ESC101: Introduction to Computing

f(unction)

Dot Product

- Problem: write a function dot_product that takes as argument two integer arrays, a and b, and an integer, size, and computes the dot product of a and b.
- Declaration of dot_product

```
int dot_product(int a[], int b[], int);
```

OR

```
int dot_product(int [], int [], int);
```

```
#include<stdio.h>
int dot_product (int[], int[], int);
int main(){
   int vec1[] = {2,4,1,7,-5,0, 3, 1};
   int vec2[] = {5,7,1,0,-3,8,-1,-2};
   printf("%d\n", dot_product(vec1, vec1, 8));
   printf("%d\n", dot_product(vec1, vec2, 8));
   return 0;
}
int dot_product (int a[], int b[], int size){

}
```

$$p = \sum_{i=1}^{size} (a_i \times b_i)$$

Convert to C

OUTPUT
105
49

```
#include<stdio.h>
int dot_product (int[], int[], int);
int main(){
   int vec1[] = {2,4,1,7,-5,0, 3, 1};
   int vec2[] = {5,7,1,0,-3,8,-1,-2};
   printf("%d\n", dot_product(vec1, vec1, 8));
   printf("%d\n", dot_product(vec1, vec2, 8));
   return 0;
}
int dot_product (int a[], int b[], int size){
   int p = 0, i;
   for(i=0;i<size; i++)
       p = p + (a[i]*b[i]);
   return p;
}
```

OUTPUT

105

49

Generating Prime Numbers

- Problem: Given a positive integer N, generate all prime numbers up to N.
- A Greek mathematician Eratosthenes came up with a simple but fast algorithm
- Sieve of Eratosthenes

Sieve of Eratosthenes

- Write down all the integers starting from 2 till N.
- Starting from 2 strike off all multiples of 2, except 2.
- Next, find the first number that has not been struck and strike off all its multiples, except the number.
- Continue until you cannot strike out any more numbers.
- The numbers that have not been struck, are PRIMES.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Generating Prime Numbers using Sieve of Eratosthenes

- No more numbers can be marked. Algorithm terminates.
- Primes up to 100 are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

Sieve of Eratosthenes: Program

```
int prim[10000]; // global array
void sieve(int n) {
  int i, j = 2;
  prim[0]=0; prim[1]=0;
  for (i=2; i<=n; i++)  prim[i] = 1;
  while (j <= n) {
      if (prim[j] == 0) { // composite
      j++;      continue;
      }
      for (i=2*j; i<=n; i=i+j)
              prim[i] = 0;
      j++;
  }
}
```

```
int main() {
  int i, n;
  scanf("%d", &n);
  // check n < 10000

  sieve(n); // set primes

  for (i=2; i<=n; i++) {
       if (prim[i] == 1)
            printf("%d\n", i);
  }

  return 0;
}
```

Printing strings

Strings are printed using %s option.

E.g. 1

```
printf("%s", "I am DON");
```

Output

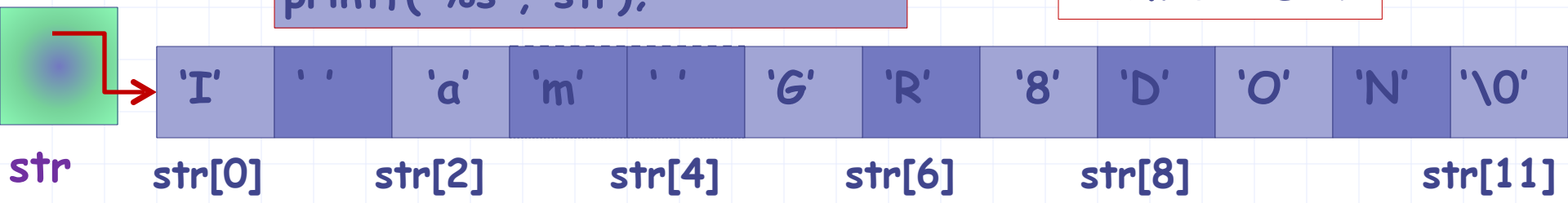
```
I am DON
```

E.g. 2

```
char str[]="I am GR8DON";
printf("%s", str);
```

Output

```
I am GR8DON
```



State of memory after definition of str in E.g. 2. Note the NULL char added in the end.

This NULL char is not printed.

```
#include <stdio.h>

int main() {
 char str1[20], str2[20];

 scanf("%s",str1);
 scanf("%s",str2);

 printf("%s + %s\n", str1, str2);

 return 0;
}
```

INPUT
IIT Kanpur

OUTPUT
IIT + Kanpur

INPUT
I am DON

OUTPUT
I + am

Other string functions

- Return length of a string.
- Concatenates one string with another.
- Search for a substring in a given string.
- Reverse a string
- Find first/last/k-th occurrence of a character in a string
  - … and more
- Case sensitive/insensitive versions

string.h

- Header file with functions on strings
- strlen(s): returns length of string s (without '\0')
- strcpy(d, s): copies s into d
- strcat(d, s): appends s at the end of d ('\0' is moved to the end of result)

string.h

- strcmp(s1, s2):
  - return an integer less than 0 if s1 is less than s2,
  - returns 0 if s1 equal to s2,
  - returns an integer greater than 0 if s1 is greater than s2.
- Example:

```
char str1[] = "Hello", str2[] = "Helpo";
int i = strcmp(str1,str2);
if (i > 0){ printf("str1 is greater than str2");}
else if (i<0){ printf("str2 is greater than
str1");}
else {printf("str1 is same as str2");}
```

string.h

- strncpy(d, s, n)
- strncat(d, s, n)
- strncmp(d, s, n)
  - restrict the function to “n” characters at most (argument n is an integer)
  - first two functions - truncate the string s to the first “n” characters.
  - third function - truncate the strings d, s to the first “n” characters.

```
char str1[] = "Hello", str2[] = "Helpo";
printf("%d",strncmp(str1,str2,3));
```

0

string.h

- strcasecmp, strncasecmp:

  case insensitive comparison.

