[248:  362][249:  379][250:  393][251:  383][252:  381][253:  400][254:  434][255:  404]

100000 records were found in bins

results verified okay
run took 0.000511236s
```

histogram01 – Single Thread



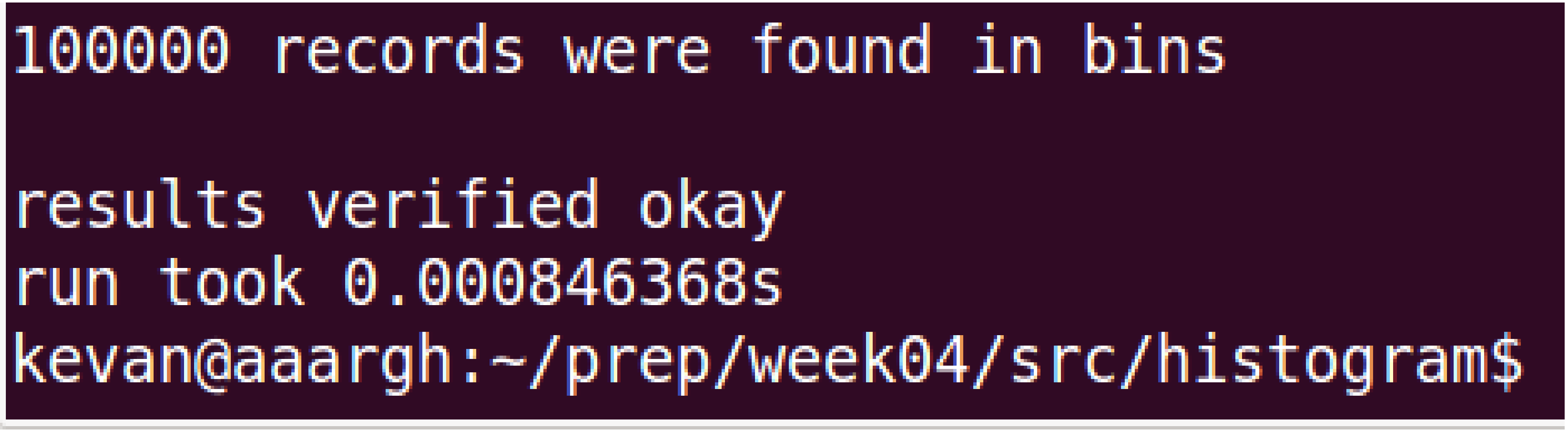
```c
typedef struct {
  int *bins;
  unsigned char  *data;
  int start_record;
  int n_records;
} param_t;

void *thread_function(param_t *params){
  int i, j;

