POINTERS

Pointer is a variable that represents the location of a data item.

Pointers are used to pass information back and forth between a function and its reference point.

In fact pointers provide a way to return multiple data items from a function via functional arguments.

Pointers are also closely associated with arrays. They provide an alternate way to access individual array elements.

POINTER FUNDAMENTALS:

Every data item is stored in the computer memory and it occupies one or more bytes of contiguous (neighboring, adjscent) memory cells.

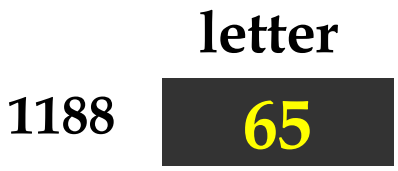
For example a character data occupies one memory location, integer type data occupies 2 byte of memory location, floating type data occupies 4 bytes of memory locations

Example 1:

```
char letter;
letter = 'A';
```

The first statement declares a variable name letter. ie., a computer memory location is referenced by the name "letter". In the second statement a numerical value 65 (ASCII value for character 'A") is placed into the memory cell.

Every memory cell has an unique address. In this example the address is assumed to be 1188. The address corresponding to the variable letter can be determined using the address operator & where & is an unary operator.



The address of the variable letter can be determined using the expression &letter.

The address can be stored to a new variable.
ie., pv = &letter;

The variable pv contains the address of the variable (1188) “letter”.

The new variable is called pointer to “letter” since it points to the location where letter is stored in memory.

The data stored in the memory cell letter can be accessed either by calling the name of the variable as “letter” or by accessing the memory address.

The data item represented by “letter” can be accessed by the expression *pv, where * is an unary operator , called indirection operator.

Note: The indirection operator * can operate only on address. Both “letter”, *(&letter) and *pv represent the same data item ‘A”.

Example 2:

```
#include<stdio.h>
void main() {
char letter;
/*  *ptletter is a pointer variable pointing to a charatcter*/
char  *ptletter;
letter = 'A';
printf("The address of the variable letter is %x\n",&letter);
ptletter = &letter;
printf("The address of the variable letter is %x\n",ptletter);
printf("The character stored in the letter is %c\n",letter);
printf("The character stored in the letter is %c\n",*ptletter);
return;
}
```

Result:

```
The address of the variable letter is fff8
The address of the variable letter is fff8
The character stored in the letter is A
The character stored in the letter is A
```

Note: The value pointed by a pointer variable can be used in an expression.

Example 3:

```
#include<stdio.h>
void main(void) {
int a,b;
int *pta;
a = 10;
pta = &a;
b = 2 * ( *pta + 15);
printf("The expression value is %d\n",b);
return;}
```

Result:

The expression value is 50

Note: The indirect reference operator can appear on the left hand side of an assignment statement.

Example 4:

```
#include<stdio.h>
void main(void) {
int num = 5;
int *ptnum;
ptnum = &num;
printf("value in num is %d\n",num);
/* indirection operator is used on the left hand side of an assignment
statement*/
*ptnum = 30;
/* As num and *ptnum referes to the same location the value stored in
num is also 30 */
printf("value in num is %d\n",num);
return; }
```

Result:

value in num is 5
value in num is 30

The following is a simple program to multiply two numbers using pointers.

```
#include<stdio.h>
void main(void)
{
float x = 12.0;
float y = 24.0;
float *ptrx, *ptry;
float result = 0.0;
float *ptrresult;
ptrx = &x;
ptry = &y;
ptrresult = &result;
*ptrresult = *ptrx * *ptry;
printf("%6.2f * %6.2f = %6.2f\n",*ptrx, *ptry, *ptrresult);
return;
}
```

The output of the above program is

12.00 * 24.00 = 288.00

How a pointer variable can be declared?
The syntax for declaring a pointer variable is :
datatype *pt_variablename;

Note: There should be a star symbol before the variable name.
Example 5:
int *stumark;
float *weight;
char *str;
%s format %[ ] format specifier:

The %s format and the square bracket format specifiers are used to read string of characters.

When a %s format is used in a scanf statement, it will read all the character from the input buffer till it encounters a blank space.

When a %[ ] format is used in a scanf statement, it will read all the characters from the input buffer till it encounters a character that is not found in the list within the square bracket. When ^ symbol is used as a first symbol in side the %[ ]format specifier then it represents compliment of characters.

