PDS Class Test 2

- Date: October 27, 2016
- Time: 7pm to 8pm
- Marks: 20 (Weightage 50%)

Room	Sections	No of students
V1	Section 8 (All) Section 9 (AE,AG,BT,CE, CH,CS,CY,EC,EE,EX)	101 49
V2	Section 9 (Rest, if not allotted in V1) Section 10 (All)	50 98
V3	Section 11 (All)	98
V4	Section 12 (All)	94
F116	Section 13 (All)	95
F142	Section 14 (All)	96

Let us establish the Pointer from Autumn Break to PDS!!!

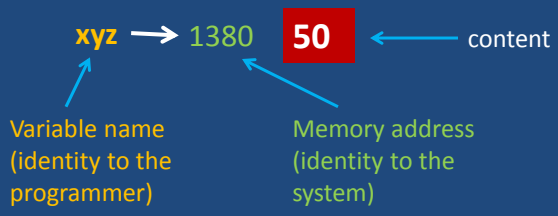
Example

- Consider the statement

  int xyz = 50;

  – This statement instructs the compiler to allocate a location for the integer variable xyz, and put the value 50 in that location.

  – Suppose that the address location chosen is 1380.



Pointers

– Variables that hold memory addresses are called pointers.

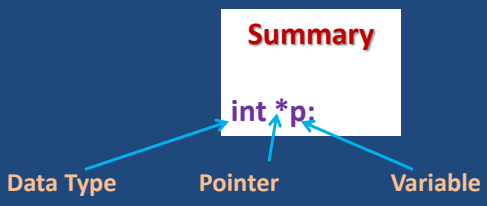
– Since a pointer is a variable, its value is also stored in some memory location.

Variable	Value	Address
xyz	50	1380
p	1380	2545

p = &xyz;

Declaration of pointer

- int xyz;
- int *p;
- p=&xyz;



- printf("%d",xyz); is equivalent to printf("%d",*p);
- So xyz and *p can be used for same purpose.
- Both can be declared simultaneously.
  - Example:
    - int xyz,*p;

Typecasting

- Typecasting is mostly not required in a well written C program. However, you can do this as follows:
  - char c = '5'
  - char *d = &c;
  - int *e = (int*)d;
  - Remember (sizeof(char) != sizeof(int))

Examples of pointer arithmetic

```
int a=10, b=5, *p, *q;
p=&a;
q=&b;
printf("*p=%d,p=%x\n",*p,p);
p=p-b;
printf("*p=%d,p=%x\n",*p,p);
printf("a=%d, address(a)=%x\n",a,&a);
```

Output:

*p=10, p=24b3f6ac
*p=4195592, p=24b3f698
a=10, address(a)=24b3f6ac

Pointers and Arrays

- Pointers can be incremented and decremented by integral values.
- After the assignment p = &A[i]; the increment p++ (or ++p) lets p one element down the array, whereas the decrement p-- (or --p) lets p move by one element up the array. (Here "up" means one index less, and "down" means one index more.)
- Similarly, incrementing or decrementing p by an integer value n lets p move forward or backward in the array by n locations. Consider the following sequence of pointer arithmetic:
  - `p = A;` /* Let p point to the 0-th location of the array A */
  - `p++;` /* Now p points to the 1-st location of A */
  - `p = p + 6;` /* Now p points to the 8-th location of A */
  - `p += 2;` /* Now p points to the 10-th location of A */
  - `--p;` /* Now p points to the 9-th location of A */
  - `p -= 5;` /* Now p points to the 4-rd location of A */
  - `p -= 5;` /* Now p points to the (-1)-nd location of A */

Remember: Increment/ Decrement is by data type not by bytes.

Example

- Consider the declaration:

```
int *p;
int  x[5]  =  {1, 2, 3, 4, 5} ;
```

  - Suppose that the base address of x is 2500, and each integer requires 4 bytes.