  for(i=params->start_record;i<params->start_record + params->n_records;i++){
    j = data[i];
    params->bins[j]++;
  }
}
```

histogram01



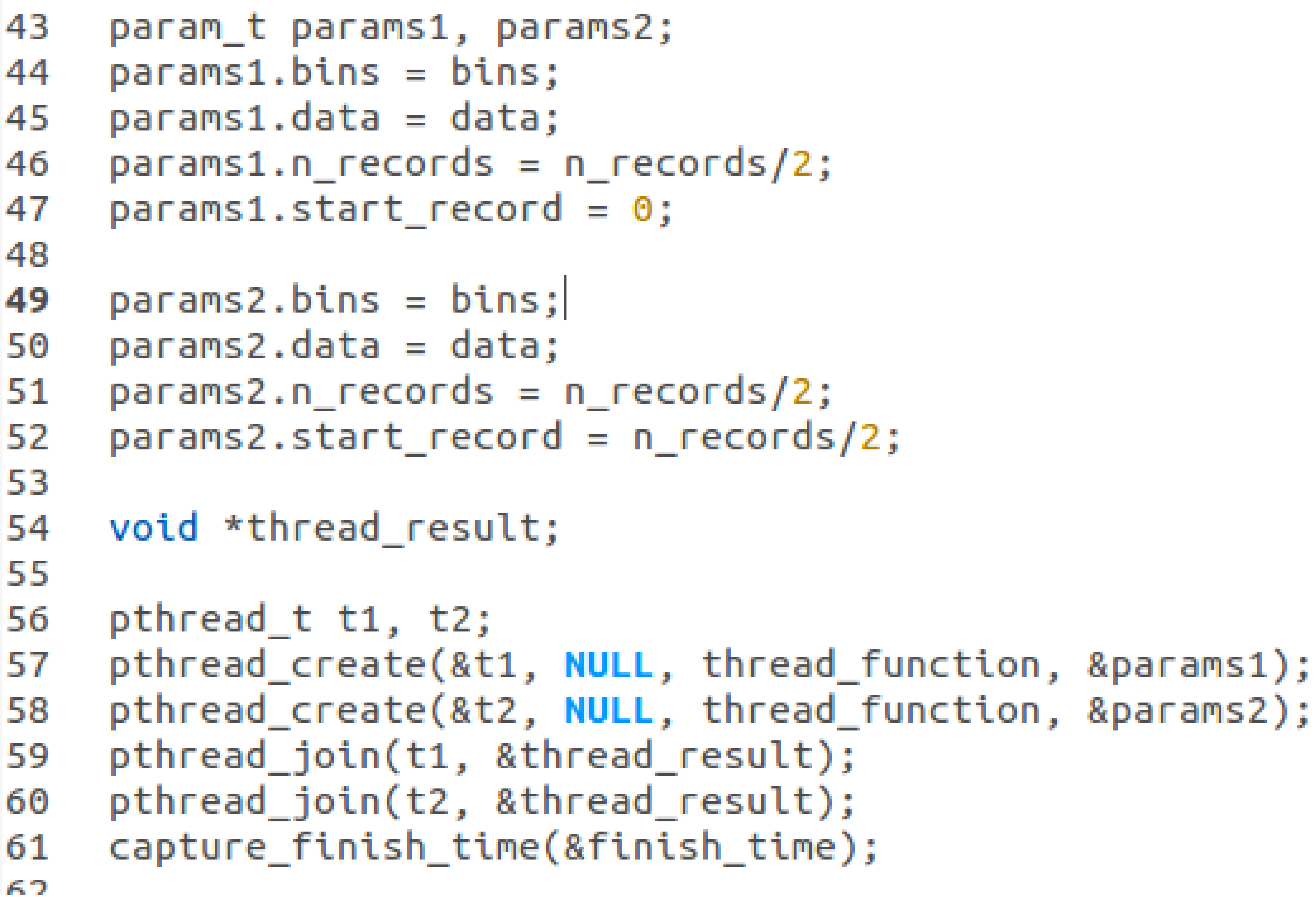
```c
int main() {
  struct timespec start_time, finish_time;
  long long int time_elapsed;

  capture_start_time(&start_time);

  int *bins = malloc(sizeof(int) * 256);
  clear_bins(bins);

  param_t params;
  params.bins = bins;
  params.data = data;
  params.n_records = n_records;
  params.start_record = 0;

  void *thread_result;
  pthread_t t;
  pthread_create(&t, NULL, thread_function, &params);
  pthread_join(t, &thread_result);
  capture_finish_time(&finish_time);

  output_results(bins);
  verify_correct_number_of_records_was_processed(bins);
  verify_results(expected_results, bins);

  if(time_difference(&start_time, &finish_time, &time_elapsed) !=0){
    fprintf(stderr,"error: start time is after finish time\n");
    return EXIT_FAILURE;
  }

  printf("run took %0.9lfs\n", (time_elapsed/1.0e9));

  return 0;
}
```

histogram01



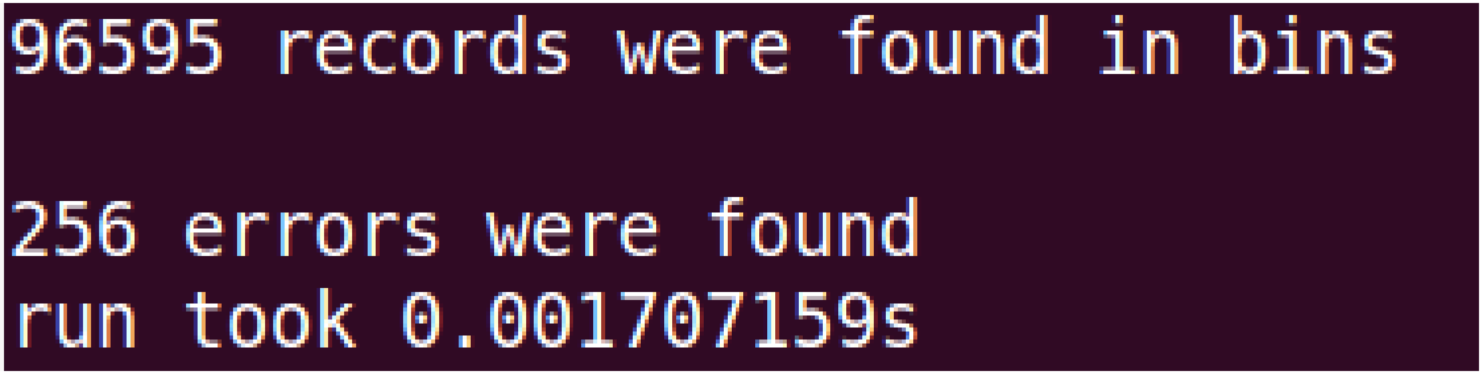
```
100000 records were found in bins