- Example:

```
char str1[] = "HELLO", str2[] = "Hello";
int i = strcmp(str1,str2);
int j = strcasecmp(str1,str2);
```

string.h

- strcasecmp, strncasecmp:

  case insensitive comparison.

- Example:

```
char str1[] = "HELLO", str2[] = "Hello";
int i = strcmp(str1,str2);
int j = strcasecmp(str1,str2);
```

```
-1 0
```

- strcmp gives -1 because ‘E’ < ‘e’ .
  - ‘E’-‘e’ = -32 .

string.h

- Many more utility functions.

- strupr(s) : converts lower to upper case.
- strlwr(s) : converts upper to lower case.
- strstr(S,s) : searches s in S. Returns a pointer to the first occurrence.

- All functions depend on '\0' as the end-of-string marker.

ESC101: Introduction to Computing

Multi dimensional Arrays

Why Multidimensional Arrays?

- Marks of 800 students in 5 subjects each.
- Distance between cities
- Sudoku
- All the above require 2D arrays
- Properties of points in space (Temperature, Pressure etc.)
- Mathematical Plots
- > 2D arrays

Multidimensional Arrays

Multidimensional arrays are defined like this:

```
double  mat[5][6];
```
OR
```
int mat[5][6];
```
OR
```
float mat[5][6];
```
etc.

The definition states that mat is a 5 X 6 matrix of doubles (or ints or floats). It has 5 rows, each row has 6 columns, each entry is of type double.

2.1	1.0	-0.11	-0.87	31.5	11.4
-3.2	-2.5	1.678	4.5	0.001	1.89
7.889	3.333	0.667	1.1	1.0	-1.0
-4.56	-21.5	1.0	-1.0	5.0	-5.78
45.7	26.9	-0.001	1000.09	15.1	1.0

Accessing matrix elements-I

1. The (i,j)th member of mat is accessed as mat[i][j]. Note the slight difference from the matrix notation in maths.
2. The row and column numbering each start at 0 (not 1).
3. The following program prints the input matrix.

```
void print_matrix(float mat[5][6]) {
    int i,j;
    for (i=0; i < 5; i=i+1) {
      for (j=0; j < 6; j = j+1) {
      printf("%f ", mat[i][j]);
      }
      printf("\n");
    }
}
```

/* prints the ith row i = 0..4. */

/* In each row, prints each of the six columns j=0..5 */

/* prints a newline after each row */

Accessing matrix elements-II

1. Code for reading the matrix.
2. The address of the i,j th matrix element is &mat[i][j].
3. This works without parentheses since the array indexing operator [] has higher precedence than &.

```
void read_matrix(float mat[5][6]) {
   int i,j;
   for (i=0; i < 5; i=i+1) {
        for (j=0; j < 6; j = j+1) {
      scanf("%f ", &mat[i][j]);
       }
    }
}
```

/* read the ith row i = 0..4. */

/* In each row, read each of the six columns j=0..5 */

scanf with %f option will skip over whitespace.

So it doesn't matter whether the entire input is given in 5 rows of 6 floats in a row or all 30 floats in a single line.

Initializing 2 dimensional arrays

We want a[4][3] to be this 4 X 3 int matrix.

1	2	3
4	5	6
7	8	9
0	1	2

Initialize as

```
int  a[][3] = {
        {1,2,3},
          {4,5,6},
        {7,8,9},
          {0,1,2}
        };
```

Initialization rules:

1. Most important: values are given row-wise, first row, then second row, so on.
2. Number of columns must be specified.
3. Values in each row are enclosed in braces {…}.
4. Number of values in a row may be less than the number of columns specified. Remaining col… '\0' for char, etc.)

```
int a[][3] = { {1}, {2,3}, {3,4,5} };
```

gives this matrix for a:

1	0	0
2	3	0
3	4	5

Accessing matrix elements

```
void read_matrix(double mat[5][6]) {
   int i,j;
   for (i=0; i < 5; i=i+1) {
        for (j=0; j < 6; j = j+1) {
      scanf("%f ", &mat[i][j]);
       }
    }
}
```

/* read the ith row i = 0..4. */

/* In each row, read each of the six columns j=0..5 */

Can we change the formal parameter to mat[6][5]? Would it mean the same? Or mat[10][3]?

That would not be correct. It would change the way elements of mat are addressed. We will discuss this in details later.

Practice Problem

- We are provided with a 3x3 matrix. We should output whether it is an identity matrix or not

Input : 1 0 0 Output: It is an identity matrix

0 1 0

0 0 1

Input: 2 1 0 Output: It is not an identity matrix

0 0 1

0 1 0