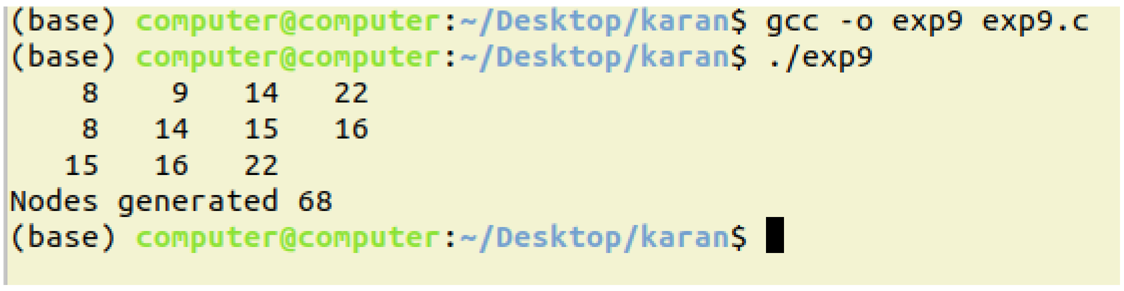
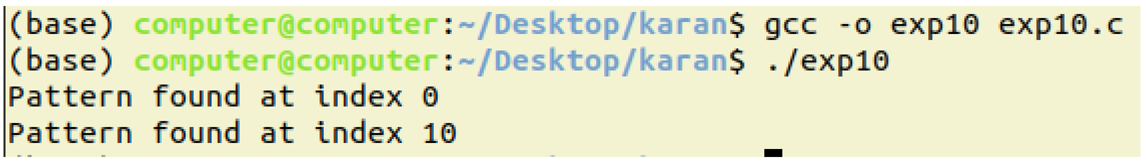
```
#include <stdio.h>
#include <stdlib.h>
#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))
static int total_nodes;
// prints subset found
void printSubset(int A[], int size)
{
for(int i = 0; i < size; i++)
{
printf("%*d", 5, A[i]);
}
printf("\n");
}
// qsort compare function
int comparator(const void *pLhs, const void *pRhs)
{
int *lhs = (int *)pLhs;
int *rhs = (int *)pRhs;
return *lhs > *rhs;
}
// inputs
// s - set vector
// t - tuplet vector
// s_size - set size
// t_size - tuplet size so far
// sum - sum so far
// ite - nodes count
// target_sum - sum to be found
void subset_sum(int s[], int t[],
int s_size, int t_size,
int sum, int ite,
int const target_sum)
{
total_nodes++;
if( target_sum == sum )
{
// We found sum
printSubset(t, t_size);

// constraint check
if( ite + 1 < s_size && sum - s[ite] + s[ite+1] <= target_sum )
{
// Exclude previous added item and consider next candidate
subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1,
target_sum);
}
return;
}
else
{
// constraint check
if( ite < s_size && sum + s[ite] <= target_sum )
{
// generate nodes along the breadth
for( int i = ite; i < s_size; i++ )
{
t[t_size] = s[i];
if( sum + s[i] <= target_sum )
{
// consider next level node (along depth)
subset_sum(s, t, s_size, t_size + 1, sum + s[i], i +
1, target_sum);
}
}
}
}
}
// Wrapper that prints subsets that sum to target_sum
void generateSubsets(int s[], int size, int target_sum)
{
int *tuplet_vector = (int *)malloc(size * sizeof(int));
int total = 0;
// sort the set
qsort(s, size, sizeof(int), &comparator);
for( int i = 0; i < size; i++ )
{
total += s[i];
}
if( s[0] <= target_sum && total >= target_sum )
{

subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
}
free(tuplet_vector);
}
int main()
{
int weights[] = {15, 22, 14, 26, 32, 9, 16, 8};
int target = 53;
int size = ARRAYSIZE(weights);
generateSubsets(weights, size, target);
printf("Nodes generated %d\n", total_nodes);
return 0;
}
```



```
(base) computer@computer:~/Desktop/karan$ gcc -o exp9 exp9.c
(base) computer@computer:~/Desktop/karan$ ./exp9
    8     9    14    22
    8    14    15    16
   15    16    22
Nodes generated 68
(base) computer@computer:~/Desktop/karan$ 
```

```
#include<stdio.h>
#include<string.h>
// d is the number of characters in the input alphabet
#define d 256
/* pat -> pattern
txt -> text
q -> A prime number
*/
void search(char pat[], char txt[], int q)
{
int M = strlen(pat);
int N = strlen(txt);
int i, j;
int p = 0; // hash value for pattern
int t = 0; // hash value for txt
int h = 1;
// The value of h would be "pow(d, M-1)%q"
for (i = 0; i < M-1; i++)
h = (h*d)%q;
// Calculate the hash value of pattern and first
// window of text
for (i = 0; i < M; i++)
{
p = (d*p + pat[i])%q;
t = (d*t + txt[i])%q;
}
// Slide the pattern over text one by one
for (i = 0; i <= N - M; i++)
{
// Check the hash values of current window of text
// and pattern. If the hash values match then only
// check for characters one by one
if ( p == t )
{
/* Check for characters one by one */
for (j = 0; j < M; j++)
{
if (txt[i+j] != pat[j])
break;
}

// if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]
if (j == M)
printf("Pattern found at index %d \n", i);
}
// Calculate hash value for next window of text: Remove
// leading digit, add trailing digit
if ( i < N-M )
{
t = (d*(t - txt[i]*h) + txt[i+M])%q;
// We might get negative value of t, converting it
// to positive
if (t < 0)
t = (t + q);
}
}
}
int main()
{
char txt[] = "GEEKS FOR GEEKS";
char pat[] = "GEEK";
// A prime number
int q = 101;
// function call
search(pat, txt, q);
return 0;
}
```



```
(base) computer@computer:~/Desktop/karan$ gcc -o exp10 exp10.c
(base) computer@computer:~/Desktop/karan$ ./exp10
Pattern found at index 0
Pattern found at index 10
```