results verified okay
run took 0.000846368s
kevan@aaargh:~/prep/week04/src/histogram$
```

histogram02 – two threads



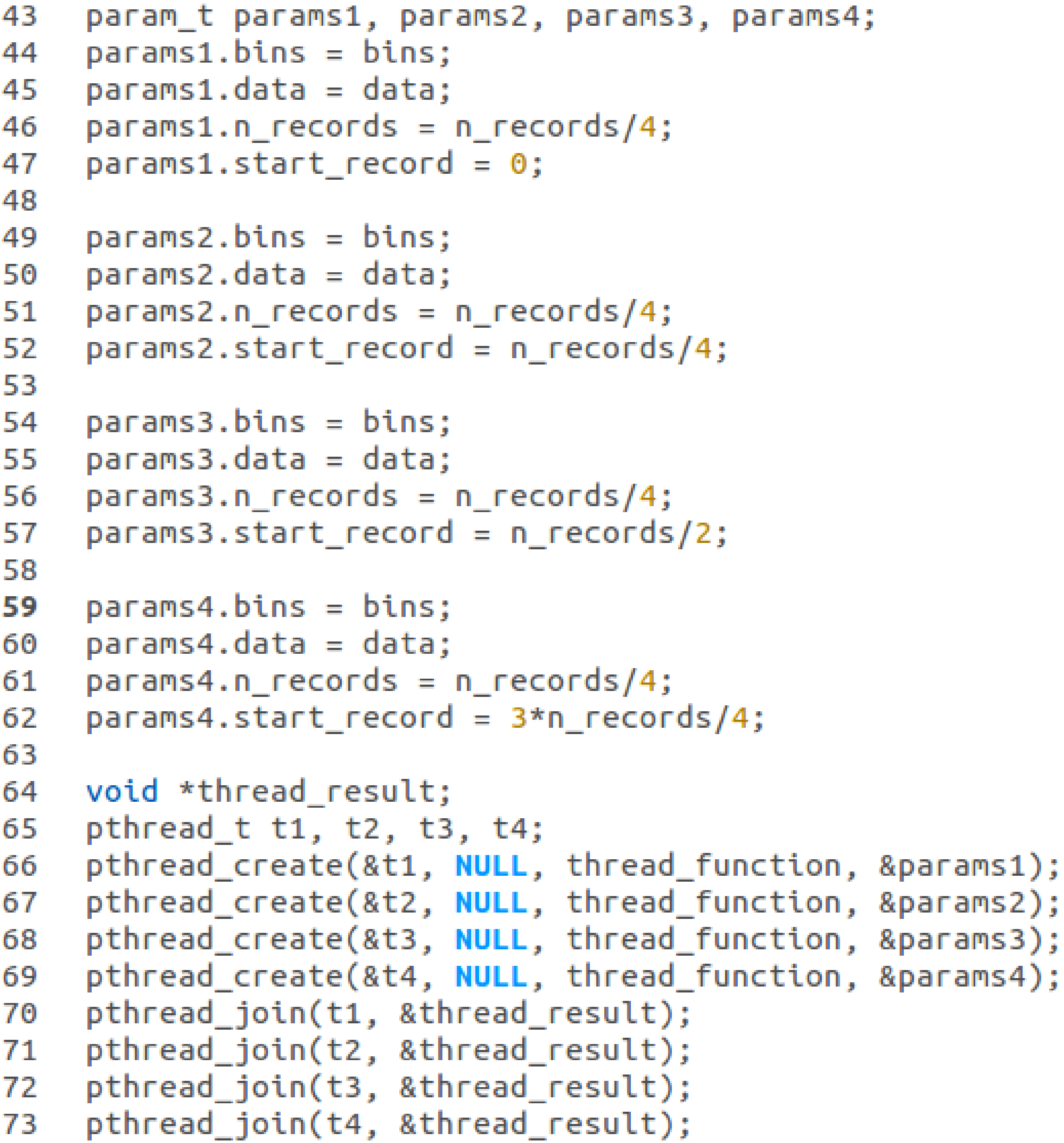
```
param_t params1, params2;
params1.bins = bins;
params1.data = data;
params1.n_records = n_records/2;
params1.start_record = 0;

