ECE264 Summer 2013
Exam 2, July 3, 2013

In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action.

Name:

Signature:

You must sign here. Otherwise you will receive a **2-point** *penalty.*

Read the questions carefully.
Some questions have conditions and restrictions.

This is an *open-book, open-note* exam. You may use any book, notes, or program printouts. No personal electronic device is allowed. You may not borrow books from other students.

Two learning objectives (recursion and structure) are tested in this exam. To pass an objective, you must receive 50% or more points in the questions in this exam.
If you have already passed a learning objective earlier, you will never "unpass" the learning objective later.

Contents

Learning Objective 1 (Recursion)	Pass	Fail
Learning Objective 3 (Structure)	Pass	Fail

Total Score:

1 Recursive Formula (5 points). Learning Objective Recursion

For a given positive integer, we want to partition it into the sum of some positive integers, or itself. For example, 1 to 5 can be partitioned as

```
1 = 1        2 = 1 + 1        3 = 1 + 1 + 1       4 = 1 + 1 + 1 + 1
               = 2              = 1 + 2               = 1 + 1 + 2
                                = 2 + 1               = 1 + 2 + 1
                                = 3                   = 1 + 3
                                                      = 2 + 1 + 1
                                                      = 2 + 2
                                                      = 3 + 1
                                                      = 4

5 = 1 + 1 + 1 + 1 + 1
    1 + 1 + 1 + 2
    1 + 1 + 2 + 1
    1 + 1 + 3
    1 + 2 + 1 + 1
    1 + 2 + 2
    1 + 3 + 1
    1 + 4
    2 + 1 + 1 + 1
    2 + 1 + 2
    2 + 2 + 1
    2 + 3
    3 + 1 + 1
    3 + 2
    4 + 1
    5
```

We observe that there exists

- One way to partition 1.
- Two ways to partition 2.
- Four ways to partition 3.
- Eight ways to partition 4.

In general, there are 2^{n-1} ways to partition value n. You do not have to prove this.

Also by observation, we find

- One way to partition 1 using only odd numbers, i.e., itself

- One way to partition 2 using only odd numbers, i.e., 1 + 1. The value 2 cannot be used because it is an even number.
- Two ways to partition 3 using only odd numbers, i.e., 1 + 1 + 1 and 3 itself.
- Three ways to partition 4 using only odd numbers, i.e.,
  1 + 1 + 1 + 1,
  1 + 3,
  and 3 + 1.
- Five ways to partition 5 using only odd numbers, i.e.,
  1 + 1 + 1 + 1 + 1,
  1 + 1 + 3,
  1 + 3 + 1,
  3 + 1 + 1,
  and 5.

Q1: How many ways can you partition 8 using only odd numbers (1 point)?
Q2: How many ways can you partition value n ($n > 2$) using only odd numbers? Write down the general rule (4 points). You can write down a recursive form. You do not need to write the closed form.
Hint: You may want to solve Q2 before answering Q1.
Consider whether n is an odd number or an even number.
Do not intend to draw a pattern from f(1) = 1, f(2) = 1, f(3) = 2, f(4) = 3, and f(5) = 5. You should observe the pattern for a general value n.
You **must** explain your answer. *An answer without explanation will receive no point.*
This question asks you to write a mathematical formula, not a C program.

2 Integer Partition (5 points). Learning Objective Recursion

Consider the following program that partitions a positive integer.
By observation,

- Only one number is used to partition 1, i.e., itself.
- Three numbers are used to partition 2, i.e., 1 + 1 (two numbers) and 2 (one number).
- Eight numbers are used to partition 3, i.e., 1 + 1 + 1 (three numbers), 1 + 2 (two numbers), 2 + 1 (two numbers) and and 3 (one number).

Modify the program below so that it reports to the `main` function in line 34 how many numbers are used in generating the partitions. The program needs to print **how many** numbers are used and does **not** need to print the partitions (such as 1 + 1 + 1 ...).

Hint: Try to make as few changes to the program as possible to save your time. One solution needs to add/remove/change *no more than 10 lines of code and no more than 100 characters.* If you find youselves writing a lot of code, please consider to find a simpler solution.

