0 Preliminaries

A. Print your name on your exam booklet.

B. Print your netid next to your name.

C. Write and sign the honor pledge.

1 Bits (30 minutes, 20 points)

The ASCII character ‘a’ is represented by 97 decimal (01100001 binary) and ‘z’ is represented by 122 decimal (01111010 binary). Because there are only 26 different letters, they could be represented in 5 bits each, a=1 and z=26. Therefore, up to 12 characters can be packed into a single 64-bit word (with 4 bits left over).

0000 00000 00000 00000 00000 00000 00000 00000 01111 01100 01100 00101 01000

hello

Write a module word.c that implements this interface:

```c
/* this is word.h */
typedef unsigned long WERD;

/* w=W_make(s) */
/* w=W_make(s) */
/* w=W_make(s) */
/* w=W_make(s) */
/* W_make(const char *s); */

/* W_get(w,s) */
/* W_get(w,s) */
/* W_get(w,s) */
/* W_get(w,s) */
/* void W_get(WERD w, char s[13]); */

/* Remark: If w=W_make(s) succeeds, and then W_get(w,buf), */
/* then strcmp(buf,s)==0. */
```
2 Explain a program (55 minutes, 35 points)

The program at right works correctly and has no bugs that I know of.

When the program is run with different values of argv[1], it takes different amounts of time. (Shown here is “user time”, that is, CPU seconds of user-mode execution.)

Details you can ignore:
Intel Core i7-6600 @ 2.60GHz, 16GB RAM.

<table>
<thead>
<tr>
<th>argv[1]</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.18 sec.</td>
</tr>
<tr>
<td>10</td>
<td>1.62</td>
</tr>
<tr>
<td>100</td>
<td>0.21</td>
</tr>
<tr>
<td>1000</td>
<td>0.06</td>
</tr>
<tr>
<td>10000</td>
<td>0.01</td>
</tr>
<tr>
<td>100000</td>
<td>0.03</td>
</tr>
<tr>
<td>1000000</td>
<td>0.06</td>
</tr>
<tr>
<td>10000000</td>
<td>0.42</td>
</tr>
</tbody>
</table>

A. Write a function comment for the isnew function, appropriate to put in new.h.

B. Write a “user view” function comment for the main function.

C. What is taking time when $N$ is small?

D. What is taking time when $N$ is large?

E. For each of these variables, say what memory section it is stored in: $N$, table, num, b, p.

F. How many bytes of stack space does this program use? Pick the closest number from this list: $100$, $10^4$, $10^6$, $10^9$, $10^{12}$.

(Don’t explain, just write a number.)

G. What is the most probable printout of this program when $N$=100? (Don’t explain, just write a number.)

/* new.h */
int isnew(int key);

/* hash.c */
#include <stdlib.h>
#include "new.h"

struct binder {
    int key;
    struct binder *next;
};

enum { MAX = 100000000 };

int N;

struct binder *table[MAX];

int isnew(int num) {
    int b = num % N;
    struct binder *p = table[b];
    while (p != NULL) {
        if (p->key==num) return 0;
        p = p->next;
    }
    p=(struct binder *)
        malloc(sizeof (struct binder));
    p->key=num;
    p->next=table[b];
    table[b]=p;
    return 1;
}

/* main.c */
#include <stdio.h>
#include <stdlib.h>
#include "new.h"
#include "stdlib.h"

/* Function f returns a random number 0 <= r < 5479 */
int f(void) { return random() % 5479; }

enum { k = 5000000 };

int main(int argc, char *argv[]) {
    int r, s, i;
    if (argc!=2) exit(EXIT_FAILURE);
    r=sscanf(argv[1], "%d", &N);
    if (r!=1) exit(1);
    if (N<=0 || N>MAX) exit(EXIT_FAILURE);
    s=0;
    for (i=0; i<k; i++)
        s += isnew(f());
    printf ("%d\n", s);
    return 0;
}
3 Assembly language (35 minutes, 25 points)

struct binder {
    int key;
    struct binder *next;
};

struct binder *table[MAX]; /* global variable */
    /* skipping over the beginning of function isnew()*/
    /* local variables */
    int num; /* kept in register %edi */
    int b; /* kept in register %r12 */
    struct binder *p; /* kept in register %rbp */

    /* translate this */
p = table[b];
while (p != NULL) {
    if (p->key==num) return 0;
    p = p->next;
}

Suppose we translate this program to x86-64 assembly language.

A. binder, table: Define some constants useful in accessing the fields of struct binder, and write the assembly language that declares the global variable table.

B. the loop: Translate the 5-line fragment labeled “translate this” to assembly language, using the optimized pattern for local variable storage.

    Assume that upon entry to the function, registers %r12, %rbp, %rbx were saved, in that order. Also assume that this function uses NO stack-allocated local variables.
4 Processes (40 minutes, 20 points)

For your reference, excerpts from some Unix man pages are reproduced in the next few pages.

A. What does the program at right print on its standard output?

B. Does this program have a buffer-overrun vulnerability? If so, explain it briefly.

C. Explain step by step what will happen if /bin/echo does not exist?

D. What’s the simplest, most robust way to improve the program, to print an appropriate error message and take appropriate action, for the case where "/bin/echo" does not exist, or is not executable, or the process table is full?

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

char *args[3] = {
    "echo",
    "hello\ngoodbye",
    NULL};

int main(int argc, char *argv[]) {
    int p[2];
    int pid;
    FILE *f;
    char buffer[1000];
    pipe(p);
    pid = fork();
    if (pid == 0) {
        close(p[0]);
        dup2(p[1],1);
        close(p[1]);
        execvp("/bin/echo",args);
    }
    close(p[1]);
    f=fopen(p[0],"r");
    while (fgets(buffer, 1000, f) != NULL)
        printf("child: %s", buffer);
    return 0;
}
```

$ echo 'What does echo do?'
What does echo do?
$ echo "one\ntwo"
one
two