C PROGRAMMING FOR PROBLEM SOLVING (18CPS23)

Module 5
Structure and Pointers
Pre-processor directives

Prof. Prabhakara B. K.
Associate Professor
Department of Computer Science and Engineering
Vivekananda College of Engineering and Technology, Puttur.

STRUCTURE

(User Defined Datatype)

What is Structure?

It is collection of related variables of same or different datatype.

OR

It is a mechanism for packing variables of same or different datatypes.

OR

Fixed size sequenced collection of elements of same or different data type.

- It is a user defined datatype.
- struct keyword is used to define a structure.

DECLARING A STRUCTURE
(Defining a Structure)

Syntax

```
struct tag_name
{
    datatype member 1;
    datatype member 2;
        :
    datatype member n;
};
```

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
};
```

Memory is not allocated when we define or declare a structure

DECLARING STRUCTURE VARIABLES

Syntax

```
struct tag_name
{
   datatype member 1;
   datatype member 2 ;
      :
   datatype member n;
};
struct tag_name variable1,variable2,...;
```

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
};
struct student s1,s2;
```

Memory is allocated when we declare structure variables

DECLARING STRUCTURE VARIABLES

Syntax

```
struct tag_name
{
    datatype member 1;
    datatype member 2;
                :
    datatype member n;
} variable1,variable2,...;
```

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
} s1,s2;
```

Memory is allocated when we declare structure variables

ACCESSING STRUCTURE MEMBERS

Syntax

```
structurevariable.member
```

dot operator

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
};
struct student s1,s2;
```

Accessing members

s1.rollno

s1.name

s1.marks

s2.rollno

s2.name

s2.marks

ARRAY OF STRUCTURES

Syntax

```
struct tag_name
{
   datatype member 1;
   datatype member 2 ;
      :
   datatype member n;
};
struct tag_name arrayname[size];
```

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
};
struct student s[100];
```


By: Prof. Prabhakara B K, VCET Puttur 9844526585

ARRAY OF STRUCTURES

Syntax

```
struct tag_name
{
    datatype member 1;
    datatype member 2 ;
        :
    datatype member n;
} arrayname[size];
```

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
} s[100];
```

ACCESSING MEMBERS OF STRUCTURE ARRAY

Syntax

```
structurevariable[subscript].member
```

dot operator

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
};
struct student s[100];
```

OR

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
} s[100];
```

Accessing members

```
s[0].rollno
s[0] .name
s[0].marks
   .
s[99].rollno
s[99].name
s[99].marks
```

Write a C program to read name, usn and marks of N students using structure. Also search a given student name to display marks.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct student
{
    int usn, marks;
    char name[30];
};
void main()
{
   int i, n;
   char sname[30];
   struct student s[100];
   printf("Enter the number of students \n");
   scanf ("%d" , &n);
   for (i=0; i<n; i++)
   {
      printf("Enter the USN:");
      scanf ("%d", &s[i].usn);
      printf ("Enter the name of student:");
      scanf ("%s", s[i]. name);
      printf ("Enter the marks:");
      scanf ("%d", &s[i]. marks);
   }
```

Write a C program to read name, usn and marks of N students using structure. Also search a given student name to display marks.

```
    printf ("Enter the name of student to be searched for marks\n");
    scanf ("%s", sname);
    for(i=0; i<n; i++)
    {
        if (strcmp(sname, s[i].name) = = 0)
      {
        printf("%s is scored %d marks", sname, s[i].marks);
        exit(0);
      }
    }
    printf ("Student is not found");
}
```

13. Implement structures to read, write, compute average-marks and the students scoring above and below the average marks for a class of N students.

```
#include<stdio.h>
struct student
{
    int usn, marks;
    char name[20];
}s[20];
void main()
{
    int i, n;
    float total=0,avg;
    printf("Enter the number of student\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the %d student details\n",i+1);
        printf("Enter the USN:");
        scanf("%d",&s[i].usn);
        printf("Enter the student name without white spaces:");
        scanf("%s",s[i].name);
        printf("Enter the  marks:");
        scanf("%d",&s[i].marks);
        total=total+s[i].marks;
}
```

student

usn	name	marks
101	Smith	50
105	Jhon	60
107	Rajesh	70
total		180
avg=total/3		60

13. Implement structures to read, write, compute average-marks and the students scoring above and below the average marks for a class of N students.