Element	Value	Address
x[0]	1	2500
x[1]	2	2504
x[2]	3	2508
x[3]	4	2512
x[4]	5	2516

  - Relationship between p and x:

```
p    =  &x[0]  =  2500
p+1  =  &x[1]  =  2504
p+2  =  &x[2]  =  2508
p+3  =  &x[3]  =  2512
p+4  =  &x[4]  =  2516
```

Accessing Array elements

NOTE

1. The name of the array is the starting address (base address) of the array.
2. It is the address of the first element in the array.
3. Thus it can be used as a normal pointer, to access the other elements in the array.

```
#include<stdio.h>
int main()
{
        int iarray[5]={1,2,3,4,5};
        int i, *ptr;
        ptr=iarray;
        for(i=0;i<5;i++) {
                printf("iarray[%d] (%x): %d\n",i,ptr,*ptr);
                ptr++;
                printf("iarray[%d] (%x): %d\n",i, (iarray+i),*(iarray+i));
        }
        return 0;
}
```

Swapping two numbers

```
void swap(int *a, int *b)
{
        int temp = *a;
        *a = *b;
        *b = temp;
}
```

~~void swap(int a, int b)~~
~~{~~
~~int temp = a;~~
~~a = b;~~
~~b = temp;~~
~~}~~

```
void main( )
{
    int i, j;
    scanf("%d %d", &i, &j);
    printf("After swap: %d %d",i,j);
    swap(&i,&j);
    printf("After swap: %d %d",i,j);
}
```

Revisited
Character Array / String

Declaring String Variables

- A string is declared like any other array:

```
char  string-name [size];
```

  - size determines the number of characters in string_name.

- When a character string is assigned to a character array, it automatically appends the null character ('\0') at the end of the string.
  - size should be equal to the number of characters in the string plus one.

Examples

```
char   name[30];
char   city[15];
char   dob[11];
```

- A string may be initialized at the time of declaration.

```
char   city[15] = "Calcutta";
char   city[15] = {'C', 'a', 'l', 'c', 'u', 't', 't', 'a'};

char   dob[] = "12-10-1975";
```

Equivalent

Reading "words"

- scanf can be used with the "%s" format specification.

```
char   name[30];
:
:
scanf ("%s", name);
```

  - The ampersand (&) is not required before the variable name with "%s".
  - The problem here is that the string is taken to be upto the first white space (blank, tab, carriage return, etc.)
    - If we type "Amit Ray"
    - name will be assigned the string "Amit"

Reading a "line of text"

- In many applications, we need to read in an entire line of text (including blank spaces).
- We can use the getchar() or gets() function for the purpose.
- Terminating criterion will be '\n'.

Writing Strings to the Screen

- We can use printf with the "%s" format specification.

```
char name[50];
:
:
printf ("\n %s", name);
```

Processing Character Strings

- There exists a set of C library functions for character string manipulation.
  - strcpy :: string copy
  - strlen :: string length
  - strcmp :: string comparison
  - strtcat :: string concatenation
  - …….
- It is required to include the following

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

strcpy()

- Works very much like a string assignment operator.

```
strcpy (string1, string2);
```

  - Assigns the contents of string2 to string1.

- Examples:

```
strcpy (city, "Calcutta");
strcpy (city, mycity);
```

- Warning:
  - Assignment operator do not work for strings.

```
city = "Calcutta";
```
➔ INVALID

strlen()

- Counts and returns the number of characters in a string.

```
len = strlen (string);   /* Returns an integer */
```

  - The null character ('\0') at the end is not counted.

  - Counting ends at the first null character.

strcmp()

- Compares two character strings.

  int strcmp (string1, string2);

  – Compares the two strings and returns 0 if they are identical; non-zero otherwise.

- Examples:

```
if  (strcmp (city, "Delhi") = = 0)
 {  .....  }

if  (strcmp (city1, city2) ! = 0)
  { ..... }
```

strcat()

- Joins or concatenates two strings together.

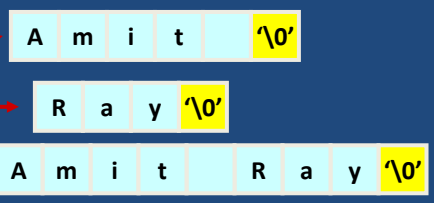
  strcat (string1, string2);

  – string2 is appended to the end of string1.

  – The null character at the end of string1 is removed, and string2 is joined at that point.