Example 6:

char string[40];
scanf(“%s”,string);
input : today is very hot.

Value stored in string is “today”
char string [40];

scanf(“%[1234567890abcdefghijklmnopqrstuvwxyz]” , string);

input: today is very hot
output: today is very hot

scanf(“%[^\n]”,string);

Example 7:

```
/* Program to read a string and to count the number of vowels */

#include<stdio.h>
void count(char string[ ], int *vowel,
int *const, int *whitespace);
void main(void)
{
char string[80];
int vowel = 0;
int const = 0;
int whitespace = 0;
printf("Please enter a string");
scanf("%[^\n]",string);
count(string,&vowel,&const,&whitespace);
printf("The number of vowels = %d\n",vowels);
printf("The number of constants = %d\n",const);
printf("The number of whitespace = %d\n",whitespace);
return;
}
```

```
void count(char string[ ], int *vowel,*const, int *whitespace)
{
int i  = 0;
char c;
while( (c=toupper(string[i]) != '\0') {
if( c == 'A'  || c== 'E' || c == 'I' || c == 'O'  ||  c == 'U')
            ++ *vowel;
else if(c >='A' &&  c<='Z')
            ++ *const;
else if(c == ' '  || c == '\t')
            ++ *whitespace;
++ i;
}
return;
}
```

POINTERS AND ONE-DIMENSIONAL ARRAY

Consider the following assignment statement:

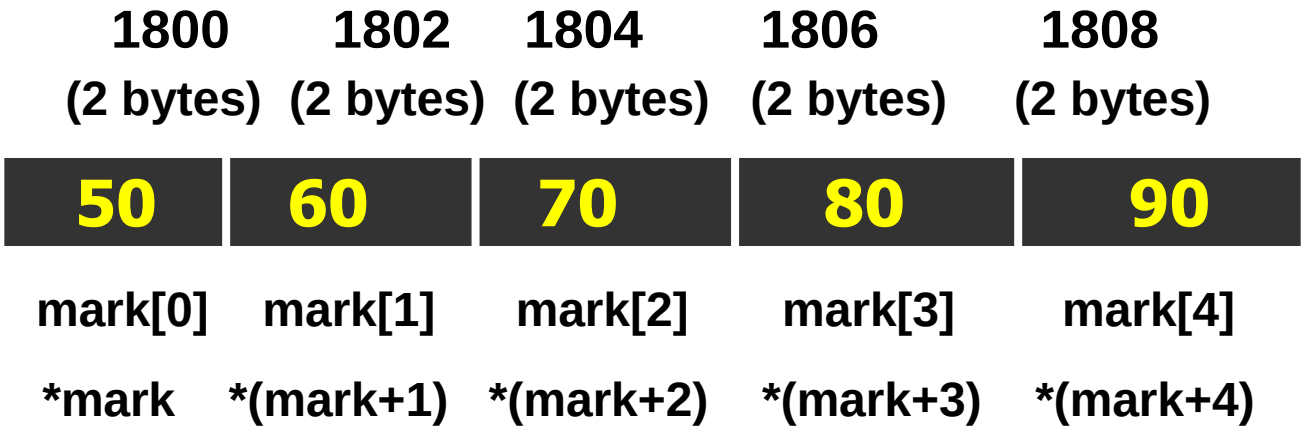
int mark[5] = {50,60,70,80,90};

The above statement assigns the values of array cells as:

Mark[0] = 50; mark[1] = 60; mark[2] = 70;

Mark[3] = 50; mark[4] = 90;

The following diagram shows the way in which the data are stored contiguously.



With reference to the above figure, the address of mark[0] is 1800.

The address of the first array element of mark can be represented as &mark[0] or simply by "mark" itself. ie., &mark[0] and mark represents the address 1800.

The address of the second array element can be written as &mark [1] or (mark + 1) and similarly the address of the fifth element can be written as &mark [4] or (mark+4).

In general, the address of (i+1)th array element can be written as &mark[i] 0r (mark+i).

Similarly mark[i] or *(mark+i) represents the value at the (i+1)th location. mark[1] or *(mark+1) represents the value 60. Either term can be used in any particular application. The choice depends upon the programmer's individual preferences.