```
        avg=total/n;
    printf("Students scored above the average marks\nUSN\tNAME\tMARKS\n");
    for(i=0;i<n;i++)
    {
       if(s[i].marks>=avg)
       {
            printf("%d\t%s\t%d\n", s[i].usn,s[i].name,s[i].marks);
       }
    }
    printf("Students scored below the average marks\nUSN\tNAME\tMARKS\n");
    for(i=0;i<n;i++)
    {
        if(s[i].marks<avg)
       {
            printf("%d\t%s\t%d\n", s[i].usn,s[i].name,s[i].marks);
        }
      }
}
```

INITIALIZATION OF STRUCTURE

Syntax

```
struct tag_name variable = { list of values };
```

Example

```
struct student
{
        int rollno ;
        char name[30];
        float marks ;
};
struct student s1={ 123, "John", 92.4};
```

OR

Example

```
struct student
{
        int rollno ;
        char name[30];
        float marks ;
} s1={ 123, "John", 92.4};
```

ASSIGNING STRUCTURE VARIABLE

Syntax

```
struct tag_name
{
    datatype member 1;
        datatype member 2;
                :
    datatype member n;
} variable1,variable2={list of elements};
variable1= variable2;
```

Example

```
struct student
{
        int rollno;
        char name[30];
        float marks;
} s1, s2 = { 123, "John", 92.4};
s1=s2;
```

NESTED STRUCTURES

- A structure present inside another structure
- A structure is a member of another structure

Syntax

```
struct tag_name1
{
    datatype member 1;
            :
    datatype member n;
};
struct tag_name2
{
    datatype member 1;
            :
    struct tag_name1 variable;
};
```

Example

```
struct name
{
   char first[30];
   char last[30];
;
struct student
{
   int age;
   struct name sname;
;
struct student s;
```

ACCESSING NESTED STRUCTURES

Example

```
struct name
{
   char first[30];
   char last[30];
;
struct student
{
   int age;
   struct name sname;
;
struct student s;
```

name is a structure inside student structure.

To access members of name structure

```
sname.first
        }
sname.last
```

To access members of name structure & student structure

```
s.age
                }
s.sname.first
s.sname.last
```

INITIALIZATION OF NESTED STRUCTURES

Example

```
struct name
{
    char first[30];
    char last[30];
 };
struct student
 {
     int age;
     struct name sname;
};
struct student s = { 42, {"John", "Smith"} };
```

typedef

Used to form complex or similar data type from the basic data types.

typedef is a keyword

Syntax

```
typedef  oldname  newname;
```

Consider below declaration

```
int a, b;
```

Above declaration by using typedef

```
typedef  int  cps;
cps a, b;   // a and b variables are of int data type
```

where cps is alternative name for int.

typedef IN STRUCTURES

Syntax

```
typedef struct
{
    datatype member 1;
        :
      datatype member n;
} tag_name;
tag_name variable1,variable2,...;
```

Example

```
typedef struct
{
        int rollno;
       char name[30];
        float marks;
} student;
student s1, s2;
```

While declaring the variable for structure, the keyword struct is not required.

typedef IN STRUCTURES

Syntax

```
typedef struct old_tag_name
{
    datatype member 1;
       :
      datatype member n;
} new_tag_name;
new_tag_name variable1,variable2,...;
```

Example

```
typedef struct stud
{
        int rollno;
       char name[30];
        float marks;
} student;
student s1, s2;
```

While declaring the variable for structure, the keyword struct is not required.

STRUCTURES AND FUNCTIONS

Types of parameter passing in functions based on structures.

1. Passing individual members of a structure to a function.

2. Passing entire structure to a function.

3. Passing structure as a pointer to a function.

1. PASSING INDIVIDUAL MEMBERS OF A STRUCTURES

Program to add two integers using structure and function

```
#include<stdio.h>
struct addition
{
        int a, b;
};
void add(int a, int b)
{
        int c;
        c = a + b;
        printf ("Sum =%d", c);
}
```

```
void main( )
{
        struct addition p;
        printf("Enter two integers\n");
        scanf("%d%d", &p.a, &p.b );
        add(p.a, p.b );
}
```

2. PASSING ENTIRE STRUCTURE TO A FUNCTION

Program to add two integers using structure and function

```
#include<stdio.h>
struct addition
{
    int a, b;
};
void add (struct addition p)
{
    int c;
    c = p.a + p.b;
    printf("Sum=%d", c);
}
```

