Module Code: CSE401

Session 07: Strings and User Defined Data Types

Session Speaker:

Prema Monish
premamonish.tlll@msruas.ac.in

Session Objective

- At the end of this lecture, student will be able to
  - Create String Variable
  - Use of string handling functions
  - create user defined data types
  - use ***typedef*** and ***struct*** keywords in C programming language
  - apply pointers to structures
  - explain self-referential structures and their applications
  - Use unions and enums
  - Use of bitfields

Session Topic

- String declaration and initialization
- String handling functions
- Structure definition
- Structure variable declarations
- Pointers to structures
- Defining data types with typedef
- Union
- Bit fields

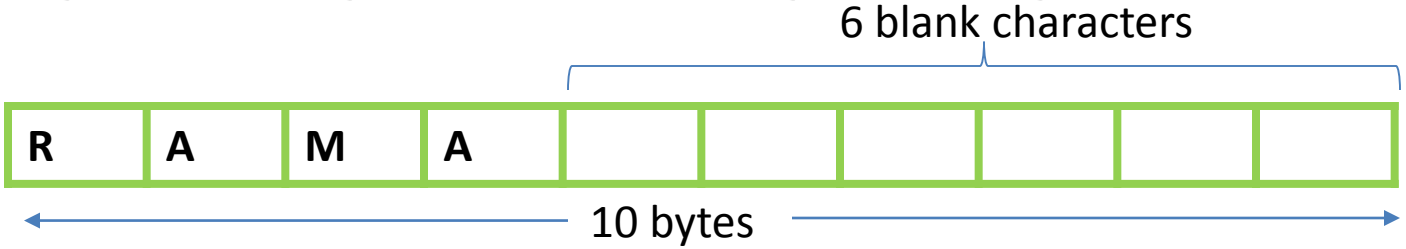
Strings

- A string is a combination of *characters* or a word enclosed in ” “.
- E.g. printf("This is a string.");

  printf("This is on\n two lines!");
- Storage representation of strings :
  1. Fixed length string
  2. Variable length string
     - Length controlled string
     - Delimited string

Fixed length string

- Length of string is always *fixed*
- Is a string whose storage requirement is known when the string is created
- Length of string cannot be changed during execution



E.g. String "RAMA " is stored in fixed format where size is fixed to 10 bytes

Disadvantage

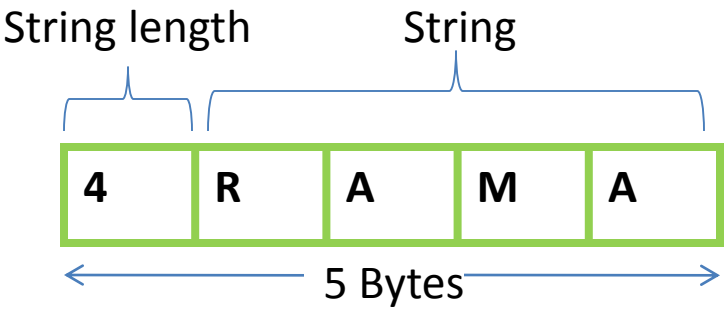
- If length reserved is too *small*, it is not possible to store large data
- If length reserved is too *large*, too much memory reserved

Variable length String

- Length of variable is *not fixed* nor predefined
- Storage structure for a string can expand or shrink to accommodate any size of data
- But, should be a mechanism to indicate the end of data
- Two common techniques to implement variable length string
    1. Length controlled string
    2. Delimited string

Length controlled string

- Is a string whose length is stored as part as string itself
- count that specified string length is normally stored as the first byte followed by string

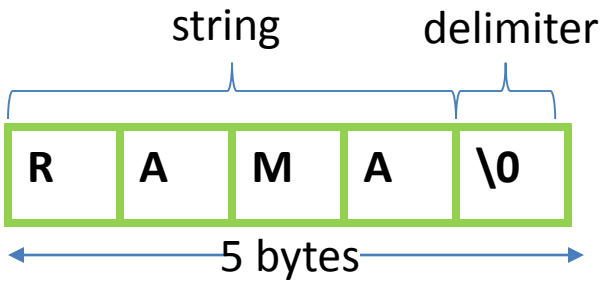


Disadvantage

- Since count length is 1 Byte so we can have a string length whose range is from 0 to 255
- So string length should not exceed 255

