Pointers

Pointer

- **A *pointer* is a variable that represents the *location* (rather than the *value)* of a data item**
- **Indirect Variable**
- pointers can be used to pass information back and forth between a function and its reference point
- pointers provide a way to return multiple data items from a function via function arguments

Contd…

- pointers also permit references to other functions to be specified as arguments to a given function
- **pointers are also closely associated with arrays and therefore provide an alternate way to access individual array elements**
- int *u

  u is a pointer that store address

```
#include<stdio.h>
main( )
{
int u=5;
int *pu;
pu=&u;
printf("\n u=%d  &u=%x  pu=%x
    *pu=%d",u,&u,pu,*pu);
}
```

```
#include<stdio.h>
main()
{
int u=5;
int *pu;
printf("\n u=%d  &u=%x  pu=%x
   *pu=%d",u,&u,pu,*pu);
}
```

**pu and pu give garbage value*

```
#include<stdio.h>
main()
{
int u=5;
int *pu=6;
printf("\n u=%d  &u=%x  pu=%x
   *pu=%d",u,&u,pu,*pu);
}
```

Non portable pointer conversion

*u=5, &u=fff4, pu=6, *pu=27762*

```
#include<stdio.h>
main()
{
int u=5;
int *pu;
pu=&u;
printf("\n u=%d  &u=%x  pu=%x  *pu=%d",u,&u,pu,*pu);

*pu=u+2;

printf("\n u=%d  &u=%x  pu=%x  *pu=%d",u,&u,pu,*pu);

u=*pu*5;

printf("\n u=%d  &u=%x  pu=%x  *pu=%d",u,&u,pu,*pu);
}
```

PASSING POINTERS TO A FUNCTION

- passing pointer as an argument to the function allows data items within the calling portion of the program to be accessed by the function, altered within the function, and then returned to the calling portion of the program in altered form
- passing pointers as arguments is known as passing by ***reference*** (or by ***address*** or by ***location),*** in contrast to passing arguments by ***value***

```
void passingbyvalue(int u,int v)
{
u=0, v=0;
printf("\n Within passingbyvalue
    function: u=%d  v=%d",u,v);
return;
}

void passingbyref(int *pu,int *pv)
{
*pu=0, *pv=0;
printf("\n Within passingbyref
    function: *pu=%d
    *pv=%d",*pu,*pv);
return;
}
```

```
main()
{
int u=5,v=6;

printf("\nBefore calling function
passingbyvalue: u=%d v=%d",u,v);

passingbyvalue(u,v);

printf("\nAfter calling function
passingbyvalue: u=%d v=%d",u,v);

printf("\n\nBefore calling function
passingbyref: u=%d v=%d",u,v);

passingbyref(&u,&v);

printf("\nAfter calling function
passingbyref: u=%d v=%d",u,v);
}
```

```
#include<stdio.h>
#include<ctype.h>
void scan_line(char line[],int *pv,int
    *pc,int *pd,int *pw,int *po);
main()
{
char line[80];
int
    vowels=0,consonants=0,digits=0,whi
    tespc=0,other=0;
printf("Enter a line of text below:\n");
scanf("%[^\n]",line);
scan_line(line,&vowels,&consonants,&d
    igits,&whitespc,&other);
printf("\nNo. of vowels:%d",vowels);
printf("\nNo. of
    consonants:%d",consonants);
printf("\nNo. of digits:%d",digits);
printf("\nNo. of whitespace
    characters:%d",whitespc);
printf("\nNo. of other
    characters:%d",other);
}
```

```
void scan_line(char line[],int *pv,int *pc,int
*pd,int *pw,int *po)
{
char c;
int count=0;
while ((c=toupper(line[count]))!='\0'){
        if(c=='A'|| c=='E'|| c=='I'||c=='O'||
c=='U')
        ++*pv;
        else if(c>='A' && c<='Z')
                ++*pc;
                else if(c>='0' && c<='9')
                        ++*pd;
                    else if(c ==' ' || c == '\t')
                            ++*pw;
                else

        ++*po;

        ++count;
        }
        return;
}
```

POINTERS AND ONE-DIMENSIONAL ARRAYS

```
#include<stdio.h>

char x[ ] = "This string is declared
    externally\n";

main()
{
static char y[ ] = "This string is
    declared within main";
printf("%s",x);
printf("%s",y);
}
```

```
#include<stdio.h>

char *x= "This string is declared
externally\n";

main()
{
char *y = "This string is declared
within main";
printf("%s",x);
printf("%s",y);
}
```

Contd…

- an array name is a pointer to the first element in the array
- **x** is a one dimensional array, then the address of the first array element can be expressed **as** either **&x[0]** or simply **as x**

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

DYNAMIC MEMORY ALLOCATION

- an array name is actually a pointer to the first element within the array, it should be possible to define the array as a pointer variable rather than as a conventional array
- *a conventional array definition results in a fixed block of memory being reserved at the beginning of program execution, whereas this does not occur if the array is represented in terms of a pointer variable*

Contd…

- the use of a pointer variable to represent an array requires some type of initial memory assignment before the array elements are processed
- known as ***dynamic memory allocation***

- the **malloc** library function is used for this purpose
- stdlib.h or malloc.h

- x=(data_type *)malloc(n*sizeof(data_type));

```
#include<stdio.h>

int SUM(int n,int *x)
{
int i,sum=0;
for(i=0;i<n;++i){
    sum=sum+ *(x+i);
    }
return sum;
}
```

```
main()
{
int *x,i,n,y;
printf("how many number are u entering?");
scanf("%d",&n);
printf("\n");
x=(int *)malloc(n*sizeof(int));
for(i=0;i<n;++i)
            {
            scanf("%d",x+i);
            }
y=SUM(n,x);
printf("sum of %d nos is %d",n,y);
}
```