```
void main()
{
    struct addition p;
    printf("Enter two integers\n");
    scanf("%d%d",&p.a, &p.b);
    add(p);
}
```

3. PASSING STRUCTURE AS A POINTER TO A FUNCTION

Program to add two integers using structure and function

```
#include<stdio.h>
struct addition
{
        int a, b;
};
void add(int *a, int *b)
{
        int c;
        c= *a + *b;
        printf("Sum=%d", c);
}
```

```
void main()
{
        struct addition p,*x;
        x=&p;
        printf("Enter two integers\n");
        scanf("%d%d",&x->a,&x->b);
        add(&x->a, &x->b);
}
```

ARRAY	STRUCTURE
It is derived data type.	It is user defined data type.
The individual entries are called elements.	The individual entries are called members.
The elements are of same data type.	The members are of same or different data types.
It is a pointer to its first element.	It is not pointer.
To access an element requires less time compared with structure.	To access a member requires more time compared with an array.
Index/subscript is used for accessing an element.	.(dot) and ->(pointer) operators are used for accessing the members.
It allocates static memory.	It allocates dynamic memory.

POINTERS

Pointer is a variable that holds address of another variable.

Uses of Pointers

1. Pointers provide direct access to memory.
2. Pointers provide a way to return more than one value to the functions.
3. Reduces the storage space during execution of a program.
4. Reduces the execution time of a program.
5. Provides an alternate way to access array elements.
6. Pointers provide platform to do communication between the calling function and called function.
7. Pointers allow us to perform dynamic memory allocation and de-allocation.
8. Pointers helps us to build complex data structures like linked list, stack, queues, trees, graphs etc.

DECLARATION OF POINTERS

Syntax

```
datatype *pointer;
```

Example

```
int *p;
```

* is dereference operator or indirection operator.

In above example p is a pointer variable of type integer then

*p can hold address of some other integer variable.

ASSIGNING / INITIALIZATION OF POINTERS

When a pointer is declared, it points to the garbage value. Such a pointer is called stray pointer. Hence it is required to be initialized.

Syntax

```
datatype variable;
datatype *pointer;
pointer = &variable;
```

OR

```
datatype variable;
datatype *pointer = &variable;
```

Example

```
int a = 10;
int *p;
p = &a; // initializing pointer
```

OR

```
int  a = 10;
int *p = &a;  //declaring and initializing pointer
```

POINTERS

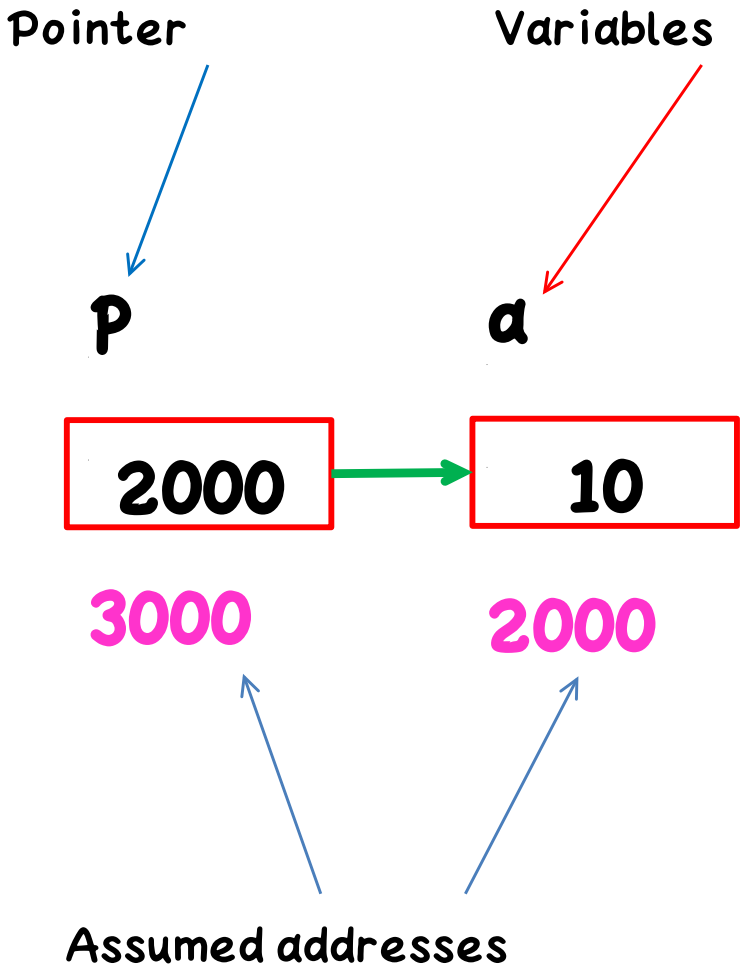
Pointer is a variable which holds address of another variable.

Example