params2.bins = bins;
params2.data = data;
params2.n_records = n_records/2;
params2.start_record = n_records/2;

void *thread_result;

pthread_t t1, t2;
pthread_create(&t1, NULL, thread_function, &params1);
pthread_create(&t2, NULL, thread_function, &params2);
pthread_join(t1, &thread_result);
pthread_join(t2, &thread_result);
capture_finish_time(&finish_time);
```

histogram02



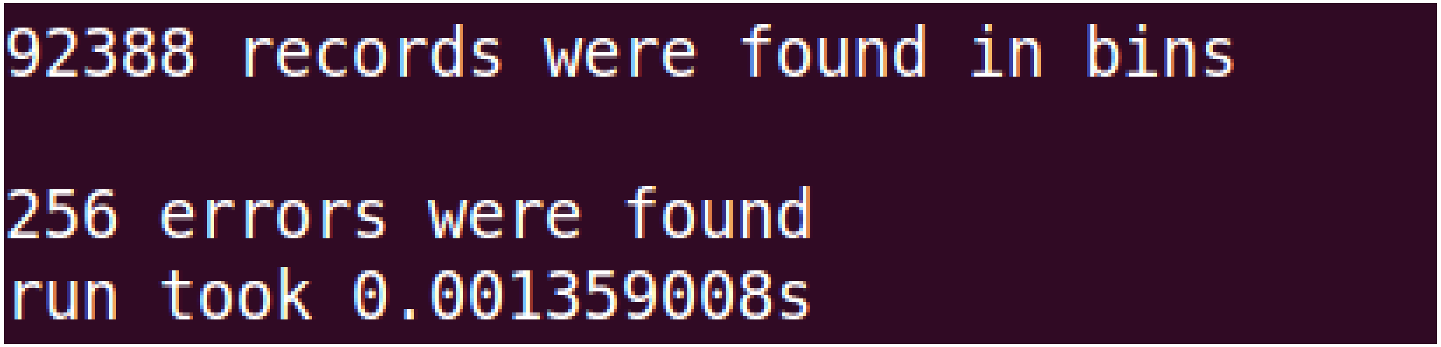
```
96595 records were found in bins

256 errors were found
run took 0.001707159s
```

histogram04 – four threads



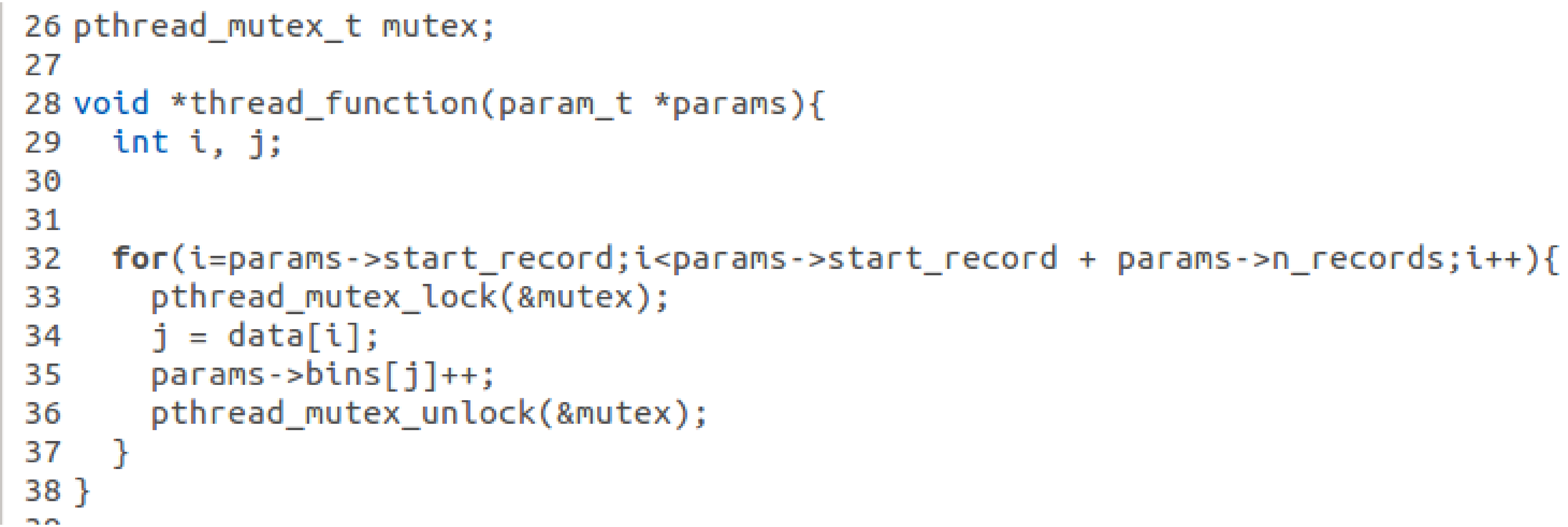
```c
param_t params1, params2, params3, params4;
params1.bins = bins;
params1.data = data;
params1.n_records = n_records/4;
params1.start_record = 0;

