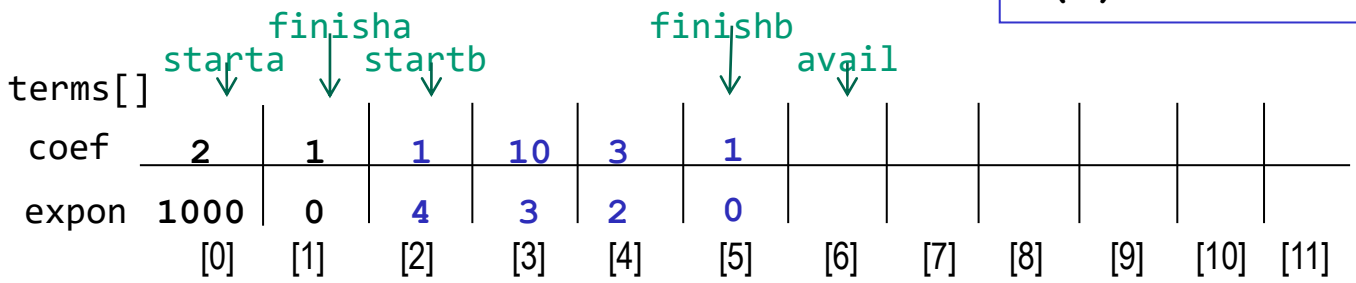
Polynomial Addition (1)

```
void padd (int starta, int finisha, int startb, int finishb,
           int* startd, int* finishd)
{
  float coefficient;
  *startd = avail;
  while (starta <= finisha && startb <= finishb){
    switch(COMPARE(terms[starta].expon,terms[startb].expon)){
      case -1: /* a.expon < b.expon */
        attach(terms[startb].coef, terms[startb].expon);
        startb++;
        break;
      case 0: /* equal exponents */
        coefficient = terms[starta].coef + terms[startb].coef;
        if (coefficient)
           attach(coefficient,terms[starta].expon);
        starta++;
        startb++;
        break;
      case 1: /*  a.expon > b.expon */
        attach(terms[starta].coef, terms[starta].expon);
        starta++;
    } /* switch */
  } /* while */
```

$A(x) = 2x^{1000} + 1,$

$B(x) = x^4 + 10x^3 + 3x^2 + 1$



Polynomial Addition (2)

```
  /* add in remaining terms of A(x) */
  for(; starta <= finisha; starta++) {
     attach(terms[starta].coef, terms[starta].expon);
  }
  /* add in remaining terms of B(x) */
  for(; startb <= finishb; startb++) {
     attach(terms[startb].coef, terms[startb].expon);
  }
  *finishd = avail-1;
} /* end of padd */

void attach(float coefficient, int exponent)
{
    /* add a new term to the polynomial */
    if (avail >= MAX_TERMS) {
        fprintf(stderr, "too many terms!!!\n");
        exit(1);
    }
    terms[avail].coef = coefficient;
    terms[avail++].expon = exponent;
}
```

$A(x) = 2x^{10} + 2x^3 + x + 1,$
$B(x) = x^4 + 10x^3 + 3x^2$

$A(x) = 2x^5 + 2x^4,$
$B(x) = x^4 + 5x^3 + x^2 + 4$

Matrix

- m rows and n columns
- Standard representation : two-dimensional array as `a[MAX_ROWS][MAX_COLS]`

$$\begin{bmatrix} -27 & 3 & 4 \\ 6 & 82 & -2 \\ 109 & -64 & 11 \\ 12 & 8 & 9 \\ 48 & 27 & 47 \end{bmatrix}$$

```
a[0][0]=-27;
a[1][2]=-2;
a[3][1]=8;
```

Sparse Matrix

- Sparse matrix has many zero entries

$$\frac{\text{no. of non-zero elements}}{\text{no. of total elements}} << \text{small}$$

➔ two-dimensional array wastes space

➔ Represent using the triple <row, col, value>

$\frac{15}{15}$

	col0	col1	col2
row0	-27	3	4
row1	6	82	-2
row2	109	-64	11
row3	12	8	9
row4	48	27	47

$\frac{8}{36}$

	col0	col1	col2	col3	col4	col5
row0	**15**	0	0	**22**	0	**-15**
row1	0	**11**	**3**	0	0	0
row2	0	0	0	**-6**	0	0
row3	0	0	0	0	0	0
row4	**91**	0	0	0	0	0
row5	0	0	**28**	0	0	0

