Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

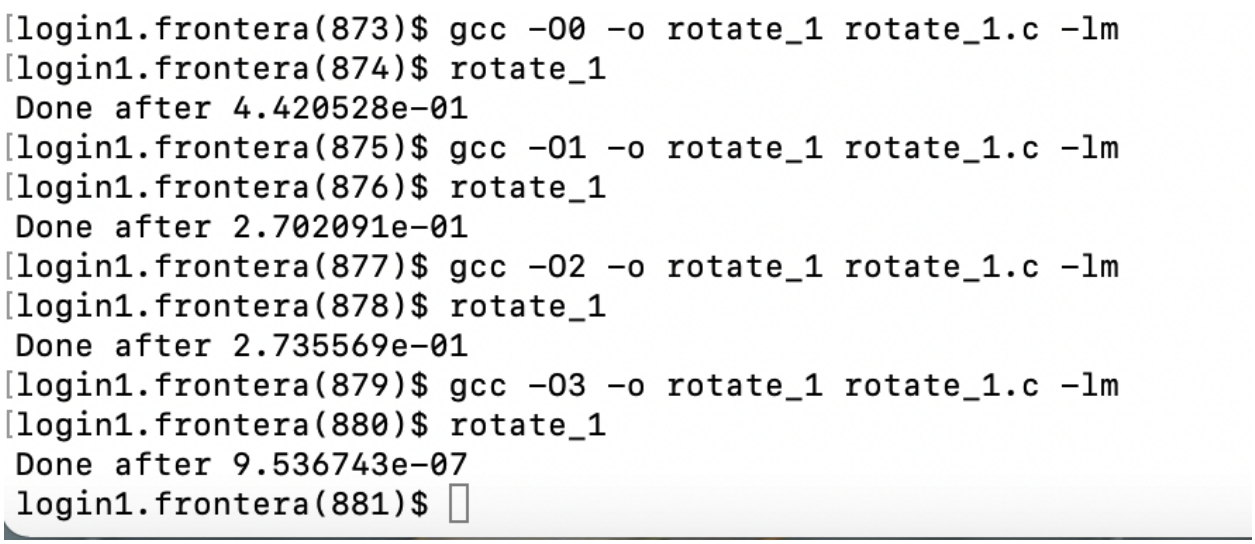
Homework #3

The screenshot below shows the source file “rotate.c” in the first iteration (with no transformations.

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

void rotate(double *x,double *y,double alpha) {
  double x0 = *x, y0 = *y;
  *x = cos(alpha) * x0 - sin(alpha) * y0;
  *y = sin(alpha) * x0 + cos(alpha) * y0;
  return;
}

#include <sys/time.h>

double mysecond()
{
  struct timeval tp;
  //struct timezone tzp;
  int i;

  i = gettimeofday(&tp,NULL /* &tzp */);
  return ( (double) tp.tv_sec + (double) tp.tv_usec * 1.e-6 );
}

#define NREPS 10000000

int main() {

  double x=.5, y=.5, alpha=1.57;
  double start = mysecond();
  for (int i=0; i<NREPS; i++)
    rotate(&x,&y,alpha);
  double duration = mysecond()-start;
  printf("Done after %e\n",duration);

  return 0;
}
```

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

The following screenshot presents the compilations and associated runtimes of the source file “rotate.c” in decreasing order of runtime (increasing efficiency) under the different compilations. We can see that the “gcc -03” compiler improves the performance by a magnitude of 10^6 relative to the “gcc -00” compiler, namely, an improvement from 4.42 * e^-1 to 9.54 * e^-7.



The screenshot below presents the source file “rotate.c” with transformations. In particular, variables “cos_alpha” and “sin_alpha” were defined in the “rotate()” function definition as a substitute for explicit function calls for “sin()” and “cos()” in the assignment for “*x” and “*y” as was provided in the original iteration of the source file. This is determined to be good programming practice, and will generally improve the performance of code.

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

```
File Edit Options Buffers Tools C Help
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

void rotate(double *x,double *y,double alpha) {
  double x0 = *x, y0 = *y;
  double cos_alpha = cos(alpha), sin_alpha = sin(alpha);
  *x = cos_alpha * x0 - sin_alpha * y0;
  *y = sin_alpha * x0 + cos_alpha * y0;
  return;
}

#include <sys/time.h>

double mysecond()
{
  struct timeval tp;
  //struct timezone tzp;
  int i;

  i = gettimeofday(&tp,NULL /* &tzp */);
  return ( (double) tp.tv_sec + (double) tp.tv_usec * 1.e-6 );
}

#define NREPS 10000000

int main() {

  double x=.5, y=.5, alpha=1.57;
  double start = mysecond();
  for (int i=0; i<NREPS; i++)
    rotate(&x,&y,alpha);
  double duration = mysecond()-start;
  printf("Done after %e\n",duration);

  return 0;
}
```

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

The following screenshot presents the compilations and associated runtimes of the source file “rotate.c” in decreasing order of runtime (increasing efficiency) after the aforementioned changes to the computations of “*x” and “*y” were applied. We can see that the “gcc -03” compiler again improves the performance by a magnitude of 10^6 relative to the “gcc -00” compiler, namely, an improvement from 2.44 * e^-1 to 9.54 * e^-7. Overall, the performance improvement for the runtime for each compiler is comparable to that of the runtime of the associated compiler for the first iteration of the code. Put differently, this transformation marginally improved the runtime for each compiler and the differences in runtime across the compiler has a similar pattern.

