CSC 180H1 F October Midterm
2008
Duration — 120 minutes
Aids allowed: none

Student Number: |_|_|_|_|_|_|_|_|_|_|_|

Last Name: _____ **First Name:** _____

Lecture Section (circle a number): 1 2

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above,
and read the instructions below.)

This midterm consists of 9 questions on 11 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.* Comments are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.
For 3 bonus mark write your student number at the bottom of pages 2-11 of this test.

BONUS
MARKS: _____/ 3

1: _____/ 10

2: _____/ 15

3: _____/ 10

4: _____/ 10

5: _____/ 10

6: _____/ 15

7: _____/ 10

8: _____/ 10

9: _____/ 10

TOTAL: _____/100

Question 1. [10 MARKS]

Define a function `find_item` that returns the position of the first occurrence of an item in an array. If no such item were found in the array, it returns -1. `find_item` has three parameters: `a` is array, `n` is array size, and `item` is the item to find. For example, given the following array,

```
int a[] = {0, 1, 1, 8, 0, 1, 0, 1, 8, 0};
```

`find_item(a, 10, 8)` returns 3.

```
/* return index of item in a, -1 if none */
int find_item(int a[], int n, int item)
{
  int i;

  for (i = 0; i < n; i++)
    if (a[i] == item)
      return i;
  return -1;
}
```

Question 2. [15 MARKS]

Define a function `find_pattern` that returns the position of the first occurrence of a pattern in an array. If no such pattern were found in the array, it returns -1. `find_pattern` has four parameters: `a` is array, `n` is array size, `pattern` is pattern array, and `m` is size of pattern array. For example, given the following two arrays,

```
int a[] = {0, 1, 1, 8, 0, 1, 0, 1, 8, 0};
int b[] = {1, 8, 0};  /* pattern array */
```

`find_pattern(a, 10, b, 3)` returns 2, the position of the first digit of pattern `1 8 0` in array `a`.

```
a: 0 1 1 8 0 1 0 1 8 0
       ^
       |
       position: 2
```

```
/* return index of pattern in a, -1 if none */
int find_pattern(int a[], int n, int pattern[], int m)
{
  int i, j, k;

  for (i = 0; i < n - m + 1; i++) {
    for (j = i, k = 0; k < m && a[j] == pattern[k]; k++, j++)
      ;
    if (k > 0 && k == m)
      return i;
  }
  return -1;
}
```

Question 3. [10 MARKS]

Write a program that prints the number of non-whitespace characters read from standard input. For simplicity, consider all characters other than *space*, *tab*, and *newline* as non-whitespace characters. You must use and only use the library function `getchar` to read from standard input.

`getchar` has the following function prototype:

```
int getchar(void);
```

`getchar` reads a character at a time from standard input and returns the character as an `int`, or returns `EOF` on end of standard input or error. The symbolic constant `EOF` is an integer defined in `<stdio.h>`, whose value is typically -1.

```
#include <stdio.h>  /* for getchar */

/* count number of non-whitespace characters */
int main()
{
  int c, n;

  n = 0;
  while ((c = getchar()) != EOF)
    if (c != '\n' && c != ' ' && c != '\t')
      n++;
  printf("%d\n", n);

  return 0;
}
```

Question 4. [10 MARKS]

A binary value, i.e., a sequence of 1's and 0's, can be converted to a decimal value using the following formula:

$$\mathbf{d_n d_{n-1} \cdots d_1 d_0} = \sum_{\mathbf{i=0}}^{\mathbf{n}} \mathbf{d_i} \cdot \mathbf{2^i},$$

where $\mathbf{d_i} \in \{\mathbf{0}, \mathbf{1}\}$, $\mathbf{0} \leq \mathbf{i} \leq \mathbf{n}$. For example, $1001 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9$.

Define a function `bin2dec` that returns a decimal value given a binary value. The binary value is passed to `bin2dec` as a string. A string can be considered as a character array, ended with a null character, `'\0'`. `'\0'` marks the end of the string. For example, the string `"1011"` consists of five characters, `'1'`, `'0'`, `'1'`, `'1'`, and `'\0'`.

`bin2dec` has one parameter, `a`, which is a pointer to the string to be converted. For example, `bin2dec("1001")` returns 9.

You cannot call any other function in `bin2dec`.

```
#include <stdio.h>

int bin2dec(char a[])
{
  int val, i;

  val = 0;
  for (i = 0; a[i] != '\0'; i++)
    val = 2 * val + (a[i] - '0');

  return val;
}

int main()
{
  printf("%d\n", bin2dec("1001"));  /* print 9 */
  return 0;
}
```

Question 5. [10 MARKS]

Complete the two functions `dna2aa` and `protein` below.

`dna2aa` returns an amino acid given a DNA codon, which consists of three capital letters. `dna2aa` has three parameters: `a` is the first letter of a DNA codon, `b` the second, and `c` the third. A partial mapping from DNA codons to amino acids is shown in the following table:

DNA codon	amino acid
CTT, CTG	L
GAA	E
TGT	C
TAA, TAG, TGA	*

If a DNA codon does not appear in the above table, `dna2aa` returns `'?'`.

Note the many-to-one mapping for amino acids `L` and `*`: e.g., given either `CTT` or `CTG`, `dna2aa` returns `'L'`.