OPERATION ON POINTERS

```
#include<stdio.h>
main()
{
int *px;
int i=1;
float f=0.3;
float d=0.005;
char c= '*';
px=&i;
printf("Value: i:%i f:%f d:%f c:%c\n",i,f,d,c);
printf("Addresses: &i:%x &f:%x &d:%x &c:%x\n",&i,&f,&d,&c);
printf("Pointer values: px:%x px+1:%x px+2:%x
    px+3:%x",px,px+1,px+2,px+3);
}
```

Contd…

```
#include<stdio.h>
main()
{
int *px, *py;
int a[6] = {1,2,3,4,5,6};
px=&a[0];
py=&a[5];
printf("a[0]:%x a[5]:%x px:%x py:%x\n",&a[0],&a[5],px,py);
printf("*py-*px:%d\n",*py-*px);
}
```

POINTERS AND MULTIDIMENSIONAL ARRAYS

- multidimensional array can also be represented with an pointer notation
- two-dimensional array, for example, is actually a collection of one-dimensional arrays
- we can define a two-dimensional array as a pointer to a group of contiguous one-dimensional arrays

Contd…

- a two-dimensional array declaration can be written as

  data- type (*ptvar) [expression 2] ;

  rather than

  data- type array[expression 1] [expression 2];

Contd…

"x" is a two-dimensional integer array having 10 rows and 20 columns

int (*x)[20];

rather than

int x[10][20];

"x" is defined to be a pointer to a group of contiguous, one-dimensional, 20-element integer arrays

Contd…

```
x[0]= (int *) malloc (20 * sizeof(int));
.
.
x[9]= (int *) malloc (20 * sizeof(int));
```

Contd…

- the item in row 3, column 6 can be accessed by writing either

  a[2][5]

  or

  * (* (x + 2) + 5)

matrix addition using pointers

- *(*(c+row)+col)

  =

  ((a+row)+col)+*(*(b+row)+col)

```
#include<stdio.h>
#define MAXROWS 20

void readinput(int *a[MAXROWS], int nrows, int ncols);
void computesums(int *a[MAXROWS],int *b[MAXROWS],int *c[MAXROWS],int nrows,int ncols);
void writeoutput(int *c[MAXROWS],int nrows,int ncols);

 main()
{
int row,nrows,ncols;
int *a[MAXROWS],*b[MAXROWS],*c[MAXROWS];
printf("How many rows?");
scanf("%d",&nrows);
printf("\n How many columns");
scanf("%d",&ncols);
for(row=0;row<nrows;++row){
        a[row]=(int *)malloc(ncols*sizeof(int));
        b[row]=(int *)malloc(ncols*sizeof(int));
        c[row]=(int *)malloc(ncols*sizeof(int));
        }
printf("\n\nFirst table:\n");
readinput(a,nrows,ncols);
printf("\n\nSecond table:\n");
readinput(b,nrows,ncols);
computesums(a,b,c,nrows,ncols);
printf("\n\nSums of the elements:\n\n");
writeoutput(c,nrows,ncols);
}

void readinput(int *a[MAXROWS],int m,int n)
{
int row, col;
for(row=0;row<m;++row){
        for(col=0;col<n;++col){
                scanf("%d",(*(a+row)+col));
                }
        }
}

void computesums(int *a[MAXROWS],int *b[MAXROWS],int *c[MAXROWS],int m,int n)
{
int row, col;
for(row=0;row<m;++row){
        for(col=0;col<n;++col){
                *(*(c+row)+col) = *(*(a+row)+col)+*(*(b+row)+col);
                }
        }
}

void writeoutput(int *a[MAXROWS],int m,int n)
{
int row, col;
for(row=0;row<m;++row){
        for(col=0;col<n;++col){
```

```
#include<stdio.h>
#define MAXROWS 20

void readinput(int *a[MAXROWS], int nrows, int
    ncols);

void computesums(int *a[MAXROWS],int
    *b[MAXROWS],int *c[MAXROWS],int nrows,int
    ncols);

void writeoutput(int *c[MAXROWS],int nrows,int
    ncols);
```

```
 main()
{
int row,nrows,ncols;
int *a[MAXROWS],*b[MAXROWS],*c[MAXROWS];
printf("How many rows?");
scanf("%d",&nrows);
printf("\n How many columns");
scanf("%d",&ncols);
for(row=0;row<nrows;++row){
     a[row]=(int *)malloc(ncols*sizeof(int));
     b[row]=(int *)malloc(ncols*sizeof(int));
     c[row]=(int *)malloc(ncols*sizeof(int));
     }
printf("\n\nFirst table:\n");
readinput(a,nrows,ncols);
printf("\n\nSecond table:\n");
readinput(b,nrows,ncols);
computesums(a,b,c,nrows,ncols);
printf("\n\nSums of the elements:\n\n");
writeoutput(c,nrows,ncols);
}
```

```
void readinput(int *a[MAXROWS],int m,int
   n)
{
int row, col;
for(row=0;row<m;++row){
   for(col=0;col<n;++col){
        scanf("%d",(*(a+row)+col));
        }
   }
}
```

```
void computesums(int *a[MAXROWS],int
   *b[MAXROWS],int *c[MAXROWS],int m,int n)
{
int row, col;
for(row=0;row<m;++row){
   for(col=0;col<n;++col){
        *(*(c+row)+col) =
   *(*(a+row)+col)+*(*(b+row)+col);
        }
   }
}
```

```
void writeoutput(int *a[MAXROWS],int m,int
   n)
{
int row, col;
for(row=0;row<m;++row){
   for(col=0;col<n;++col){
        printf("%d\t",*(*(a+row)+col));
        }
   printf("\n");
   }
}
```