Delimited String

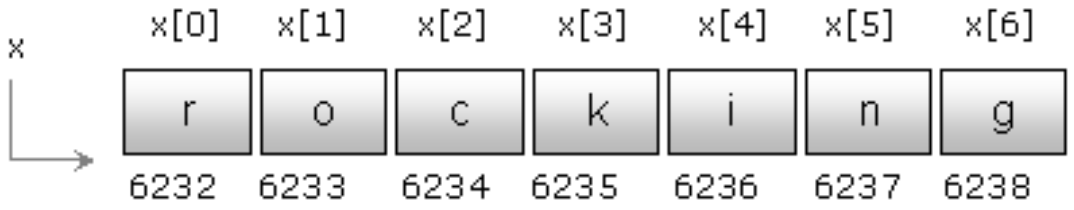
- In a variable length string ,the string ends with a *delimiter* or NULL character *'\0'*



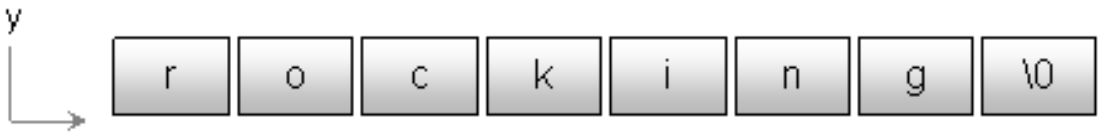
- Disadvantage of length controlled string is over come here

Character Vs. String

```
1. char x[]={'r','o','c','k','i','n','g'};      /* character array  */
```




```
2. char y[]="rocking";                          /* string */
```



Character Vs. String

Program:

```
#include <stdio.h>
int main()
{
char x[]={'r','o','c','k','i','n','g'};
char y[]="rocking";
printf("Size of character array %d",sizeof(x));
printf("\nSize of string %d",sizeof(y));
return 0;
}
```

Output:

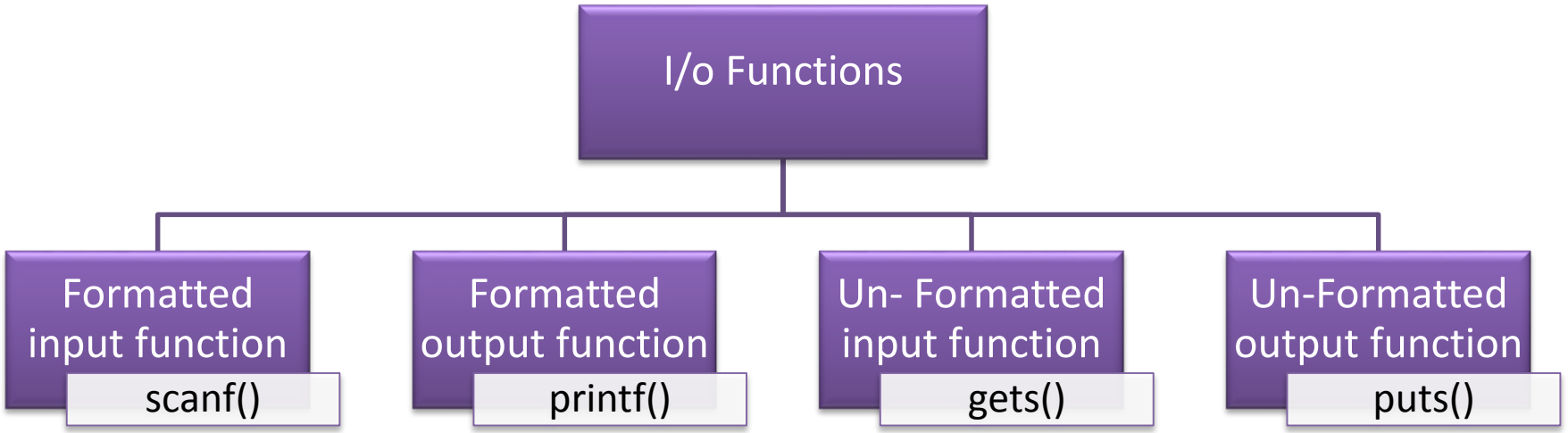
Size of character array 7

Size of string 8

- With the output of above program, it is confirmed that a string has an additional character.