```
int a = 10;
int *p;
p = &a;
```

OR

```
int a = 10;
int *p = &a;
```



POINTERS

```
#include<stdio.h>
void main()
{
    int a=10;
    int *p;
    p=&a;
    printf("The value of a = %d\n", a);
    printf("The address of a = %d\n", &a);
    printf("The value of p = %d\n", p);
    printf("The value of *p = %d", *p);
}
```

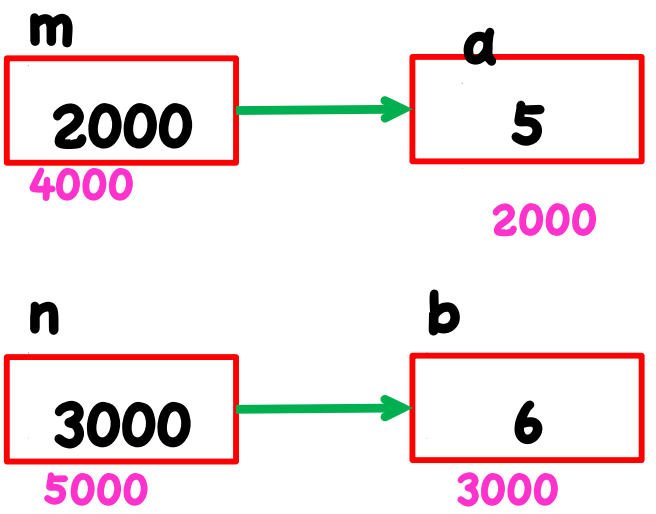
Output

```
The value of a = 10
The address of a = -2025589956
The value of p = -2025589956
The value of *p = 10
```

POINTERS AND FUNCTIONS

//Program to swap two integers using pointers (Call by reference)

```
#include<stdio.h>
void swap(int *m, int *n)
{
      int p;
      p=*m;
      *m=*n;
      *n=p;
}
void main()
{
       int a, b;
       printf("Enter two integers\n");
       scanf("%d%d",&a, &b);
       swap(&a, &b);
       printf("a=%d\tb=%d", a, b);
}
```



a=5	b=6
&a=2000	&b=3000
m=2000	n=3000
&m=4000	&n=5000
*m=5	*n=6

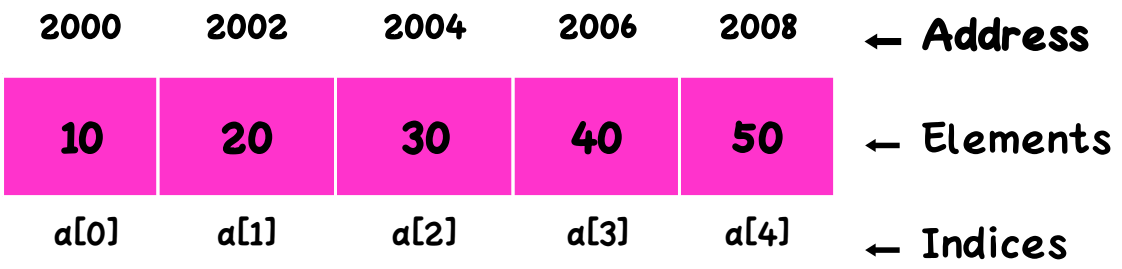
POINTERS AND ARRAYS

Syntax

