Introduction to Computing

MCS1101B

Lecture 9

Recall Array

```
int states[5], i;                    //declaration
for (i=0; i<5; i++)
        states[i]=i;                 // initialization
for (i=0; i<5; i++)
        printf("%d, ", states[i]);   //access
        // outputs 1, 2, 3, 4, 5,
```

```
int a[5]; int *arr;
printf("%d, %d, %d \n", sizeof(int), sizeof(a[3]), sizeof(a));
        // outputs 4, 4, 20
printf (“%d”, sizeof (arr));              // outputs 8
arr = a;
printf (“%d”, sizeof (arr));              // outputs ??
```

- This is also called a one dimensional array or 1D array
- Sometimes we need to work on multidimensional data, e.g. 2D coordinates, matrix, system of equations
- 1D array is not convenient enough for such problems

2D Array

- int arr[m][n];
  - m is the number of rows and n is the number of columns
  - Array of m*n integers
  - Useful to store multidimensional data
- Accessing element at i^{th} row and j^{th} column using arr[i][j]
- Each arr[i] is an 1D array of size n

2D Array (contd.)

```
int a[2][3], i, j; //declaration

//initialization
for (i=0; i<2; i++)
    for (j=0; j<3; j++)
        a[i][j] = i + j;

//access
for (i=0; i<2; i++)
    for (j=0; j<3; j++)
        printf("%d, ", a[i][j]);

printf("%d, %d, %d", sizeof(a), sizeof(a[1]), sizeof(a[1][2]));
//outputs 24, 12, 4
```

3D Array

- int arr [m][n][p];
  - Array of m*n*p integers
- Each arr[i] is an 2D array of size n*p integers
- Each arr[i][j] is an 1D array of p integers
- Each arr[i][j][k] is a single integer element

→how to calculate the address of any element in the array?

3D Array (contd.)

```
int a[2][3][4], i, j, k; //declaration

//initialization
for (i=0; i<2; i++)
    for (j=0; j<3; j++)
        for (k=0; k<4; k++)
            a[i][j][k] = (i + j)*k;

//access
for (i=0; i<2; i++)
    for (j=0; j<3; j++)
        for (k=0; k<4; k++)
            printf("%d, ", a[i][j][k]);

printf("%d, %d, %d, %d \n", sizeof(a), sizeof(a[1]), sizeof(a[0][2]), sizeof(a[1][0][2]));
//outputs ??
```

Array of Pointers

- It’s an array of pointer variables
- Each element in the array can contain address of a variable of the declared type
- So, array of different sized arrays can be done

```
double *buf[3];

double d0 = 8, d1[2] = {11, 12}, d2 = 10;
buf[0] = &d0; buf[1] = &d1[0]; buf[2] = &d2;

printf("%d, %d \n", sizeof(void*), sizeof(buf));
//prints 8, 24

printf("%p, %p, %p, %lf \n", buf, &buf[0],
buf[1], *buf[1]);
//prints 0x..b200, 0x..b200, 0x..b1f0, 11.000000

 .. what will be buf[1][1] ?
```

Pointer to an Array

- It's a pointer that can point to a whole array
- It's has subtle difference from a normal array variable

```
double (*buf)[2]; ⇒just another pointer

printf("%d, %d, %d \n", sizeof(double),
sizeof(*buf), sizeof(*(buf+1)));
//prints 8, 16, 16

printf("%p, %p \n", buf, buf+1);
//prints 0x..7e30, 0x..7e40 ← garbage

→ buf[0][1] = 309; ⇒ this is illegal
```

Operations on strings

- Find the length of a string
- Compare two strings
- Concatenate two strings
- Change a string to uppercase
- Change a string to lowercase
- Duplicate strings
- Split strings into words
- Split strings based on a given delimiter

- Array of strings

```
char arr[3][10] = {"IACS", "UG", "2022"};
```

- Array of pointers to strings

```
char *arr[] = {"IACS", "UG", "2022"};
```

#include<string.h>

```
char str1[20]="A string", str2[20]="Another string";

strlen (str1)                 // gives the length of the string ⇒ 8
strcpy (str2,str1)            // copies str1 into str2
strncpy (str2, str1, n)       // copies first n characters from str1 into str2
strcmp (str1, str2)           // returns 0 if both strings are the same
strcmpi (str1, str2)          // compares two strings ignoring the case
strcat (str1, str2)           // concatenates str2 at the end of str1
strchr (str1, 'r')            //finds the position(pointer) of first 'r' in str1
```

Link to more all string.h functions with examples.

#include<math.h>

Some Functions:

```
double sqrt (double);
double exp(double);
double log(4.0));
double log10(100.0));
double fabs(double);
int ceil(double);
int floor(double);
double pow(double, double);
double fmod(double, double);
double sin(double);
double cos(double);
double tan(double);
```

Some Constants:

M_PI, M_PI_2, M_PI_4

M_1_PI, M_2_PI

M_E, M_LOG2E, M_LOG10E

M_LN2, M_LN10

Link to the full List of constants from math.h

Command Line Arguments (CLA)

- Compile the code ⇒
  - It will generate a.exe file

Run the code as follows:

.\a.exe Hello

argument supplied is Hello

.\a.exe Hello Hi

Too many arguments.

.\a.exe

One argument expected.

```
int main( int argc, char *argv[] )
{
    if( argc == 2 )
      printf ("argument supplied is %s\n", argv[1])

    else if ( argc > 2 )
      printf ("Too many arguments.\n");

    else
      printf ("One argument expected.\n");
}
```

Recall File

- **FILE*** is a datatype used to represent a pointer to a file
- To open a file we use a function called **fopen**
  - It takes two parameters
    - Name of the file
    - Mode in which it is to be opened
  - It returns a pointer to the file if the file is opened successfully, otherwise it returns NULL

Example of a file creation for writing

```
FILE *fp;
char filename[] = “a_file.dat”
fp = fopen (filename, “w”);
if (fp != NULL)
{
    /* WRITE SOMETHING IN FILE */
fclose (fp);
}
```

File operations

- **fputc**
- **fputs**
- **fprintf**
- **fflush**
- fgetc
- fgets
- fscanf
- feof
- ungetc

```
FILE *fp = fopen("abc.txt", "w");

if (fp != NULL) {
        fputc('a', fp);
        fputs("cde", fp);
        fprintf(fp, "%d, %c, %s", 25, 'I', "hello");
        fflush(fp);
        fclose(fp);
}
```

File operations (contd)

- fputc
- fputs
- fprintf
- fflush
- **fgetc**
- **fgets**
- **fscanf**
- feof
- ungetc

```
FILE *fp = fopen("abc.txt", "r");
char buf[10];     int num;     char c;
if (fp != NULL) {
        c = fgetc(fp);            // printf (“%c”, c);
        fgets(buf, 4, fp);        // printf (“%s", buf);
        fscanf(fp, "%d, %c, %s", &num, &c, buf);
        printf (“%d %c %s”, num, c, buf);
        fclose(fp);
}
```

File operations (contd)

- fputc
- fputs
- fprintf
- fflush
- fgetc
- fgets
- fscanf
- **feof**
- **ungetc**

```
char c, buf[256];
FILE *fp = fopen("abc.txt", "r");
if (fp != NULL) {
while (!feof(fp)) {
        c = fgetc(fp);
        if (c == ‘a’)
                ungetc(‘b’, fp);
        fgets(buf, 255, fp);
        printf("%s", buf);
} }
```

That’s all.

- Questions?