The following table is for your reference.

input argument	returned value
1	1
2	3
3	8
4	20

You should modify the program to compute how many numbers are used and do **not** write a mathematical formula. You will receive no point if you write a mathematical formula.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4
 5 static void partitionHelper(int value, int * arr, int ind)
 6 {
 7   if (value == 0)
 8     {
 9       return;
10     }
11   int nextVal;
12   for (nextVal = 1; nextVal <= value; nextVal ++)
13     {
14       arr[ind] = nextVal;
```

```
      partitionHelper(value - nextVal, arr, ind + 1);
    }
}

int partition(int value)
{
  int * arr;
  arr = malloc(sizeof(int) * value);
  partitionHelper(value, arr, 0);
  free (arr);
  return 0;
}

int main(int argc, char * * argv)
{
  int val;
  int MAXLENGTH = 8;
  for (val = 1; val <= MAXLENGTH; val ++)
    {
      int total = partition(val);
      printf("%d numbers are used to partition %d\n", total, val);
    }
  return EXIT_SUCCESS;
}
```

3 Structure (4 points). Learning Objective Structure

Consider the following declaration of a structure.

```
#ifndef STUDENT_H
#define STUDENT_H
typedef struct
{
  int id;
  char * name;
} Student;
#endif
```

The following program contains four `swap` functions. The purpose is to swap the two input arguments of the `Student` structure. Please identify which of the four `swap` functions can correctly swap the attributes of the two arguments.
For each function, please answer "Yes" (correctly swap the arguments *and the changes are visible in the calloer*) or "No" (not correctly swap the arguments).
Briefly explain the reasons of your answers. *An answer without explanation will receive no point.*

- `swap1`: Yes No
  Reason:

- `swap2`: Yes No
  Reason:

- `swap3`: Yes No
  Reason:

- `swap4`: Yes No
  Reason:

The manual of `strdup` is included here.

```
SYNOPSIS
       #include <string.h>
       char *strdup(const char *s);

DESCRIPTION
       The strdup() function returns a pointer to a  new  string  which  is  a
       duplicate  of the string s.  Memory for the new string is obtained with
       malloc(3), and can be freed with free(3).
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "student.h"
void swap1(Student a, Student b)
{
  Student t = a;
  a = b;
  b = t;
}

void swap2(Student a, Student b)
{
  int id;
  char * name;
  name   = a.name;
  id     = a.id;
  a.id   = b.id;
  a.name = b.name;
  b.id   = id;
  b.name = name;
}

void swap3(Student * a, Student * b)
{
  Student * t = a;
  a = b;
  b = t;
}

void swap4(Student * a, Student * b)
{
  int id;
  char * name;
  id        = a -> id;
  name       = a -> name;
  a -> id    = b -> id;
  a -> name = b -> name;
  b -> id    = id;
  b -> name = name;
}

int main(int argc, char * * argv)
{
  char * name1 = "Purdue";
  char * name2 = "Engineering";
  Student s1;
  Student s2;

  s1.id = 264;
  s1.name = strdup(name1);
  s2.id = 2013;
  s2.name = strdup(name2);
  printf("s1: %d, %s\n", s1.id, s1.name);
  printf("s2: %d, %s\n", s2.id, s2.name);
  swap1(s1, s2);
  printf("s1: %d, %s\n", s1.id, s1.name);
  printf("s2: %d, %s\n", s2.id, s2.name);
  free(s1.name);
  free(s2.name);
  printf("================================\n");

  s1.id = 264;
  s1.name = strdup(name1);
  s2.id = 2013;
  s2.name = strdup(name2);
  printf("s1: %d, %s\n", s1.id, s1.name);
  printf("s2: %d, %s\n", s2.id, s2.name);
  swap2(s1, s2);
  printf("s1: %d, %s\n", s1.id, s1.name);
  printf("s2: %d, %s\n", s2.id, s2.name);
  free(s1.name);
  free(s2.name);
  printf("================================\n");