Sparse Matrix Abstract Data Type

ADT *SparseMatrix* is

objects: a set of triples, <row, column, value>, where row and column are integers and from a unique combination, and value comes from the set item

functions:

for all a, b ∈ *SparseMatrix*, x ∈ *item*, i,j,max_col, max_row ∈ *index*

SparseMatrix Create(maxRow, maxCol)::=

SparseMatrix Transpose(a)::=

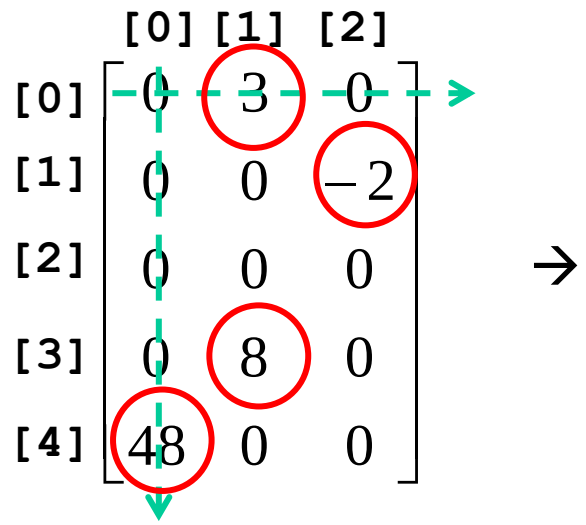
SparseMatrix Add(a,b)::=

SparseMatrix Mutiply(a,b)::=

end *SparseMatrix*

Sparse Matrix Representation

```
#define MAX_TERMS 101
typedef struct {
        int row;
        int col;
        int value;
} term;
term a[MAX_TERMS];
```



Number of elements

Number of columns

Number of rows

	row	col	value
a[0]	5	3	4
a[1]	0	1	3
a[2]	1	2	-2
a[3]	3	1	8
a[4]	4	0	48

Approximate Memory Requirements

500 x 500 matrix with 1994 nonzero elements,
4 bytes per element

2D array: 500 x 500 x 4 = 1M bytes

1D array of triples: 3 x 1994 x 4 ≈ 23K bytes

Quiz 7

Name and student ID

We have a 500 x 500 matrix. Find the smallest number of nonzero elements such that the memory space of 2-D array representation becomes less than that of the sparse matrix representation.

Transposing a Matrix (1)

Transpose operation ($A(i,j) \rightarrow A(j,i)$)

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 15 & 0 & 0 & 0 & 91 & 0 \\ 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 28 \\ 22 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -15 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28

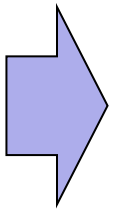
→

	row	col	value
b[0]	6	6	8
b[1]	0	0	15
b[2]	0	4	91
b[3]	1	1	11
b[4]	2	1	3
b[5]	2	5	28
b[6]	3	0	22
b[7]	3	2	-6
b[8]	5	0	-15

Transposing a Matrix (2)

```
for each row i
  take element <i,j,value> and store it as element
  <j,i,value>
```

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28



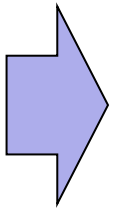
	row	col	value
a[0]			
a[1]			
a[2]			
a[3]			
a[4]			
a[5]			
a[6]			
a[7]			
a[8]			

a[??] 3 0 22

Transposing a Matrix (3)

```
for all elements in column j
  place element <i,j,value> in element <j,i,value>
```

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28



	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	4	91
a[3]	1	1	11
a[4]	2	1	3
a[5]	2	5	28
a[6]	3	0	22
a[7]	3	2	-6
a[8]	5	0	-15

Transposing a Matrix (3)

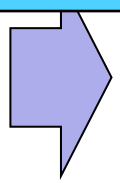
```
void transpose(term a[], term b[])
{
 /* b is set to the transpose of a*/
 int n, i, j, currentb;
 n = a[0].value;     /* total number of elements */
 b[0].row = a[0].col; /* rows in b = columns in a */
 b[0].col = a[0].row; /* columns in b = rows in a */
 b[0].value = n;
 if (n > 0) { /* non-zero matrix*/
    currentb = 1;
    for (i=0; i<a[0].col; i++){ /*transpose by the column*/
      for (j = 1; j <= n; j++){
        if (a[j].col == i){
           b[currentb].row = a[j].col;
           b[currentb].col = a[j].row;
           b[currentb].value = a[j].value;
           currentb++;
        } /* if a[j].col */
      } /* for j */
    } /* for i */
  } /* if n */
}
```

→ O (*columns* · *elements*)

Fast Matrix Transpose (1)

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28

3	0	22



	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]			
a[3]			
a[4]			
a[5]			
a[6]	3	0	22
a[7]			
a[8]			