```
datatype arrayname[size];

datatype *pointer;

pointer = arrayname;
```

Example

```
int a[5];

int *p;

p = a;
```



The array name holds the starting address of an array.

In above example we have assumed the starting address a=2000

Array name is a pointer, which points to first element of that array.

This address is known as Base Address.

LOCATING AN ARRAY ELEMENTS

The address of an array element is calculated using below formula

Memory location of an array element = Base Address + (Index * Size)

int a[5]={10,20,30,40,50};

assume base address is 2000

a[0]	10	= 2000+(0*2) = 2000
a[1]	20	= 2000+(1*2) = 2002
a[2]	30	= 2000+(2*2) = 2004
a[3]	40	= 2000+(3*2) = 2006
a[4]	50	= 2000+(4*2) = 2008

2000	2002	2004	2006	2008
10	20	30	40	50
a[0]	a[1]	a[2]	a[3]	a[4]

ARRAYS OF POINTERS

With the help of arrays user can declare collection of pointers.

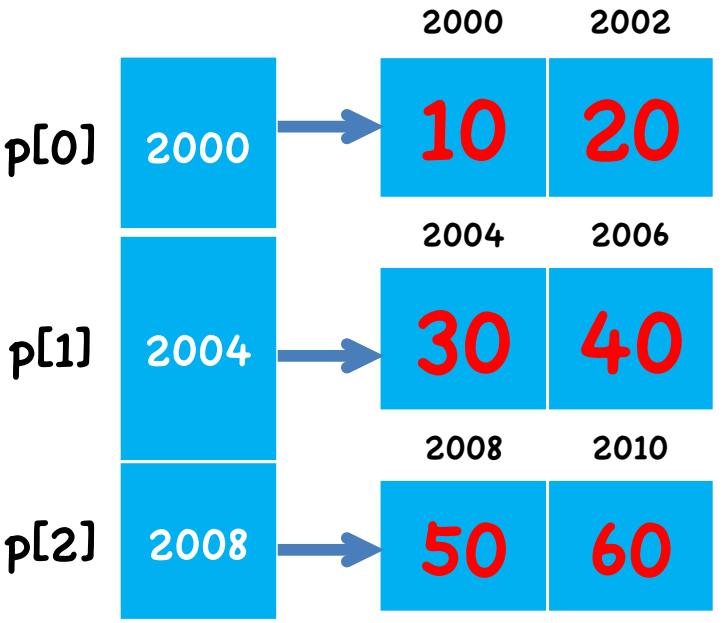
These pointers can have same name but different subscripts.

Declaration

```
datatype *arrayname[size];
```

Example

```
int *p[3];
int i, a[3][2]={ {10,20}, {30,40}, {50,60} };
for (i=0; i<3; i++)
{
   p[i] = &a[i][0];
}
```



Above code segment creates 3 pointers p[0], p[1] and p[2].

If we assume base address of array is 2000, then

p[0] is pointing to base address. i.e., p[0] is pointing to first row first element.

p[1] is pointing to second row first element.

p[2] is pointing to third row first element.

ADDRESS ARITHMETIC
(POINTER ARITHMETIC)

It is a method of calculating the address of an object with the help of arithmetic operations on pointers .

Incrementing or decrementing the address pointed by pointer based upon memory size of data to which it is pointing.

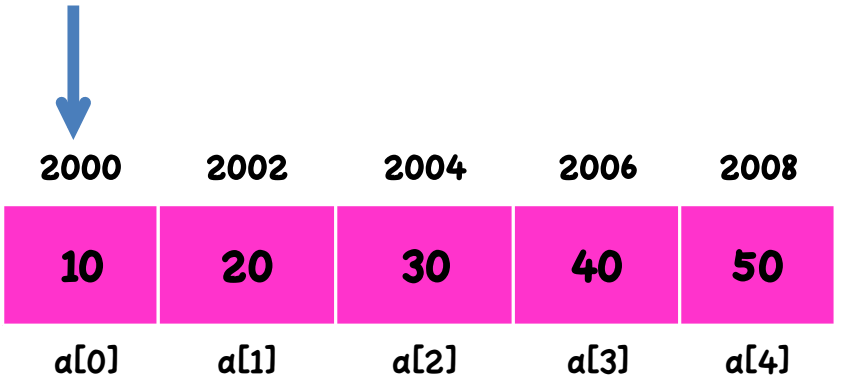
Example

```
int a[10];
int *p;
p=a;
p++;   // Here pointer is incremented by 2 i.e., p=p+2.
```

In above example, p is pointing to array is of type integer consumes 2 bytes that's why p++ means p=p+2.

Write a C program to find sum of n integers using pointer.

