Lab 8

Demo – 3 different memory allocation strategies

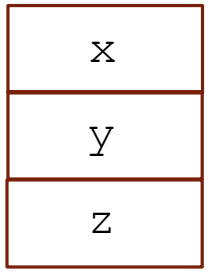
Goal – Calculate how long it takes to append an increasing number of points in `point` array, using 3 different memory allocation strategies

point_array.h

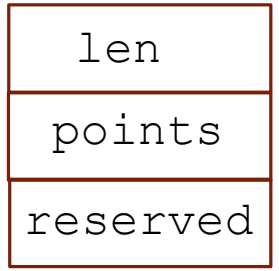
- 2 structures:

```
typedef struct point
{
  double x, y, z;
} point_t;

typedef struct
{
  // number of points in the array
  size_t len;

  // pointer to an array of point_t structs
  // There is space for 'reserved' point_t structs,
  // but 'len' point_t structs have been used so far.
  point_t* points;

  // to be discussed in class - see Demo
  size_t reserved;

} point_array_t;
```

point_t



point_array_t

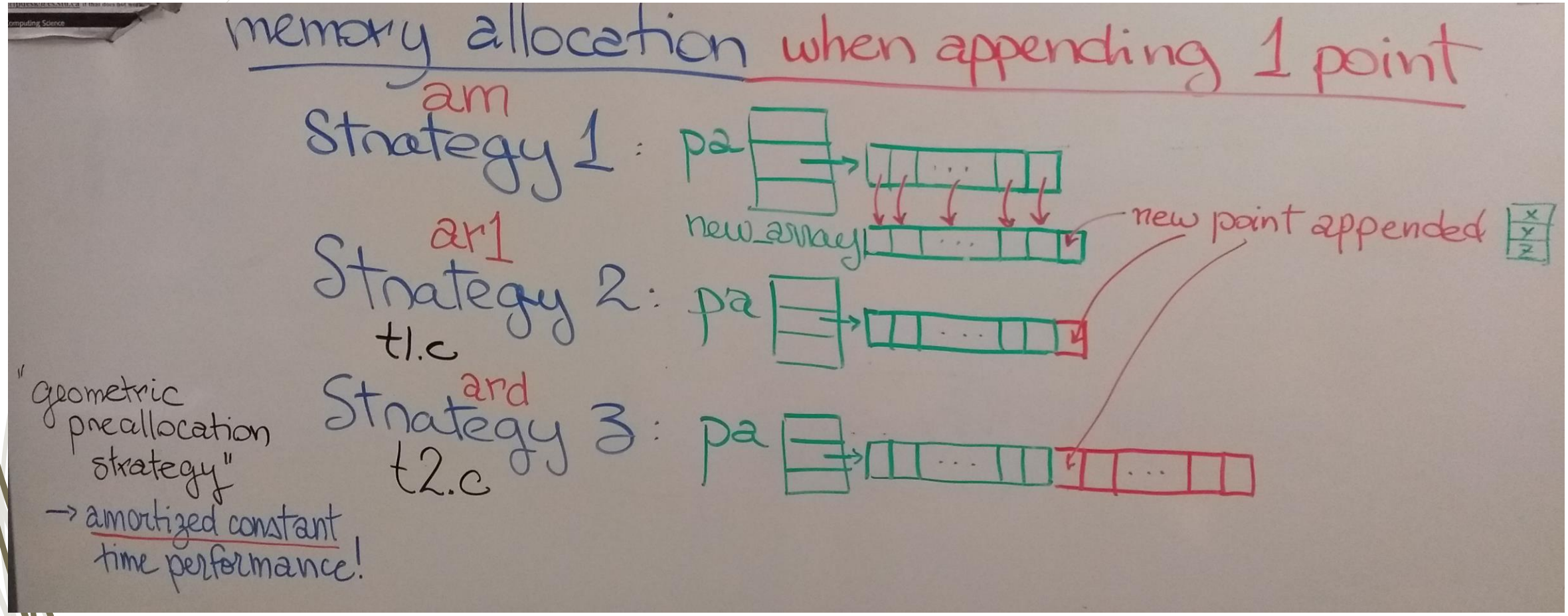


point_array.h

- 4 functions:

```
/* ALL THESE FUNCTIONS REQUIRE A VALID POINT_ARRAY_T POINTER AS THEIR
   FIRST PARAMETER. THEY SHOULD FAIL ON ASSERTION IF THIS POINTER IS
   NULL */

/* TASK 1 */

// Safely initalize an empty array structure.
void point_array_init( point_array_t* pa );

/* TASK 2 */

// Resets the array to be empty, freeing any memory allocated if
// necessary.
void point_array_reset( point_array_t* pa );

/* TASK 3 */

// Append a point to the end of an array. If successful, return 0,
// else return 1;
int point_array_append( point_array_t* pa, point_t* p );

/* TASK 4 */

// Remove the point at index i from the array, reducing the number of elements
// stored in the array by one. The order of points in the array may change.
// If successful, return 0, else return 1;
int point_array_remove( point_array_t* pa, unsigned int i );
```

3 different ways to implement `point_array_append(...)` *each using* a different memory allocation strategy



demo.c - 1

```
// returns the current system time in microseconds
uint64_t time_now_usec( void )
{
  struct timeval now;
  gettimeofday( &now, NULL );
  return (uint64_t) now.tv_sec * 1e6 + now.tv_usec;
}

int main( int argc, char** argv )
{
  printf( "#           # of PTS          APPEND\n" );
  fflush(stdout);

  point_t p;
  p.x = drand48();
  p.y = drand48();
  p.z = drand48();

  size_t pts = 16;

  for( int rep = 0; rep<12; rep++ )
  {
      point_array_t A;
      point_array_init( &A );

      uint64_t start_append = time_now_usec();

      for( unsigned long i=0; i<pts; i++ )
           point_array_append( &A, &p );

      uint64_t app_time = time_now_usec() - start_append;
```

usec is a microsecond (μs)
i.e., 0.000001 of a second
or 1.0×10^{-6} second

demo.c - 2

```