String I/O Functions

- The various input and output functions that are associated with strings can be classified as follows



Formatted Input Function

- scanf() is the formatted input function
- scanf() uses format specifier '%s to read strings

```
int main()
{
char x[8];
int i;
printf("Enter any string:");
scanf("%s",x);
printf("The given string:");
for(i=0;x[i]!='\0';i++)
printf("%c",x[i]);
return 0;
}
```

input – RAMA ↵

output- RAMA

R	A	M	A	\0	?	?	?

Formatted output function

- printf() is the formatted output function
- printf() uses format specifier '%s' to write strings

```
int main()
{
char x[8];
int i;
printf("Enter any string:");
scanf("%s",x);
printf("The given string:");
printf("%s",x);
return 0;
}
```

Output:

Enter any string

Hello world

The given string :

Hello

Un-Formatted output function

- Function a string of characters including the white spaces.
- All characters till the '\0' character is displayed
  - puts(str);

```
#include<stdio.h>
int main()
{
char x[]="rocking";
puts(x);
puts("Hello world");
return 0;
}
```

Output:
rocking
Hello world

Un-formatted input function

- Function accepts string from keyboard .
- The string entered may include white spaces .

```
#include<stdio.h>
int main()
{
char x[100];
printf("Enter a line of text:");
gets(x);
printf("The given text is : %s",x);
return 0;
}
```

Output:
Enter a line of text: hello world
The given text is: hello world

String handling functions

- The Various string handling functions that in C Language are show below
- All this function is defined in string.h

String handling functions

Functions	Description
strlen(s1)	Returns the length of a string s1
strcpy(s1,s2)	Copy a string s2 into string s1
strncpy(s1,s2)	Copies character of a string s2 to another string s1 up to the specified length n.
strcmp(s1,s2)	Compare two string s1 and s2 (Function discriminates between upper case and lower case) Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2
stricmp(s1,s2)	Compare two string s1 and s2 (Function doesn't discriminates between upper case and lower case) Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2
strncmp(s1,s2,n)	Compares character of two string s1 and s2 upto a specified length.
strnicmp(s1,s2,n)	Compares characters of two string s1 and s2 upto a specified length. Ignore case.
strlwr(s1)	Converts the uppercase character of string s1 to lowercase.
strupr(s1)	Converts the lowercase character of string s1 to uppercase.
strdup(s1)	This function duplicates the string s1 at the allocated memory which is pointed by a pointer variable.
strchr(s1,c)	Returns a pointer to the first occurrence of character c in string s1.
strrchr(s1,c)	Returns a pointer to the last occurrence of character c in string s1.
strcat(s1,s2)	Concatenates string s2 onto the end of string s1.
strncat(s1,s2,n)	Concates string s1and s2 upto a specified length n.
strrev(s1)	Reverse all the characters of string s1.
strstr(s1, s2);	Returns a pointer to the first occurrence of string s2 in string s1.

String Handling Functions

- strlen() is a predefined function in the header file “string.h” used to find the length of string.
- It accepts the string (address of array) as argument and returns the length excluding with ‘\0’.

```
#include<stdio.h>
#include<string.h>
int main()
{
char x[50];
int l;
printf("Enter any string:");
scanf("%s",x);
l=strlen(x);
printf("The length of string %d",l);
return 0;
}
```

Output:
Enter any string: MSRUAS
The length of string 6

String handling functions

- strrev() is a predefined function defined within the header file “string.h” used to reverse a string.
- It accepts the address of a string as an argument and reverses the string.

```
#include<stdio.h>
#include<string.h>
int main()
{
char x[50];
int i;
printf("Enter any string:");
scanf("%s",x);
strrev(x);
printf("The reverse string %s",x);
printf("%s",x);
return 0;
}
```