- Example:

```
strcpy (name1, "Amit ");
strcpy (name2, "Ray");
strcat  (name1, name2);
```



Multi Dimensional Arrays

Two Dimensional Arrays

- We have seen that an array variable can store a list of values.
- Many applications require us to store a table of values.
- The table contains a total of 20 values, five in each line.
  - The table can be regarded as a matrix consisting of four rows and five columns.
- C allows us to define such tables of items by using two-dimensional arrays.

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Student 1	75	82	90	65	76
Student 2	68	75	80	70	72
Student 3	88	74	85	76	80
Student 4	50	65	68	40	70

Declaring 2-D Arrays

- General form:

```
data_type   array_name [row_size][column_size];
```

- Examples:

```
int  marks[4][5];
float  sales[12][25];
double  matrix[100][100];
```

Accessing Elements of a 2-D Array

- Similar to that for 1-D array, but use two indices.
  - First indicates row, second indicates column.
  - Both the indices should be expressions which evaluate to integer values.

- Examples:

```
x[m][n] = 0;
c[i][k] += a[i][j] * b[j][k];
a = sqrt (a[j*3][k]);
```

Read the elements of a 2-D array

- By reading them one element at a time

```
for  (i=0; i<nrow; i++) {
   for  (j=0; j<ncol; j++) {
      scanf  (“%d”, &a[i][j]);
   }
}
```

- The ampersand (&) is necessary.
- The elements can be entered all in one line or in different lines.

Print the elements of a 2-D array

```
for  (i=0; i<nrow; i++)
      for  (j=0; j<ncol; j++)
         printf  (“\n %d”, a[i][j]);
```

– The elements are printed one per line.

```
for  (i=0; i<nrow; i++)
      for  (j=0; j<ncol; j++)
         printf  (“%d”, a[i][j]);
```

– The elements are all printed on the same line.

```
for  (i=0; i<nrow; i++)  {
      printf  (“\n”);
      for  (j=0; j<ncol; j++)
         printf (“%d   ”, a[i][j]);
   }
```

– The elements are printed nicely in matrix form.

Example: Matrix Addition

```
#include <stdio.h>

void main()
{
   int a[100][100], b[100][100],
         c[100][100], p, q, m, n;

   scanf ("%d %d", &m, &n);

   for (p=0; p<m; p++) {
      for (q=0; q<n; q++) {
        scanf ("%d", &a[p][q]);
      }
   }

   for (p=0; p<m; p++) {
      for (q=0; q<n; q++) {
        scanf ("%d", &b[p][q]);
      }
   }

   for (p=0; p<m; p++) {
      for (q=0; q<n; q++) {
        c[p][q] = a[p][q] + b[p][q];
      }
   }

   for (p=0; p<m; p++) {
      printf ("\n");
      for (q=0; q<n; q++) {
         printf ("%d ", a[p][q]);
      }
   }
}
```

How to print three matrices side by side?

```
2 3 4     1 2 3     3 5 7
2 1 3     6 7 5     8 8 8
2 1 5     3 3 3     5 4 8
```

Passing 2-D Arrays

- Similar to that for 1-D arrays.
  - The array contents are not copied into the function.
  - Rather, the address of the first element is passed.
- For calculating the address of an element in a 2-D array, we need:
  - The starting address of the array in memory.
  - Number of bytes per element.
  - Number of columns in the array.
- The above three pieces of information must be known to the function.

The Actual Mechanism

- When an array is passed to a function, the values of the array elements are not passed to the function.
  - The array name is interpreted as the address of the first array element.
  - The formal argument therefore becomes a pointer to the first array element.
  - When an array element is accessed inside the function, the address is calculated using the formula stated before.
  - Changes made inside the function are thus also reflected in the calling program.

