The exam was a 3-hour, closed-book, closed-notes exam.

Question 1A

Public interface methods/operations/functions, private/hidden representation

Question 1B

Note: There was a little bug in the "conj" example. It does not make sense to have:

\[ C_{\text{add}}(C_{\text{Re}}(...)) \]

because \( C_{\text{Re}}() \) should return a real number, but \( C_{\text{add}}() \) should take a complex number. So it should be something like:

\[ C_{\text{add}}(C_{\text{r2c}}(C_{\text{Re}}(...))) \]

```c
#ifndef C_INCLUDED
#define C_INCLUDED
/* A complex number. */
typedef struct Complex *C_T;
/* Return a complex number whose real part is x and imaginary part 0. */
C_T C_r2C(double x);
/* The square root of minus one. */
extern C_T C_i;
/* Return the real part of a complex number x. */
double C_Re(C_T x);
/* Return the (complex) sum of complex numbers x and y. */
C_T C_add(C_T x, C_T y);
/* Return the (complex) product of complex numbers x and y. */
C_T C_mul(C_T x, C_T y);
#endif
```

Question 1C

```c
#include <stdlib.h>
#include "complex.h"
/* Cartesian representation of a complex number. */
struct Complex {
    /* Real part. */
    double r;
    /* Imaginary part. */
    double i;
};
/* Create and return a C_T object whose real part is r and whose imaginary part is i. */
static C_T C_make(double r, double i) {
    ...
C_T p = (C_T)malloc(sizeof(struct complex));
if (p == NULL) {
    fprintf(stderr, "Could not allocate a complex number\n");
    exit(EXIT_FAILURE);
} 
p->r = r;
p->i = i;
return p;

/* The following interface functions are documented in the header file */
C_T C_r2C(double x) {
    return C_make(x, 0.0);
}

/* The square root of minus one. */
static struct complex i0 = {0.0, 1.0};
C_T C_i = &i0;

double C_Re (C_T x) {
    assert(x);
    return x->r;
}

C_T C_add(C_T x, C_T y) {
    assert(x);
    assert(y);
    return C_make(x->r + y->r, x->i + y->i);
}

C_T C_mul(C_T x, C_T y)
    assert(x);
    assert(y);
    return C_make((x->r * y->r) - (x->i * y->i), (x->r * y->i) + (x->i * y->r));

Question 2A

.globl g
  g:
  subq $16, %rsp
  movq $5, (%rsp)
  movq $6, 8(%rsp)
  movq $7, (%rsp, %rdi, 8)
  movq %rsi, %rdi
  call f
  addq (%rsp), %rax
  addq $16, %rsp
  ret

Question 2B

Flattened C:

long f(long i) {
    long a[3];
    long j;
    j=0;
L1:
    if (!(j<3)) goto L2;
    a[j] = j+j;
    j++;
    goto L1;
L2:
    return a[i];
}
Idiomatic C:

long f(long i) {
    long a[3];
    long j;
    for (j=0; j<3; j++)
        a[j] = j+j;
    return a[i];
}

Question 2C

<table>
<thead>
<tr>
<th>Description</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>array[0]</td>
<td>0</td>
</tr>
<tr>
<td>array[1]</td>
<td>2</td>
</tr>
<tr>
<td>array[2]</td>
<td>4</td>
</tr>
<tr>
<td>return address in g()</td>
<td>saved %rip</td>
</tr>
<tr>
<td>c[0]</td>
<td>7</td>
</tr>
<tr>
<td>c[1]</td>
<td>6</td>
</tr>
<tr>
<td>return address in main()</td>
<td>saved %rip</td>
</tr>
</tbody>
</table>

Question 2D

9

Question 2E

14

Question 3A

1_Minimal:  0
2_Pad:  0
X_LnC:  16
3_List:  16
4_DLL:  32
Y_DLL_o:  16
5_Bins:  32
Z_Segregated_data:  0

Question 3B

1_Minimal:  Y
2_Pad:  Y
X_LnC:  X
3_List:  W
4_DLL:  W
Y_DLL_o:  W
5_Bins:  W
Z_Segregated_data:  X

Question 3C

1_Minimal:  Z
2_Pad:  X
X_LnC:  Y
3_List:  Y
4_DLL:  Y
Y_DLL_o:  Y
Alternative answer (full credit): All answers "Z", must be accompanied by the explanation: If the client does only malloc(100000000) again and again, then probably all the malloc/free implementations will do a system call each time.

Question 3D

1_Minimal: W
2_Pad: W
X_LnC: W
3_List: Y
4_DLL: W
Y_DLL_o: W
5_Bins: W
Z_Segregated_data: W
6_VM: Z

Question 3E

P Q R

Question 3F

P R

The definition of "harm" includes "uses way more memory than it should."

Question 3G

For client P: 2_Pad, Z_Segregated_data
For client Q: Z_Segregated_data
For Client R: Z_Segregated_data
For Client S: Y_DLL_o

Question 4A

Program in Java

For 1 point credit: Do array-bounds checking.

For 2 points credit: Do array-bounds checking and garbage collection

For 3 points credit: Program in a guaranteed safe language such as Java, Python, ML, C#, JavaScript...
All of these languages have garbage collection, but array-bounds checking and garbage collection is not quite enough.

Question 4B

No, dude, that's a bad idea.