Output:
Enter any string: MSRUAS
The reverse string SAURSM

String handling functions

- strcpy() is a predefined function defined with in the header file “string.h” used to copy a string onto another.
- It accepts the addresses of both target and source strings as arguments and copy the contents of Source string on to target string.

```
#include<stdio.h>
#include<string.h>
int main()
{
char x[50],y[50];
printf("Enter any string:");
scanf("%s",x);
strcpy(y,x);  /* "x" is copied onto "y" */
printf("%s",y);
return 0;
}
```

output

```
Enter any string: MSRUAS
MSRUAS
```

String handling functions

- strcat() is a predefined function defined within the header file “string.h” used to concatenate a string to another.
- It accepts the addresses of target and source strings as arguments and concatenates the source string to the target string.

```
#include
#include
int main()
{
char x[50],y[50];
printf("Enter the 1st string:");
scanf("%s",x);
printf("Enter the 2nd string:");
scanf("%s",y);
strcat(x,y);
printf("The resultant string is %s",x);
return 0;
}
```

output

```
Enter the 1st string: hello
Enter the 2nd string: World
The resultant string is helloWorld
```

String handling functions

- strcmp() is a predefined function defined within the header file “string.h”
- It accepts two strings as arguments and returns the ASCII difference among the first occurrence of unequal characters.
- It returns zero if both the strings are equal.

```
#include
#include
int main()
{
char x[50],y[50];
printf("Enter the first string:");
scanf("%s",x);
printf("Enter the second string:");
scanf("%s",y);
if(strcmp(x,y)==0)
  printf("Equal");
else if(strcmp(x,y)>0)
  printf("Biggest string %s",x);
else
  printf("Biggest string %s",y);
return 0;
}
```

Execution:
Enter the first string: hello
Enter the second string: world
Biggest string world

typedef Keyword

- **typedef** is a keyword used in C language to assign alternative names to existing data types
  - Syntax: typedef <existing Data type> <Alias name>
  - E.g typedef int number;

    number a,b,c;

    number add(int,int);

Need for User defined data type

- All students have same characteristics
  - Name
  - Roll number
  - Age
  - class

- we nearly need 40 variables to handle the details of 10students. It is cumbersome job to declare huge number of variables, writing input and output statements for them.
- Even it is so difficult to send the details of a student to a function .

User Defined data type

- C language provides no predefined data type to *store multiple values of different types* belongs to a single entity in a single variable.
- C language allows to *define our own data type* according to our requirement. Say for example, we can define our own data type called “student” to store the details of a student in a single variable.
  1. Structure
  2. Union
  3. Enum
- E.g.

```
struct student
{
      int pcode;
      char pname[20];
      float pprice;
      float tax;
      float sprice;
};
```

Structure

- Collections of related variables under one name
  - Can contain variables of different data types
- Commonly used to define records to be stored in files
- Combined with pointers, can create data structures such as linked lists, stacks, queues, and trees
- Size of structure is sum of size of all members

Structure Definition

```
struct student{
            char name[10];
            char rollNum[10];
             int age;
        };
```

- Keyword struct is used to declare the structure.
- All the members are declared with in a block { }
- **student** is structure name
- The definition must be terminated with a semi-colon (;)
- Defining the structure **doesn't allocate the memory allocation** of any member
- Generally struct is defined at the top of all the functions and bellow #include statements. Some times defined in a separate header file to include it into any program
- **student** contains two members of type **char - name** and **rollNum** and a member of type **int – age**

Structure Declaration

- Declared like other variables

```
struct student myStudent;
struct student students[60]; //Array of structures
```

- Variables can be declared as part of the structure definition

```
struct student{
        char name[10];
        char rollNum[10];
        int age;
        } myStudent, students[60];
```

Assigning a struct variable

- Initializer lists

```
student myStudent = {“sarma", “CJB0912332“, 31};
```

- Assignment statements

```
student mystu= myStudent;
    Or
student mystu;
mystu.name = “Sarma”;
mystu.rollNo= “CJB0912332”;
mystu.age = 31;
```