2 elements in row 0
1 element in row 1
2 elements in row 2

Fast Matrix Transpose (2)

Step 1: #non-zero in each row of transpose = #non-zero in each column of original matrix

Step2: Starting position of each row of transpose = sum of size of preceding rows of transpose

Step 3: Move elements, left to right, from original matrix to transpose matrix

Number of non-zero elements in each row of transpose matrix

Starting position of each row of transpose matrix

Original matrix

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28

rowTerms

[0]	2
[1]	1
[2]	2
[3]	2
[4]	0
[5]	1

startingPos

[0]	1
[1]	3
[2]	4
[3]	6
[4]	8
[5]	8

```
startingPos[0] = 1;
for(i = 1; i < num_cols; i++){
  startingPos[i]=
     startingPos[i-1]+
     rowTerms[i-1];
}
```

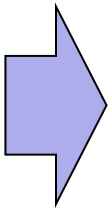
a[2]: <0,3,22> → b[startingPos[3]]: <3,0,22>

```
for(i = 1; i <= numTerms; i++) {
  j = startingPos[a[i].col];
  b[j].row=a[i].col;  b[j].col=a[i].row; b[j].value=a[i].value;
  startingPos[a[i].col]++;
}
```

	[0]	[1]	[2]	[3]	[4]	[5]
startingPos	3	4	6	8	8	9

Original matrix

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28



Transpose matrix

	row	col	value
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	4	91
a[3]	1	1	11
a[4]	2	1	3
a[5]	2	5	28
a[6]	3	0	22
a[7]	3	2	-6
a[8]	5	0	-15

→ O (*columns + elements*)

Fast Matrix Transpose (2)

```
void fastTranspose(term a[], term b[])
{
    int rowTerms[MAX_COL], startingPos[MAX_COL];
    int i,j, numCols = a[0].col, numTerms = a[0].value;
    b[0].row = numCols; b[0].col = a[0].row;
    b[0].value = numTerms;
    if (numTerms > 0) {/* non-zero matrix*/
        for(i = 0; i < numCols; i++) { rowTerms[i] = 0; }
        for(i=1; i<= numTerms; i++){rowTerms[a[i].col]++;}
        startingPos[0] = 1;
        for(i = 1; i < numCols; i++){
          startingPos[i]=startingPos[i-1]+rowTerms[i-1];
        }
        for(i = 1; i <= numTerms; i++) {
            j = startingPos[a[i].col];
            b[j].row = a[i].col;  b[j].col = a[i].row;
            b[j].value = a[i].value;
            startingPos[a[i].col]++;
        }
    }
}
```

→ O (columns + elements)

Quiz 8

Name and student ID

Rewrite the following codes without rowTerms[] to save memory.

Hint: Reuse startingPos[].

```
for(i=1; i<= numTerms; i++){rowTerms[a[i].col]++;}
startingPos[0] = 1;
for(i = 1; i < numCols; i++){
        startingPos[i]=startingPos[i-1]+rowTerms[i-1];}
```

String Abstract Data Type (1)

ADT *String* is

objects: a finite set of zero or more characters

functions:

for all s, t ∈String, i,j,m ∈ non-negative integers

String Null(m) ::= …

Integer Compare(s,t)::= …

Boolean IsNull(s) ::= …

Integer Length(s) ::= …

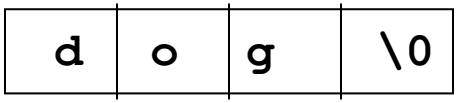
String Concat(s,t)::= …

String Substr(s,i,j)::= …

end String

String representation in C

```
#define MAX_SIZE 100
char s[MAX_SIZE] = {"dog"};
```



String Abstract Data Type (2)

C String functions

- `char *strcat(char *dest, char *src)`
- `char *strncat(char *dest, char *src, int n)`
- `int strcmp(char *str1, char *str2)`
- `int strncmp(char *str1, char *str2, int n)`
- `char *strcpy(char *dest, char *src)`
- `char *strncpy(char *dest, char *src, int n)`
- `size_t strlen(char *s)`
- `char *strchr(char *s, int c)`
- `char *strtok(char *s, char *delimiters )`
- `char *strstr(char *s, char *pat)`
- `size_t strspn(char *s, char *spanset)`
- `size_t strcspn(char *s, char *spanset)`
- `char *strpbrk(char *s, char *spanset)`