        for( int i=0; i<pts; i++ )
           point_array_remove( &A, random() % A.len );

        printf( "%d\t%10lu\t%10lu\t\n",
            rep,
            (unsigned long)pts,
            (unsigned long)app_time );

        uint64_t app_limit = 1e6;
        if( app_time > app_limit )
          {
             printf( "# appending %lu points took too long (limit %lu usec)\n",
                (unsigned long)pts, (unsigned long)app_limit );
             fflush(stdout);
             return 1;
          }

        pts *= 2;
    }
    return 0;
}
```

Makefile

```
all: am ar1 ard

am: demo.c array_malloc.c
    gcc -Wall -std=c99 -o $@ demo.c array_malloc.c

ar1: demo.c array_realloc.c
    gcc -Wall -std=c99 -o $@ demo.c array_realloc.c

ard: demo.c array_reserve.c
    gcc -Wall -std=c99 -o $@ demo.c array_reserve.c

clean:
        rm -f am ar1 ard *.o
```

At the command line:

```
$ make am
$ make ar1
$ make ard

OR

$ make
```

am -> executable with new array allocated using malloc()
ar1 -> executable with new array allocated using realloc()
and **new array size = old array size + 1**
ard -> executable with new array allocated using realloc()
and **new array size = double the old array size**
$@ -> a variable – replaced by the target

Result - am

```
username$ ./am
#       # of PTS     APPEND
0             16          9
1             32         11
2             64         31
3            128        110
4            256        283
5            512       1036
6           1024       4004
7           2048      15213
8           4096      50941
9           8192     144868
10         16384     668472
11         32768    2874128
# appending 32768 points look too long (limit 1000000 usec)
```

Result – `ar1`

```
username$ ./ar1
#       # of PTS      APPEND
0            16           3
1            32           3
2            64           5
3           128          15
4           256          25
5           512          48
6          1024          76
7          2048         138
8          4096         263
9          8192         484
10        16384        1088
11        32768        2229
```

Result – `ard`

```
username$ ./ard
#       # of PTS      APPEND
0             16           3
1             32           2
2             64           4
3            128           7
4            256           8
5            512          15
6           1024          26
7           2048          73
8           4096         118
9           8192         314
10         16384         460
11         32768         857
```