Accessing Members of Structures

- Accessing structure members
  - Dot operator (.) - use with structure variable name

```
student myStudent;
printf( "%s", myStudent.age );
```

Array of struct type

```
struct student{
            char name[10];
            char rollNum[10];
             int age;
       };
```

- Above struct is capable of holding details of single student so for n students array can be declared
  - `struct student stu [3];`
    - Initialization of a struct array:

```
struct student stu[3]={{“manu”,”1rc07001”,10},
                       {“anu”,”1rc07002”,10},
                       {“prema”,”1rc07003”,10}};
                              or
                      Stu[0].name=manu
                      Stu[0].rollNum=1rc07001
                      Stu[0].age=10
```

Structure Definition with *typedef*

- To avoid repeating ‘struct’ every time a variable is declared, use typedef

```
typedef struct {
      char name[10];
      char rollNum[10];
      int age;
  } STUDENT;
STUDENT myStudent, students[60];
```

Pointer to a structure

- Consider the following declaration:

```
struct  inventory{
        char name[30];
        float price;
} product[2], *ptr;
```

- The assignment

  ptr = product;

  would assign the address of the zeroth element of product to ptr
- Its members can be accessed using the following notation

  ptr -> name, ptr->price
- Initially the pointer ptr will point to product[0]
- when the pointer ptr is incremented by one it will point to next record, that is product[1]

Pointer to a structure

- We can also use the notation

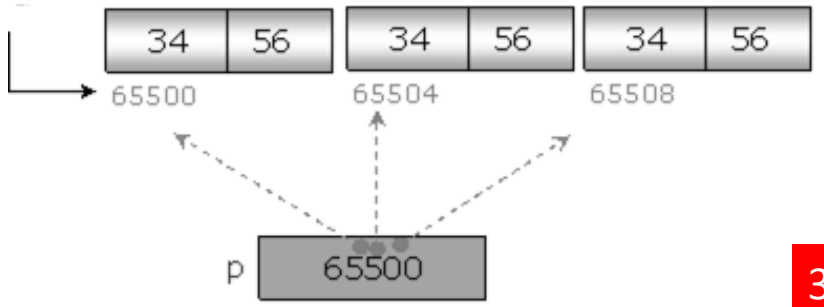
  (*p).b or p->b

  to access the member number
- P++;
  - Points to next element

```
struct box
{
 int b;
 int w;
 int h;
};
Void main()
{
struct box b={10,20,30};
struct box *p;
p=&b;
printf("Three sides of box: %d\t%d\t%d",(*p).b, p->w,(*p).h);
}
```

Pointer representation to struct array

```
struct point
{
 int x;
 int y;
};
 struct point a[]={{34,56},{22,12},{45,32}};
struct point *p;
p=a;
for(i=1;i<=3;i++)
 {
  printf("x=%d\ty=%d\n",p->x,p->y);
  p++;  /* visiting next element of array */
 }
```



Functions and struct

- Structures can be passed as arguments to functions
  - Entire structure or members of the structure

- Passing structures to functions
  - call by value
    - Pass entire structure
    - Or, pass individual members
  - call by reference
    - Pass structure variable's address

Functions and struct

Call By Value

```
struct book
{
 char name[50];
 char author[50];
 char publisher[50];
 float price;
};
void display(struct book);
int main()
{
 struct book x={"Let us C","Kanithkar","BpB",275};
 display(x);
 return 0;
}
void display(struct book a)
{
 printf("Book Name:%s",a.name);
 printf("\nAuthor: %s",a.author);
 printf("\nPublisher: %s",a.publisher);
 printf("\nPrice:%f",a.price);
}
```

Call By reference

```
#include<stdio.h>
#define PI 3.14
struct circle
{
 int rad;
 float area,cir;
};
void calculate(struct circle*);
int main()
{
 struct circle x;
 printf("Enter the radios:");
 scanf("%d",&x.rad);
 calculate(&x);  //sending the address
 printf("Area of circle %f",x.area);
 printf("\nCircumference of circle
%f",x.cir);
 return 0;
}
void calculate(struct circle *p)
{
 p->area=PI*(p->rad)*(p->rad);
 p->cir=2*PI*(p->rad);
}
```

