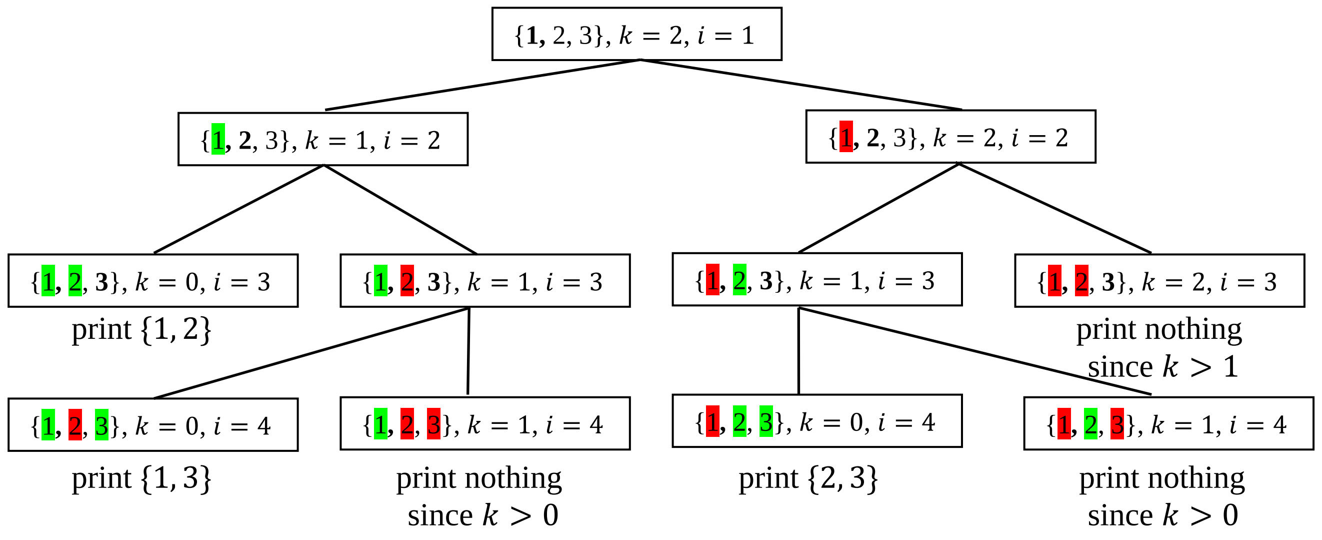
CSE 15
Introduction to Data Structures
Programming Assignment 1

In this assignment you will write a C program that uses recursion to print out all k-element subsets of the n-element set $\{1, 2, 3, \ldots, n\}$, where both n and k are given on the command line. Recall that the Binomial Coefficient $C(n, k)$ is defined to be the number of such subsets, and that these numbers can be computed recursively using Pascal's identity. For instance, if $n = 4$ and $k = 2$, one verifies $C(4, 2) = 6$, and that the 2-element subsets of $\{1, 2, 3, 4\}$ are: $\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}$. The core idea of our recursive solution is the same as in the proof (see notes from 6-25-19) of Pascal's identity.

$$C(n, k) = \begin{cases} 1 & \text{if} \quad k = 0 \text{ or } k = n \\ C(n-1, k-1) + C(n-1, k) & \text{if} \quad 0 < k < n \end{cases}$$

At each level of the recursion, we consider a particular element i in the set $\{1, 2, 3, \ldots, n\}$. First construct all k-elements subsets that include i, and then construct those subsets that exclude i. The following box trace illustrates the case $n = 3$, $k = 2$.



Observe that the element i under consideration at each level is in **bold**. Green highlight indicates that an element has been included in the subset, and red highlight indicates exclusion. For instance, at the top level, the element 1 is being considered. The left branch includes 1, and the right branch excludes it. The variable k always indicates the number of elements yet to be selected, so at each left branch k is decremented, while at each right branch it stays the same. When we reach a box in which $k = 0$, no more elements need to be selected, so we print the chosen (green) elements. These form a completed subset of the required size. At each box, the number of candidates remaining (i.e. those not highlighted) is $n - i + 1$. If we reach a box in which k is greater than this number, i.e. $k > n - i + 1$, we return without printing anything, since that box cannot generate a subset of size k.

As an exercise, construct a box trace for the case $n = 4$, $k = 2$, obtaining the six 2-element subsets listed above. (Be sure to use a large piece of paper when you do this.) It is highly recommended that you complete several such box traces by hand before you attempt to write any code for this project. In general, you cannot instruct a computer to do something that you cannot do yourself.