```
#include<stdio.h>
void main()
{
    int i, n, sum=0, a[100], *p;
    printf("Enter number of integers\n");
    scanf("%d",&n);
    printf("Enter the integers\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    p=a;
    for(i=0;i<n;i++)
        {
        sum=sum+*p;
        p++;
    }
    printf("sum=%d",sum);
}
```



POINTER TO POINTER (DOUBLE POINTER)

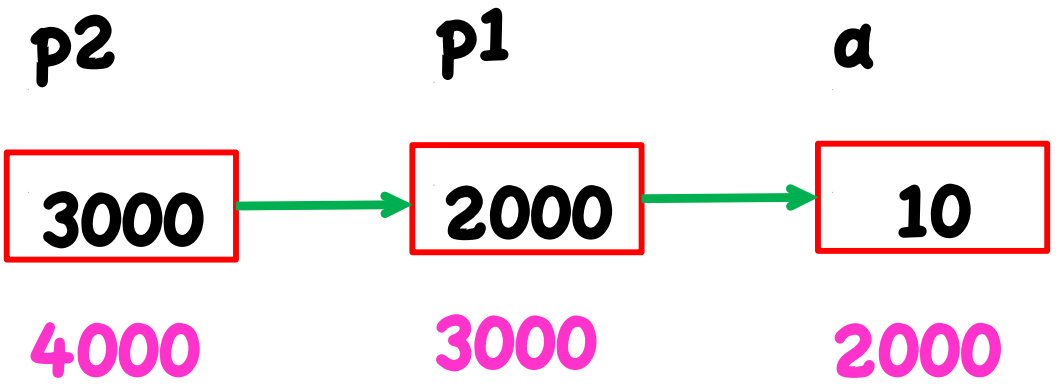
A pointer variable which holds address of another pointer variable is pointer to pointer.

Declaration

```
datatype **pointer;
```

Example

```
int a=10;
int *p1,**p2;
p1=&a;
p2=&p1;
```



Where in above example p1 and p2 are two pointers.

Pointer p1 is pointing to address of a. Where &p1=3000, p1= 2000, *p1=10

Pointer p2 is pointing to address of p1. Where &p2=4000, p2=3000, *p2=2000, **p2=10

Write a C program to copy a string to another using a pointers.

```
#include<stdio.h>
void main()
{
   char str1[100],str2[100],*p1,*p2;
   p1=str1;
   p2=str2;
   printf("Enter a string\n");
   scanf("%s",str2);
         while(*p2!='\0')
         {
            *p1=*p2;
             p1++;
             p2++;
        }
       *p1='\0';
        printf("Copied string is = %s",str1);
}
```

PREPROCESSOR DIRECTIVES

- Preprocessor directives are the instructions to the C compiler.
- These modify a source program before converting it into object program.
- These lines are not program statements.
- They begin with # (hash), the ; (semicolon) is not required end of these lines.
- The link section and definition section are the preprocessor directives.
- The link section is used to include header files to have definitions for library functions or macros used in source program.
- Definition section gives a value for a symbolic constant.
- To extend preprocessor directive in more than one line use \ (slash) at the end of the line. Like below example.

Example
```
#include     \
        <stdio.h>
```

The preprocessor directives used in C programming are

#include #define #undef #ifdef #ifndef #if #else #endif #error #line #pragma

CATEGORIES OF PREPROCESSOR DIRECTIVES

There are three categories in preprocessor directives

1. Macro substitution directive
    a) Simple macro substitution
    b) Argumented macro substitution
    c) Nested macro substitution
2. File inclusion directive
3. Compiler controlled directives

1. MACRO SUBSTITUTION DIRECTIVES

Macro substitution is a process where an identifier (symbolic name) in a program is pre-defined by a string (value).

Syntax

```
#define identifier  string
```

a) Simple macro substitution

Examples

1. #define PI 3.1415927
2. #define FALSE 0
3. #define MAX 100
4. #define M 5

```
------------------
total=M+1;
printf("Total=%d",total);
```

During execution

```
total=5+1;
printf("Total=%d",total);
```

1. MACRO SUBSTITUTION DIRECTIVES

b) Argumented macro substitution

Syntax

```
#define identifier(f1,f2,...) string
```

Example 1

```
#define CUBE(x) x*x*x

c=CUBE (side);

d=CUBE(a+b);
```

arguments

After compilation

```
c=side*side*side;

d=(a+b)*(a+b)*(a+b);
```

1. MACRO SUBSTITUTION DIRECTIVES

Example 2

```
#include <stdio.h>
#define MAX(a, b)  a > b ? a : b
void main()
{
      int c;
      c = MAX(5, 6);
      printf("%d",c);
}
```

Output