params2.bins = bins;
params2.data = data;
params2.n_records = n_records/4;
params2.start_record = n_records/4;

params3.bins = bins;
params3.data = data;
params3.n_records = n_records/4;
params3.start_record = n_records/2;

params4.bins = bins;
params4.data = data;
params4.n_records = n_records/4;
params4.start_record = 3*n_records/4;

void *thread_result;
pthread_t t1, t2, t3, t4;
pthread_create(&t1, NULL, thread_function, &params1);
pthread_create(&t2, NULL, thread_function, &params2);
pthread_create(&t3, NULL, thread_function, &params3);
pthread_create(&t4, NULL, thread_function, &params4);
pthread_join(t1, &thread_result);
pthread_join(t2, &thread_result);
pthread_join(t3, &thread_result);
pthread_join(t4, &thread_result);
```

histogram04



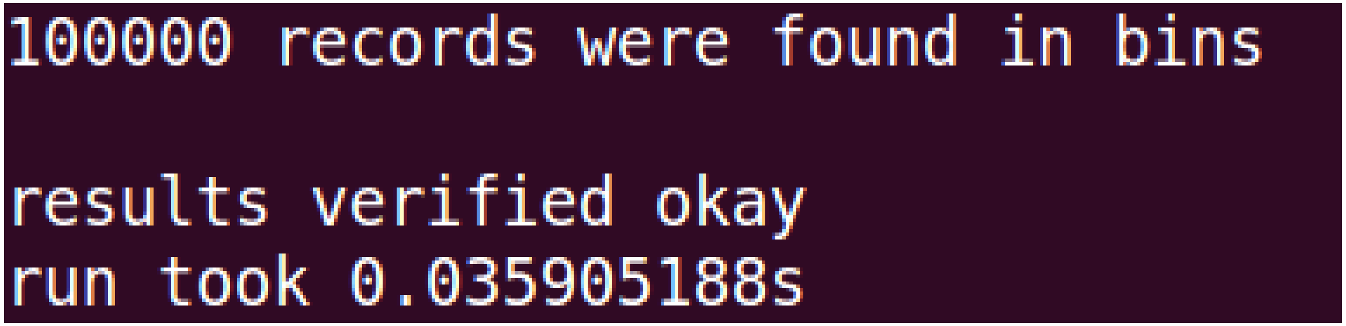
```
92388 records were found in bins

256 errors were found
run took 0.001359008s
```

histogram04b – Mutex to the Rescue



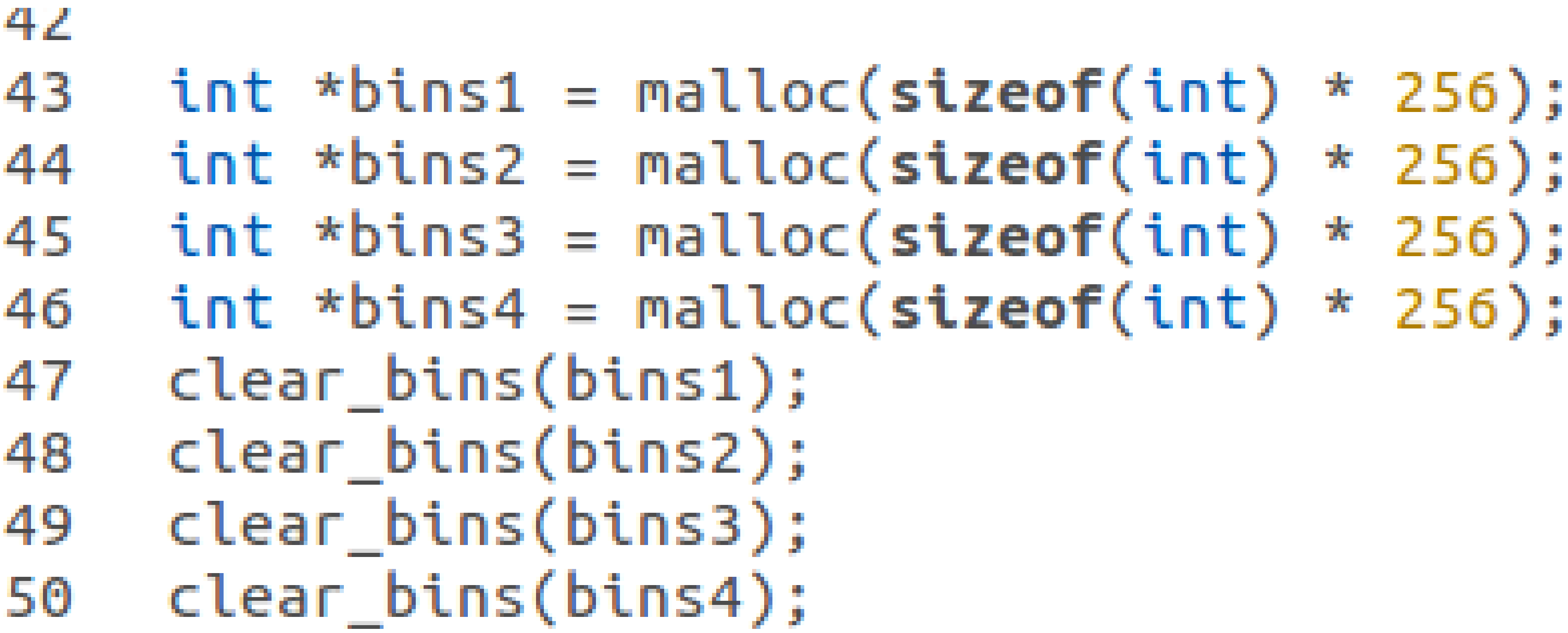
```c
pthread_mutex_t mutex;