Returning a structure

Returning a struct variable

```
#include<stdio.h>
#include<math.h>
struct point
{
 int x;
 int y;
};
struct point getpoint();
int main()
{
 struct point p;
 p=getpoint();
 printf("x=%d\ny=%d",p.x,p.y);
 return 0;
}
struct point getpoint()
{
 struct point a={43,56};
 return a;//returning the struct variable
}
```

Returning the address of a struct variable

```
#include<stdio.h>
#include<math.h>
struct point
{
 int x;
 int y;
};
struct point* getref();
int main()
{
 struct point *p;
 p=getref();
 printf("x=%d\ny=%d",p->x,p->y);
 return 0;
}
struct point* getref()
{
 struct point a={43,56};
 return &a;//returning the address
}
```

Unions

- The structure and the definition of union are similar but, differ in functionality.
- The size of union variable is size of member with the maximum length.
- All operations on ***Structures*** are valid on ***Unions***

```
union demo
{
 float x;
 int y;
 char ch;
};

union demo var;
```

Unions

- Predict and analyze the output?

```
#include<stdio.h>
struct demo1
{
      float x;
      int y;
      char ch;
};
union demo2
{
      float x;
      int y;
      char ch;
};
int main()
{
      struct demo1 var1;
      union demo2 var2;
      printf("Size of struct variable is %d bytes\n",sizeof(var1));
      printf("Size of union variable is %d bytes\n",sizeof(var2));
      return 0;
}
```

Enum

- It is a user defined type used to define integer type symbolic constants.
- the values of which are automatically initialized by the compiler.
- The value of first symbolic constant would be assigned with 0, second would be 1 and third would be with 2 and so on..
- enum is keyword used to define enum data type
  - syntax

```
enum name{ Set of symbolic consts}
```

Enum

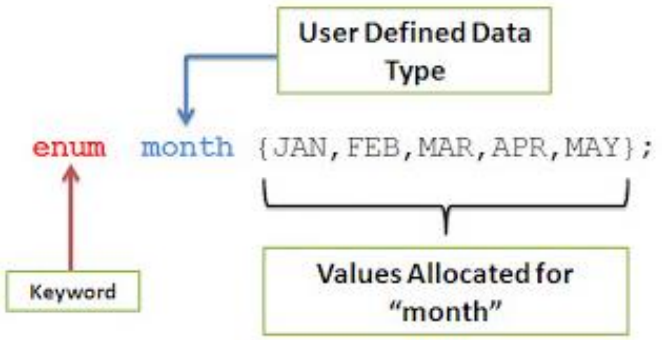
- Declare variables as normal
  - Enumeration variables can *only* assume their enumeration constant values not int values

enum DAYS {SUN, MON, TUE, WED, THU, FRI, SAT};

- Starts at **0**, increments by **1**

enum DAYS {SUN= 1, MON, TUE, WED, THU, FRI, SAT};

- Starts at **1**, increments by **1**



Bit fields in C language

- Member of a structure whose width(in bits) has been specified
- Enable better memory utilization
- Only for ***int*** or ***unsigned***
- Declaring bit fields - Example

```
struct date
{
 int dt;
 int mn;
 int yr;
};

struct date a;
```

Here struct date type of variable "a" takes 6 bytes that is 48 bits,

```
struct date
{
 int dt:6;
 int mn:5;
 int yr:12;
};
```

Here the member "dt" takes 6 bits : -32 to +31,
"mn" takes 5 bits : -16 to +15
"yr" takes 12 bits : -2048 to +2047.

Summary

- String definition and declaration
- String handling functions
- Structures are constructs used when related data of different type must be handled in a program
- Structures can be defined using ***struct*** keyword
- ***typedef*** keyword allows a structure to be used like normal data types, without the ***struct*** prefix
- ***dot operator*** is used to access members of a structure variable
- ***sizeof*** a structure variable is at least the sum of sizes of all variables
- Unions are similar to structures and are used when one memory is used to store multiple types of values
- Enum user defined data type
- Member of a structure whose width(in bits) has been specified using bitfields.