Example Usage

```
#include  <stdio.h>

main()
{
   int  a[15][25],  b[15]25];
   :
   :
   add (a, b, 15, 25);
   :
}
```

```
void  add (int x[][25],int y[][25], int rows, int cols)
{
   :
}
```

We can also write
int x[15][25], y[15][25];

Number of columns

Example: Transpose of a matrix

```
void  transpose (int x[][100], int n)
{
   int  p, q;

   for  (p=0; p<n; p++) {
      for  (q=0; q<n; q++)
        {
           t = x[p][q];
           x[p][q] = x[q][p];
           x[q][p] = t;
        }
   }
}
```

10 20 30
40 50 60
70 80 90

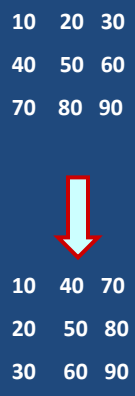
a[100][100]

transpose(a,3)

10 20 30
40 50 60
70 80 90

The Correct Version

```
void  transpose (int x[][100], n)
{
   int  p, q;

   for  (p=0; p<n; p++)
      for  (q=p; q<n; q++)
         {
            t = x[p][q];
            x[p][q] = x[q][p];
            x[q][p] = t;
         }
}
```



Multi-Dimensional Arrays

- How can you add more than two dimensions?
  - int a[100];
  - int b[100][100];
  - int c[100][100][100];
  - …….
  - How long?
  - Can you add any dimension?
  - Can you add any size?

Exercise

- Write a function to multiply two matrices of orders m x n and n x p respectively.

Homework

- Step -1: Read the number of persons from the user.
- Step -2: Read the first name of each of the persons.
- Step -3: Alphabetically sort their names.
- Step -4: Print the sorted list.

- Input:
  - Enter the number of persons: 3
  - Enter their first name:
    - Tridha
    - Susmita
    - Pranab
- Output:
    - Pranab
    - Susmita
    - Tridha

Multi Dimensional Array Initialization

- Example 1

```
int values[3][4] = {
                        {1,2,3,4},
                        {5,6,7,8},
                        {9,10,11,12}
                    };
```

- Example 2

```
int values[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

2D array to 1D array

- How?
  - Example 2D array

    1 2 3
    4 5 6
    7 8 9
  - Row-wise representation

    1 2 3 4 5 6 7 8 9
  - Column-wise representation

    1 4 7 2 5 8 3 6 9
- Why?
  - Chunk of memory is required.
  - May not be available.
  - 2D array of size 50X50 is available, but not 1D array of size 2500 **– POSSIBLE??**
  - 1D array of size 2500 is available, but not 2D array of size 50X50 **– POSSIBLE??**

2-D array representation in C

- Starting from a given memory location, the elements are stored row-wise in consecutive memory locations.

Example:

```
int A[5][4];
```

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]
A[3][0]	A[3][1]	A[3][2]	A[3][3]
A[4][0]	A[4][1]	A[4][2]	A[4][3]

2-D array representation in C

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]
A[3][0]	A[3][1]	A[3][2]	A[3][3]
A[4][0]	A[4][1]	A[4][2]	A[4][3]

A[0][0] A[0][1] A[0][2] A[0][3] A[1][0] A[1][1] A[1][2] A[1][3] A[2][0] A[2][1] A[2][2] A[2][3]

Row 0 **Row 1** **Row 2**

- x: starting address of the array in memory
- c: number of columns
- k: number of bytes allocated per array element

a[i][j] ➔ is allocated at x + (i * c + j) * k

Problems

1. Write a C program to multiply two matrices ~~of orders m x n and n x p respectively~~.

2. Write a C program to multiply to large matrices.

Interactive Input

```
#include <stdlib.h>
void main()
{
    int **mat,nrows,ncols,i;

        ……
    mat=(int **)malloc(sizeof(int *)*nrows);
    for(i=0;i<nrows;i++)
        mat[i]=(int *)malloc(sizeof(int)*ncols);
        ……
        ……

    for(i=0;i<nrows;i++)
        free(mat[i]);
    free(mat);
}
```

Memory allocation (2D pointer)

Memory allocation (1D pointer)

Memory deallocation (1D pointer)

Memory deallocation (2D pointer)

2-D Array Allocation

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

int **allocate(int h, int w)
  {
   int **p;
   int i,j;

  p=(int **) calloc(h, sizeof (int *) );
   for(i=0;i<h;i++)
    p[i]=(int *) calloc(w,sizeof (int));
   return(p);
  }
```

Allocate array of pointers

Allocate array of integers for each row

```
void read_data(int **p,int h,int w)
 {
   int i,j;
   for(i=0;i<h;i++)
   for(j=0;j<w;j++)
    scanf ("%d",&p[i][j]);
 }
```

Elements accessed like 2-D array elements.