```
6
```

Example 3

```
#include <stdio.h>
#define ABS(x)  x > 0 ? x : -x
void main()
{
      int c;
      c = ABS(-8);
      printf("%d",c);
}
```

Output

```
8
```

1. MACRO SUBSTITUTION DIRECTIVES

Write a c program to find addition of two squared number using macro SQR(x)

```
#include<stdio.h>
#define SQR(x) x*x
void main()
{
    int a,b,c;
    printf("Enter two integers\n");
    scanf("%d%d",&a,&b);
    c=SQR(a)+SQR(b);
    printf("The sum of two squared numbers=%d",c);
}
```

Output

```
Enter two integers
2
3
The sum of two squared numbers=13
```

1. MACRO SUBSTITUTION DIRECTIVES

c) Nested macro substitution

A macro defined inside another macro is referred as nested macro substitution.

Example 1

```
#define M 5

#define N  M+1
```

In above example is macro M is nested inside macro N to give value N = 6

Example 2

```
#define SQR(x)  x*x

#define CUBE(x)  SQR(x)*x
```

In above example macro SQR(x) is nested inside CUBE(x) macro.

2. FILE INCLUSION DIRECTIVES

Syntax

```
#include<filename.h>
```

OR

```
#include "filename.c"
```

Example

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

```
#include"file1.c"
```

- The macros and built-in functions are called from the included files to the current program.
- The file inclusion directive embeds the specified file into current program.
- If header file name is enclosed within < > then search for that header file is made only in standard folders. Example for standard folders include, lib etc.
- If file is included within " " then search for that file is made in current folder and then it searches in standard folders .

3. COMPILER CONTROLLED DIRECTIVES

These are instruction to control compiler

Examples #ifdef #else #endif #if #ifndef #undef #error

Condition Compilation

This is used to switch on or off a particular line or group of lines in program

#ifdef #else #endif

Syntax

```
#ifdef identifier
    statements;
#else
    statements;
#endif
```

Example

```
#include<stdio.h>
#define LINE 5
void main()
{
        #ifdef LINE
            printf("First line");
        #else
            printf("Second line");
        #endif
}
```

In above example #ifdef checks whether identifier LINE is defined?

Where LINE is defined, so the output is First line

3. COMPILER CONTROLLED DIRECTIVES

#ifndef

Syntax

```
#ifndef identifier
    statements;
#else
    statements;
#endif
```

Example

```
#include<stdio.h>
#define LINE 5
void main()
{
    #ifndef LINE
      printf("First line");
    #else
      printf("Second line");
    #endif
}
```

In above example #ifndef checks whether identifier LINE is not defined?

Where LINE is defined, so the output is Second line

3. COMPILER CONTROLLED DIRECTIVES

#if

Syntax

```
#if identifier
    statements;
#else
    statements;
#endif
```

Example

```
#include <stdio.h>
#define LINE 5
void main()
{
  #if LINE
     printf("First line");
  #else
    printf("Second line");
  #endif
}
```

In above example #if checks whether identifier LINE is found or not?

Where LINE is found so the output is First line

3. COMPILER CONTROLLED DIRECTIVES

#undef

It is used to restrict the definition of a macro to particular part of program

Syntax

```
#undef identifier
```

Example

```
#define M 5

a = M + 1;          // After compilation a = 6

#undef M

b = M + 1;          // Is an error. Because in previous statement M is undefined.
```

3. COMPILER CONTROLLED DIRECTIVES

#error

It is used to make computation fail and issue a statement that will appear in the list of compilation errors.

Syntax

```
#error message
```

Example 1

```
#include<stdio.h>
#define LINE 5
void main()
{
  #ifdef LINE
    #error Macro is available
  #else
    printf("Macro is not available");
  #endif
}
```

In example1 the LINE macro is defined.

The #ifdef checks whether the identifier LINE is defined?

Since LINE is defined, the compiler fails and executes #error so the output is

#error Macro is available

3. COMPILER CONTROLLED DIRECTIVES

Example 2

```
#include<stdio.h>
void main()
{
  #ifdef LINE
     #error Macro is available
  #else
     printf("Macro is not available");
  #endif
}
```

In example2 the LINE macro is not defined.

The #ifdef checks whether the identifier LINE is defined?

Since LINE is not defined the output is

Macro is not available