void *thread_function(param_t *params){
  int i, j;


  for(i=params->start_record;i<params->start_record + params->n_records;i++){
    pthread_mutex_lock(&mutex);
    j = data[i];
    params->bins[j]++;
    pthread_mutex_unlock(&mutex);
  }
}
```

histogram4b



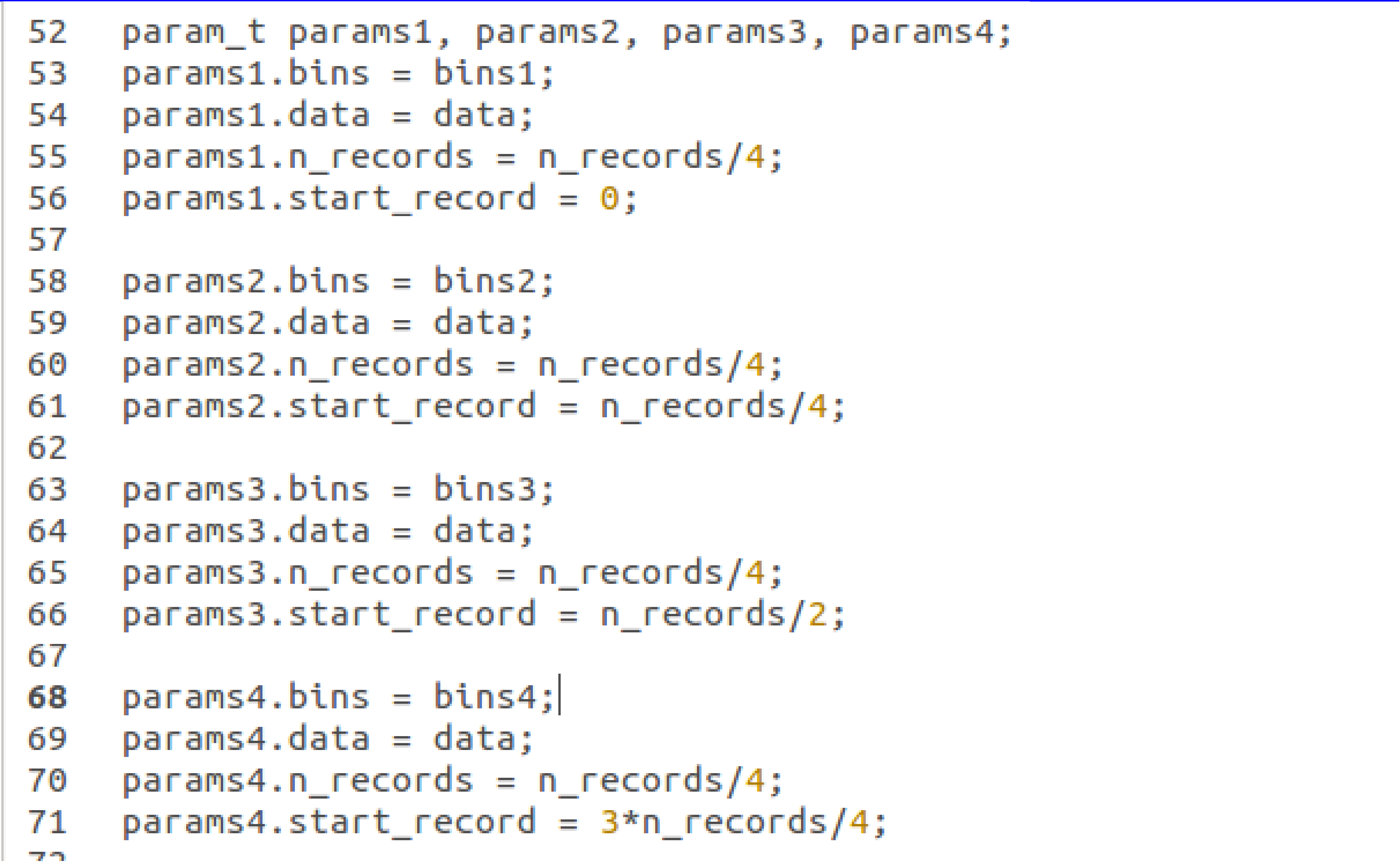
```
100000 records were found in bins