2-D Array Allocation

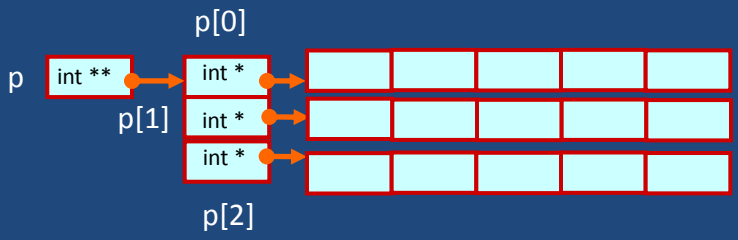
```
void print_data(int **p,int h,int w)
 {
   int i,j;
    for(i=0;i<h;i++)
    {
    for(j=0;j<w;j++)
      printf("%5d ",p[i][j]);
     printf("\n");
    }
}
```

```
void main()
{
  int **p;
  int M,N;

  printf("Give M and N \n");
  scanf("%d%d",&M,&N);
  p=allocate(M,N);
  read_data(p,M,N);
  printf("\n The array read as \n");
  print_data(p,M,N);
}
```

```
Give M and N
3 3
1 2 3
4 5 6
7 8 9

The array read as
  1   2   3
  4   5   6
  7   8   9
```

Pointer to Pointer

```
int **p;
p=(int **) malloc(3 * sizeof(int *));
p[0]=(int *) malloc(5 * sizeof(int));
p[1]=(int *) malloc(5 * sizeof(int));
p[2]=(int *) malloc(5 * sizeof(int));
```

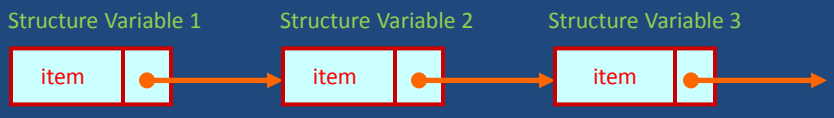


Linked List :: Basic Concepts

- A list refers to a set of items organized sequentially.
  - An array is an example of a list.
    - The array index is used for accessing and manipulation of array elements.
  - Problems with array:
    - The array size has to be specified at the beginning.
    - Deleting an element or inserting an element may require shifting of elements.

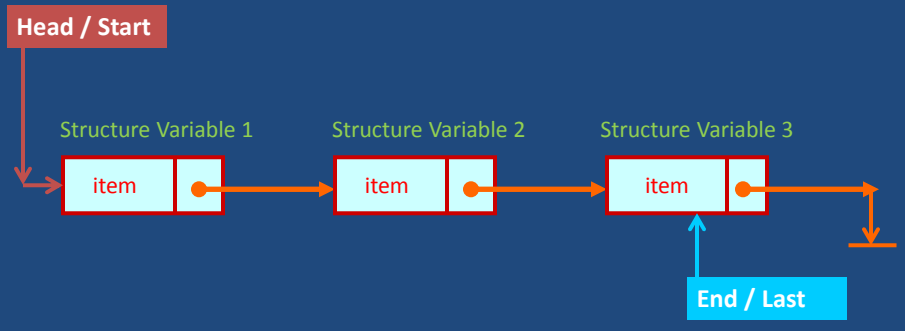
Linked List

- A completely different way to represent a list:
  - Make each item in the list part of a structure.
  - The structure contains the item and a pointer or link to the structure containing the next item.
  - This type of list is called a linked list.



Linked List

- Where to start and where to stop?

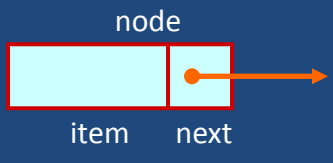


Linked List Facts

- Each structure of the list is called a node, and consists of two fields:
  - Item(s).
  - Address of the next item in the list.
- The data items comprising a linked list need not be contiguous in memory.
  - They are ordered by logical links that are stored as part of the data in the structure itself.
  - The link is a pointer to another structure of the same type.

Declaration of a linked list

```
struct node
       {
         int   item;
         struct node  *next;
       };
```



- Such structures which contain a member field pointing to the same structure type are called self-referential structures.