  s1.id = 264;
  s1.name = strdup(name1);
  s2.id = 2013;
  s2.name = strdup(name2);
  printf("s1: %d, %s\n", s1.id, s1.name);
  printf("s2: %d, %s\n", s2.id, s2.name);
  swap3(& s1, & s2);
  printf("s1: %d, %s\n", s1.id, s1.name);
  printf("s2: %d, %s\n", s2.id, s2.name);
  free(s1.name);
  free(s2.name);
  printf("================================\n");

  s1.id = 264;
  s1.name = strdup(name1);
  s2.id = 2013;
  s2.name = strdup(name2);
  printf("s1: %d, %s\n", s1.id, s1.name);
  printf("s2: %d, %s\n", s2.id, s2.name);
  swap4(& s1, &s2);
  printf("s1: %d, %s\n", s1.id, s1.name);
  printf("s2: %d, %s\n", s2.id, s2.name);
  free(s1.name);
  free(s2.name);

  return EXIT_SUCCESS;
}
```

4 Structure and Sorting (4 points). Learning Objective Structure

Consider the following declaration of a structure.

```
#ifndef STUDENT_H
#define STUDENT_H
typedef struct
{
  int id;
  char * name;
} Student;
#endif
```

The following function sorts an array of `Student` objects using the `qsort` function. The objects are sorted by the names. Please fill the missing code.
The manual of `qsort` is included here:

```
NAME
       qsort, qsort_r - sort an array

SYNOPSIS
       #include <stdlib.h>

       void qsort(void *base, size_t nmemb, size_t size,
                  int (*compar)(const void *, const void *));

DESCRIPTION
       The  qsort()  function sorts an array with nmemb elements of size size.
       The base argument points to the start of the array.

       The contents of the array are sorted in ascending order according to  a
       comparison  function  pointed  to  by  compar, which is called with two
       arguments that point to the objects being compared.

       The comparison function must return an integer less than, equal to,  or
       greater  than  zero  if  the first argument is considered to be respec
       tively less than, equal to, or greater than the second.  If two members
       compare as equal, their order in the sorted array is undefined.
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "student.h"
int compareStudent(const void * p1, const void * p2)
{
  /* compare students by their names */
  /* hint: use strcmp */
  /* fill in the code here (2 points) */










}

void sortStudent(Student * arr, int num)
{
  /* fill in the code here (1 point) */
  /* call qsort */







}

void printStudent(Student * arr, int num)
{
  int ind;
  for (ind = 0; ind < num; ind ++)
    {
      printf("Student[%d]: name = %s, id = %d\n",
             ind, arr[ind].name, arr[ind].id);
    }
}

int main(int argc, char * * argv)
```

```c
{
  int num = 10;
  int maxlen = 100;
  int maxval = 10000;
  Student * stu;
  int ind;
  stu = malloc(sizeof(Student) * num);
  for (ind = 0; ind < num; ind ++)
    {
      stu[ind].id = rand() % maxval;
      stu[ind].name = malloc(sizeof(char) * maxlen);
      sprintf(stu[ind].name, "purdue%dspring", rand() % maxval);
    }
  printStudent(stu, num);
  printf("===============================\n");
  sortStudent(stu, num);
  printStudent(stu, num);
  /* release memory */
  /* fill in code here (1 point) */







  return EXIT_SUCCESS;
}
```

5 Structure (2 points). Learning Objective Structure

Consider the following program.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  typedef struct
5  {
6    int value;
7    int * iptr;
8  } IntStruct;
9
10 void f(int n, IntStruct * istu)
11 {
12   if (n == 0)
13     {
14       return;
15     }
16   (istu -> value) ++;
17   (*(istu -> iptr)) ++;
18   f(n - 1, istu);
19 }
20
21 int main(int argc, char * * argv)
22 {
23   IntStruct * it;
24   it = malloc(sizeof(IntStruct));
25   it -> value = 0;
26   it -> iptr  = NULL;
27   f(5, it);
28   free (it -> iptr);
29   free (it);
30   return EXIT_SUCCESS;
31 }
```

The program has **Segmentation fault** during execution.

- Which line causes Segmentation fault? If there are multiple places that can cause Segmentation fault, find the **first** one encountered when the program runs. You will receive no point if you mark two or more places.
- Explain the reason.