results verified okay
run took 0.035905188s
```

histogram04c – Independent Histograms



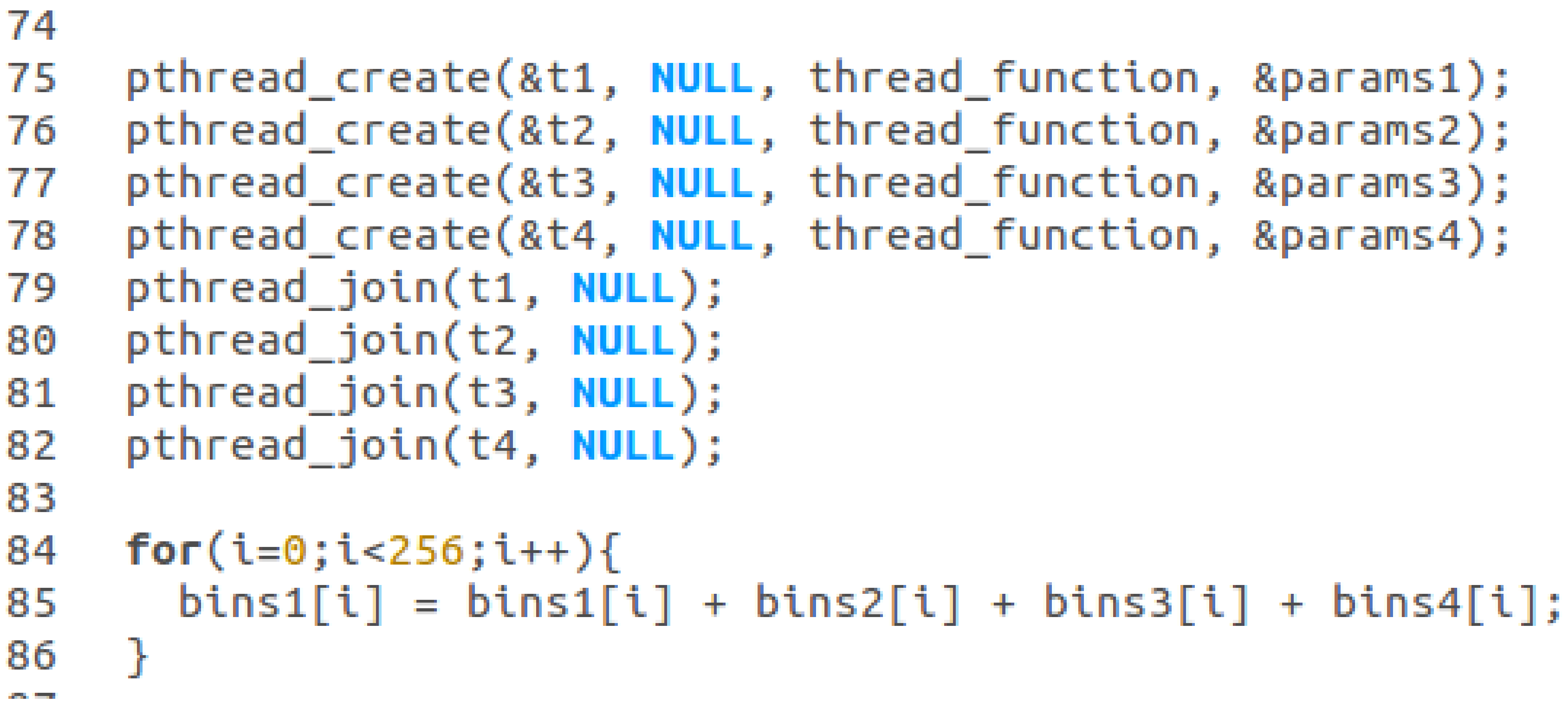
```c
int *bins1 = malloc(sizeof(int) * 256);
int *bins2 = malloc(sizeof(int) * 256);
int *bins3 = malloc(sizeof(int) * 256);
int *bins4 = malloc(sizeof(int) * 256);
clear_bins(bins1);
clear_bins(bins2);
clear_bins(bins3);
clear_bins(bins4);
```

histogram04c – Independent Histograms



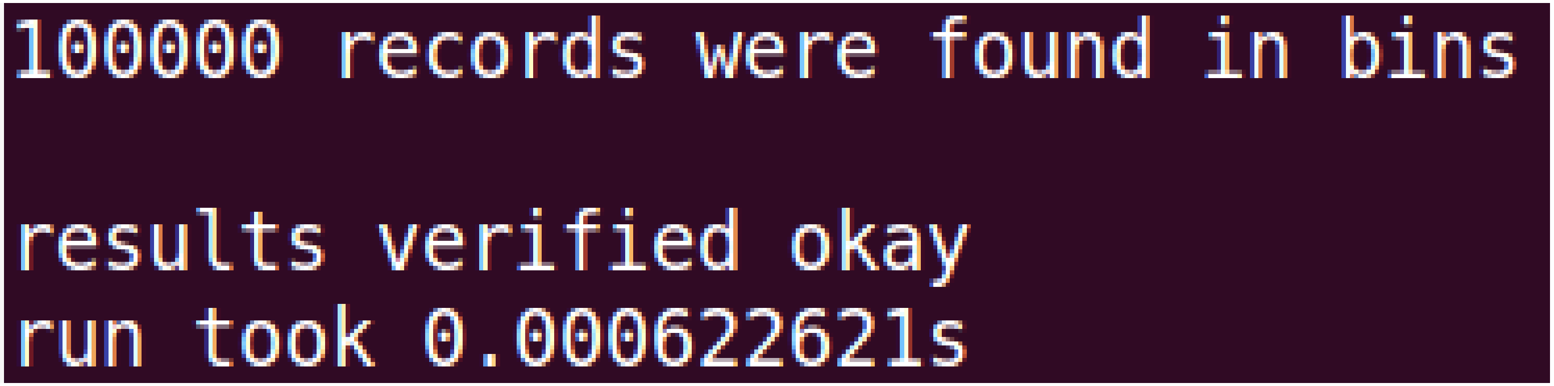
```
param_t params1, params2, params3, params4;
params1.bins = bins1;
params1.data = data;
params1.n_records = n_records/4;
params1.start_record = 0;