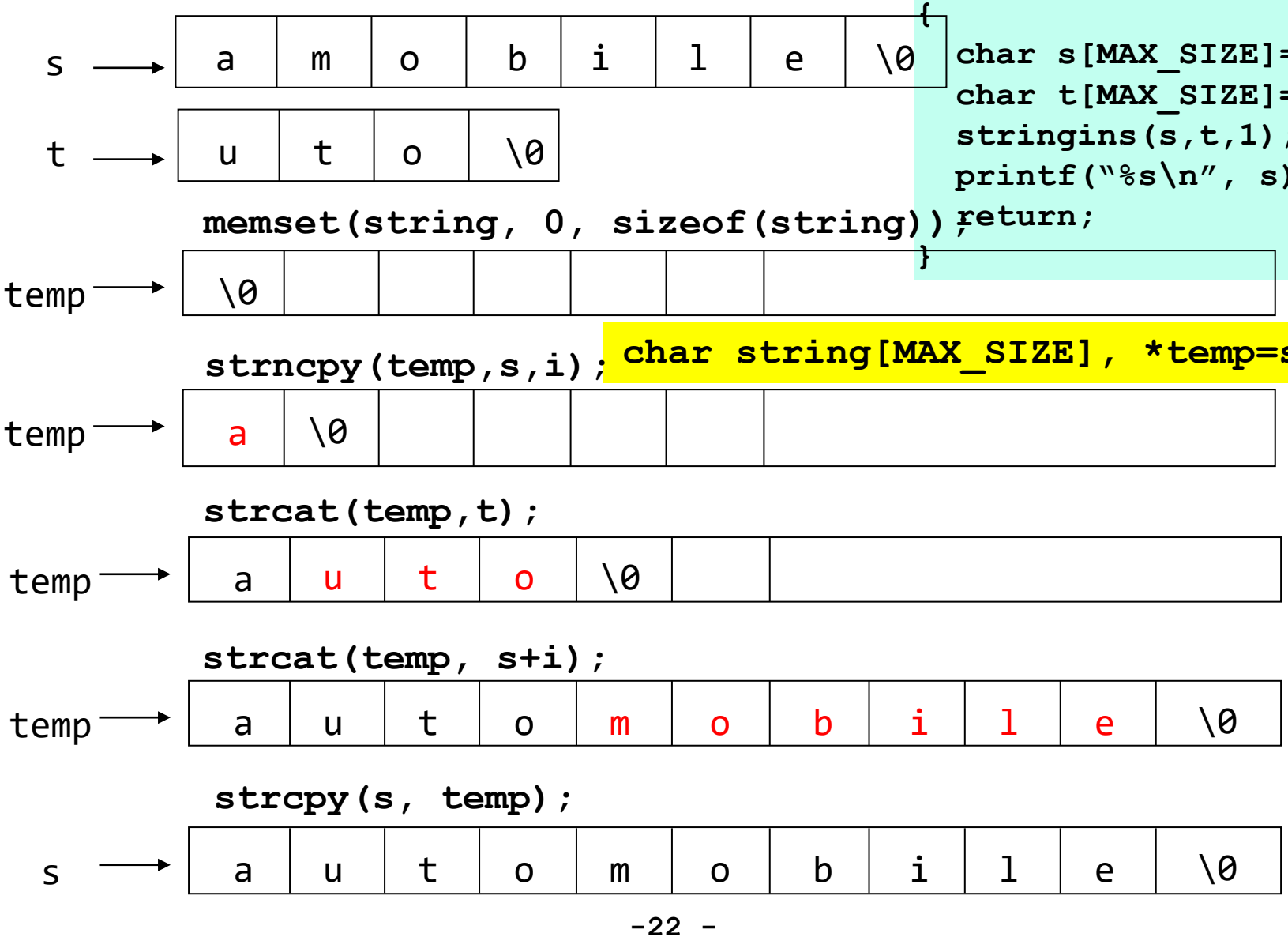
String Insertion (1)

```
void stringins (char *s, char *t, int i)
{
   char string[MAX_SIZE], *temp=string;
   memset(string, 0, sizeof(string));

   if (i<0 || i>strlen(s)) {
       fprintf (stderr, "position is out of bounds\n");
       exit(1);
   }
   if (!strlen(s)) strcpy(s,t);
   else if (strlen(t)){
       strncpy(temp, s, i);
       strcat(temp, t);
       strcat(temp, s+i);
       strcpy(s, temp);
   }
}
```

String Insertion (2)

```
void main()
{
  char s[MAX_SIZE]="amobile";
  char t[MAX_SIZE]="uto";
  stringins(s,t,1);
  printf("%s\n", s);
  return;
}
```



Quiz 9

Name and student ID

Write the string remove function to remove j characters beginning from i in string s.

```
void stringremove (char *s, int i, int j) {



}
```