```
login1.frontera(881)$ gcc -O0 -o rotate_2 rotate_1.c -lm
login1.frontera(882)$ rotate_2
Done after 2.443218e-01
login1.frontera(883)$ gcc -O1 -o rotate_2 rotate_1.c -lm
login1.frontera(884)$ rotate_2
Done after 2.623241e-01
login1.frontera(885)$ gcc -O2 -o rotate_2 rotate_1.c -lm
login1.frontera(886)$ rotate_2
Done after 2.624061e-01
login1.frontera(887)$ gcc -O3 -o rotate_2 rotate_1.c -lm
login1.frontera(888)$ rotate_2
Done after 9.536743e-07
login1.frontera(889)$
```

The screenshot below presents the source file “rotate.c” with a second set of transformations. In particular, the variable declaration “double duration = mysecond() - start” was removed from the code, and the argument “duration” was replaced with “mysecond()- start” in the “printf()” function call.

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

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

void rotate(double *x,double *y,double alpha) {
  double x0 = *x, y0 = *y;
  double cos_alpha = cos(alpha), sin_alpha = sin(alpha);
  *x = cos_alpha * x0 - sin_alpha * y0;
  *y = sin_alpha * x0 + cos_alpha * y0;
  return;
}

#include <sys/time.h>

double mysecond()
{
  struct timeval tp;
  //struct timezone tzp;
  int i;

  i = gettimeofday(&tp,NULL /* &tzp */);
  return ( (double) tp.tv_sec + (double) tp.tv_usec * 1.e-6 );
}

#define NREPS 10000000

int main() {

  double x=.5, y=.5, alpha=1.57;
  double start = mysecond();
  for (int i=0; i<NREPS; i++)
    rotate(&x,&y,alpha);
  printf("Done after %e\n",mysecond()-start);

  return 0;
}
```

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout
As shown in the screenshot below, this transformation actually marginally increases the runtime of the code. It should be noted that the performance improvement of the runtime for each of the compilers is roughly the same as that for the runtime in the first and second iterations of the code.

```
[login1.frontera(889)$ gcc -O0 -o rotate_3 rotate_1.c -lm
[login1.frontera(890)$ rotate_3
 Done after 2.516639e-01
[login1.frontera(891)$ gcc -O1 -o rotate_3 rotate_1.c -lm
[login1.frontera(892)$ rotate_3
 Done after 2.713890e-01
[login1.frontera(893)$ gcc -O2 -o rotate_3 rotate_1.c -lm
[login1.frontera(894)$ rotate_3
 Done after 2.708981e-01
[login1.frontera(895)$ gcc -O3 -o rotate_3 rotate_1.c -lm
[login1.frontera(896)$ rotate_3
 Done after 9.536743e-07
 login1.frontera(897)$
```

The screenshot below presents the source file “rotate.c” with a third set of transformations. In particular, the variable declarations for “cos_alpha” and “sin_alpha” in the “rotate()” function were replaced with macro “#define” directive statements.

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

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

void rotate(double *x,double *y,double alpha) {
  double x0 = *x, y0 = *y;
  #define cos_alpha cos(alpha)
  #define sin_alpha sin(alpha)
  *x = cos_alpha * x0 - sin_alpha * y0;
  *y = sin_alpha * x0 + cos_alpha * y0;
  return;
}

#include <sys/time.h>

double mysecond()
{
  struct timeval tp;
  //struct timezone tzp;
  int i;

  i = gettimeofday(&tp,NULL /* &tzp */);
  return ( (double) tp.tv_sec + (double) tp.tv_usec * 1.e-6 );
}

#define NREPS 10000000

int main() {

  double x=.5, y=.5, alpha=1.57;
  double start = mysecond();
  for (int i=0; i<NREPS; i++)
    rotate(&x,&y,alpha);
  printf("Done after %e\n",mysecond()-start);

  return 0;
}
```

Similar to the previous transformations, this change marginally increases the runtime of the code when the first compiler is used, marginally decreases the runtime when the second and third compiler are used, and nearly replicates the runtime when the fourth compiler is used. Again, it should be noted that the performance improvement of the runtime for each of the compilers is roughly the same as that for the runtime in the first and second iterations of the code.

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

```
[login1.frontera(898)$ gcc -O0 -o rotate_4 rotate_1.c -lm
[login1.frontera(899)$ rotate_4
 Done after 4.419219e-01
[login1.frontera(900)$ gcc -O1 -o rotate_4 rotate_1.c -lm
[login1.frontera(901)$ rotate_4
 Done after 2.619650e-01
[login1.frontera(902)$ gcc -O2 -o rotate_4 rotate_1.c -lm
[login1.frontera(903)$ rotate_4
 Done after 2.622778e-01
[login1.frontera(904)$ gcc -O3 -o rotate_4 rotate_1.c -lm
[login1.frontera(905)$ rotate_4
 Done after 9.536743e-07
 login1.frontera(906)$
```