params2.bins = bins2;
params2.data = data;
params2.n_records = n_records/4;
params2.start_record = n_records/4;

params3.bins = bins3;
params3.data = data;
params3.n_records = n_records/4;
params3.start_record = n_records/2;

params4.bins = bins4;
params4.data = data;
params4.n_records = n_records/4;
params4.start_record = 3*n_records/4;
```

histogram04c – Independent Histograms



```
pthread_create(&t1, NULL, thread_function, &params1);
pthread_create(&t2, NULL, thread_function, &params2);
pthread_create(&t3, NULL, thread_function, &params3);
pthread_create(&t4, NULL, thread_function, &params4);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
pthread_join(t3, NULL);
pthread_join(t4, NULL);

for(i=0;i<256;i++){
  bins1[i] = bins1[i] + bins2[i] + bins3[i] + bins4[i];
}
```

histogram04c



```
100000 records were found in bins

results verified okay
run took 0.000622621s
```

Summary

- We have learnt about race conditions and interference.
  - Multiple threads working on the same data can produce unpredictable results.
- Mutual exclusion (mutex) can protect a critical section.
  - Good – it can avoid data corruption
  - Bad – creates bottlenecks
- Clever algorithm design can avoid mutexes

Next

- Today we saw issues with multithreading using just 4 threads
- Next week we will see what happens when we try to use thousands of threads to create a histogram of the number of occurrences in “War and Peace”
  - When there is a critical section accessible only by one thread, there can be serious consequences is 1 thread holds back 1000s of others.