`protein` prints a sequence of n amino acids given a sequence of n DNA codons. See the `main` function below for example. Assume that the length of a sequence of DNA codons is always a multiple of three.

```
#include <stdio.h>

char dna2aa(char a, char b, char c)
{
  if (a == 'G' && b == 'A' && c == 'A')
    return 'E';
  if (a == 'C' && b == 'T' && c == 'T'
      || a == 'C' && b == 'T' && c == 'G')
    return 'L';
  if (a == 'T' && b == 'G' && c == 'T')
    return 'C';
  if (a == 'T' && b == 'A' && c == 'A'
      || a == 'T' && b == 'A' && c == 'G'
      || a == 'T' && b == 'G' && c == 'A')
    return '*';
  return '?';
}

void protein(char a[])
{
  int i;

  for (i = 0; a[i] != '\0'; i += 3)
    printf("%c", dna2aa(a[i], a[i+1], a[i+2]));
  printf("\n");
}

int main()
{
  char dna[] = "TGTGAACTGCTTAAATAG";
  protein(dna);  /* print CELL?* */
  return 0;
}
```

Question 6. [15 MARKS]

Rewrite the function `reverse` into a version called `reverse2`, using pointers instead. In `reverse2`, you must use and only use the variables that are declared for you. Failure to do so will result in considerable mark deductions.

`strlen` returns the length of the string `s`, not including the terminating null character, `'\0'`. For example, `strlen("abc")` returns 3.

```
#include <stdio.h>
#include <string.h>  /* for strlen */

/* reverse string s in place, array version */
void reverse(char s[])
{
  int c, i, j;

  for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
    c = s[i];
    s[i] = s[j];
    s[j] = c;
  }
}

/* reverse string s in place, pointer version */
void reverse2(char *s)
{
  int c;
  char *p, *q;

  for (p = s, q = s + strlen(s) - 1; p < q; p++, q--) {
    c = *p;
    *p = *q;
    *q = c;
  }
}

int main()
{
  char a[] = "expression";

  reverse(a);
  printf("%s\n", a); /* print noisserpxe */

  reverse2(a);
  printf("%s\n", a); /* print expression */

  return 0;
}
```

Question 7. [10 MARKS]

Consider the following program. Write the output after each `printf` statement inside the accompanying comment. You don't need to write *newline* characters.

```
/* scope */
#include <stdio.h>

int a;
int b = -10;

int f(int i)
{
  b += i;
  return b;
}

void g(int a)
{
  a = 0;
}

void h(void)
{
  a = 0;
}

int main()
{
  int i = -1;

  a = 8;
  {
    int i, b;

    for (i = 0; i < a && (b = f(i)) <= 0; i++)
      printf("%d ", b);   /* Output: -10 -9 -7 -4 0 */
    printf("\n%d\n", i);  /* Output: 5 */
  }
  printf("%d\n", i);      /* Output: -1 */

  g(a);
  printf("%d\n", a);      /* Output: 8 */

  h();
  printf("%d\n", a);      /* Output: 0 */

  return 0;
}
```

Question 8. [10 MARKS]

Consider the following recursive function `mystery`.

Assume that `"0123456789ABCDEF"[i]` gives the character at position `i` in the string of `"0123456789ABCDEF"`. For example, `"0123456789ABCDEF"[0]` is `'0'`, and `"0123456789ABCDEF"[10]` is `'A'`.

```
#include <stdio.h>

void mystery(int n, int b)
{
  char c;

  if (n <= 0 || (b < 2 || b > 16))
    return;

  mystery(n/b, b);
  c = "0123456789ABCDEF"[n % b];
  printf("%c", c);
}
```

(a) [2 marks] What is the output of `mystery(26, 16)`? Answer: 1A

(b) [2 marks] What is the output of `mystery(26, 2))`? Answer: 11010

(c) [2 marks] Approximately how many recursive calls are made when calling `mystery(n, 2)` for some large positive integer `n`? Answer: $\lfloor \log_2 n \rfloor + 2$

(d) [2 marks] What does `mystery` do? Explain it in one sentence.

Answer: Convert a positive decimal integer to a base-b number, where b is an integer between 2 and 16.

(e) [2 marks] What is the output of `mystery2(26, 2)`, given the definition of `mystery2` as follows?

Answer: 01011

```
#include <stdio.h>

void mystery2(int n, int b)
{
  char c;

  if (n <= 0 || (b < 2 || b > 16))
    return;

  c = "0123456789ABCDEF"[n % b];
  printf("%c", c);
  mystery2(n/b, b);
}
```

Question 9. [10 MARKS]

(a) [4 marks] Complete the following two functions `push` and `pop`.

```
/* character stack implementation using pointers */
#include <stdio.h>

#define MAXVAL  100  /* maximum depth of val stack */

char val[MAXVAL];    /* stack */
char *p = val;       /* next free stack position */

void push(char a)
{
  if (p - val < MAXVAL)
    *p++ = a;
  else
    printf("error: stack full, cannot push %c\n", a);
}

char pop(void)
{
  if (p > val)
    return *--p;
  else {
    printf("error: stack empty\n");
    return '\0';
  }
}

int main()
{
  int i;

  push('K');
  push('C');
  push('A');
  push('T');
  push('S');

  for (i = 0; i < 5; i++)
    printf("%c", pop());    /* Output: STACK */
  printf("\n");

  return 0;
}
```

(b) [6 marks] Implement a string stack.

```
#include <stdio.h>

void push(char *a);
char *pop(void);
```

```
int main()
{
  int i;

  push("stack");
  push("string");
  push("A");

  for (i = 0; i < 3; i++)
    printf("%s ", pop());  /* Output: A string stack  */
  printf("\n");

  return 0;
}

/* write string stack implementation below */

#define MAXVAL  100  /* maximum depth of val stack */

char *val[MAXVAL];   /* stack */
char **p = val;      /* next free stack position */

void push(char *a)
{
  if (p - val < MAXVAL)
    *p++ = a;
  else
    printf("error: stack full, cannot push %s\n", a);
}

char *pop(void)
{
  if (p > val)
    return *--p;
  else {
    printf("error: stack empty\n");
    return "";
  }
}
```

Total Marks = 100