Representation of Subsets

We will use the following well-known approach to represent a subset of $\{1, 2, 3 \ldots, n\}$. First, create and array $B[\,]$ of length $n + 1$ (or more), then fill it with 0's and 1's according to the following rule. For any i in the range $1 \le i \le n$, set

$$B[i] = \begin{cases} 1 & \text{iff } i \text{ is included in the subset} \\ 0 & \text{iff } i \text{ is excluded from the subset} \end{cases}$$

Observe that the array element $B[0]$ is not be used, and if the length of $B[\,]$ is greater than $n + 1$, then the elements to the right of $B[n]$ are also not used. Thus the bit-array $B[1 \cdots 7] = (0, 0, 1, 0, 1, 1, 0)$ represents the subset $\{3, 5, 6\}$ of $\{1, 2, 3, 4, 5, 6, 7\}$. As an exercise, re-do the above box trace (and any other box traces you have performed) representing each subset as a bit array. (Note this exercise is essential to success in this assignment, don't fail to do it.)

Some students may dislike the idea of wasting $B[0]$. It is possible to write the required functions for this assignment without using this convention, but then those functions would not be compatible with the tests used to evaluate your program. To get full credit, you must represent subsets of $\{1, 2, 3, \ldots, n\}$ with an int array of length (at least) $n + 1$ containing only 0's and 1's in $B[1], B[2], B[3], \ldots, B[n]$. You can put something in $B[0]$ if you like, or something in $B[(n + 1) \cdots \cdots]$, but those values will not be examined during grading of your project.

There are two required functions in this assignment. A function with the heading

```
void printSet(int B[], int n)
```

will be included that prints the subset of $\{1, 2, 3, \ldots, n\}$ represented by the bit-array $B[1 \cdots n]$ using braces ("`{`" and "`}`"), commas and positive integers. Thus if $B[\,1 \cdots 7]$ is in the state $(*, 0, 0, 1, 0, 1, 1, 0)$, then the string "`{3, 5, 6}`" will be printed. Here $*$ represents whatever is in $B[0]$. Note that the empty set will be represented as an array of all 0's, and the above function will print the string "`{ }`". (That's a pair of open and closing braces with a single space between.) Your program will include another function with heading

```
void printSubsets(int B[], int n, int k, int i)
```

that implements the recursive algorithm described above. Specifically, a call to $\text{printSubsets}(B, n, k, i)$ will print solutions to the following subinstance: assume that the elements $\{1, 2, \ldots, (i - 1)\}$ have already been decided upon (included or excluded), then extend that partial solution in all possible ways to obtain a k-element subset of $\{1, 2, 3, \ldots, n\}$.

You may assume that in all tests of your program, the value n will not exceed 100. Therefore you should `#define` a constant macro `MAX_SIZE` to be 100, and define the array $B[\,]$ to be of length `MAX_SIZE+1`. (This is why there may be unused array elements to the right of $B[n]$.) Function `main()` will read command line arguments, define and initialize the int array $B[\,]$, then call the required functions as appropriate. You may of course include other helper functions as you see fit.

Program Operation

Your program will be called `Subset.c` and will operate as follows.

```
$ Subset
Usage: Subset n k (n and k are ints satisfying 0<=k<=n<=100)
$ Subset foo bar
Usage: Subset n k (n and k are ints satisfying 0<=k<=n<=100)
$ Subset 1.2 3.4
Usage: Subset n k (n and k are ints satisfying 0<=k<=n<=100)
$ Subset 2 4
Usage: Subset n k (n and k are ints satisfying 0<=k<=n<=100)
$ Subset 4 2
{1, 2}
{1, 3}
{1, 4}
{2, 3}
{2, 4}
{3, 4}
$ Subset 5 3
{1, 2, 3}
{1, 2, 4}
{1, 2, 5}
{1, 3, 4}
{1, 3, 5}
{1, 4, 5}
{2, 3, 4}
{2, 3, 5}
{2, 4, 5}
{3, 4, 5}
$ Subset 3 3
{1, 2, 3}
$ Subset 3 0
{ }
$ Subset 8 0
{ }
$
```

As you can see, if anything other than two command line arguments are supplied, or if those two arguments cannot be parsed as ints, or if they can be parsed as ints, but do not satisfy the inequality $0 \leq k \leq n \leq$ `MAX_SIZE`, then a usage message will be printed, and the program will terminate. The required subsets are printed one to a line. Your program must match the above format (including the usage message) exactly to receive full credit.

What to turn in

Submit the files `README`, `Makefile`, and `Subset.c` to the assignment pa1 before the due date. Your Makefile must create an executable file called `Subset`, and must include a `clean` utility that removes all intermediate `.o` files as well as the executable file `Subset` itself. See Lab Assignment 1 to learn how to do this. Please start early and ask plenty of questions in lab sessions and office hours.