Array Element	Address	
	Sub Notion	Pointer Notion
0	&mark[0]	mark
1	&mark[1]	mark+1
2	&mark[2]	mark+2
3	&mark[3]	mark+3
4	&mark[4]	mark+4

Array Element	Value	
	Sub Notion	Pointer Notion
0	mark[0]	*(mark)
1	mark[1]	*(mark+1)
2	mark[2]	*(mark+2)
3	mark[3]	*(mark+3)
4	mark[4]	*(mark+4)

Example 8:

```
#include<stdio.h>
void main(void) {
int i;
int x[5] = {50,60,70,80,90};
for(i = 0;i <5;i++)
printf("i = %d, address is %x, value is %d\n",i,&x[i], x[i]);
printf("\n");
for(i = 0;i <5;i++)
printf("i = %d, address is %x, value is %d\n",i,(x+i), *(x+i));
return;}
```

```
0  address is 1800 value is 50
1  address is 1802 value is 60
2  address is 1804 value is 70
3  address is 1806 value is 80
4  address is 1808 value is 90
0  address is 1800 value is 50
1  address is 1802 value is 60
2  address is 1804 value is 70
3  address is 1806 value is 80
4  address is 1808 value is 90
```

Example 9:
Replace each of the following references to a subscripted variable with a pointer reference.

a. prices[5]	*(prices+5)
b. Celsius[16]	*(celsius+16)
c. mile[0]	*mile

Example 10:
Replace each of the following referenced using a pointer with a subscript reference.

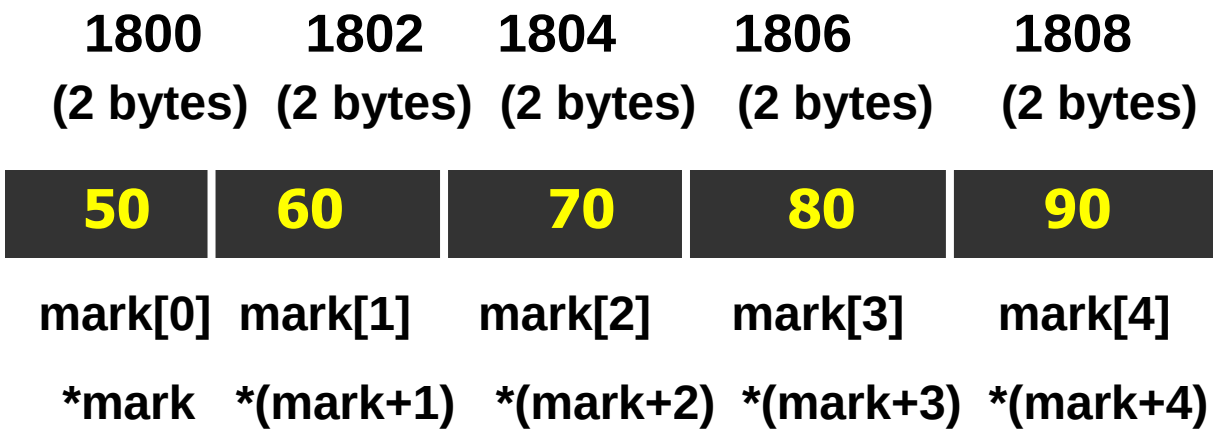
a. *(mes+6)	= mes[6]
b. *(mark+10)	= mark[10]
c. *(rate+30)	= rate[30]

Pointer Arithmetic:
Pointer variables, like all variables, contain values. The value stored in pointer variable is an address. Thus by adding , subtracting numbers to pointers we can obtain other addresses. Further, the addresses in pointers can be compared using any of the relational operators (> >= <= < == !=).

Consider the following assignment statement:

int mark[5] = {50,60,70,80,90};

Once the mark is defined, mark is a pointer variable having the initial address as 1800.
mark + 2 will yield the address 1804.
Here mark + 2 is 1800 + 2*sizeof(int) = 1804.



Note : The address can be incremented or decremented.

Pointer Initialization:

Like all variables pointer variables can be initialized when they are declared.

Example 13:

```
int speed;
int *ptspeed = &speed;
```

PASSING POINTERS TO A FUNCTION:

Pointers can be passed to a function as arguments. This allows the data items within the calling portion of then program to be accessed by the function, altered within the function, and then returned to the calling portion of the program in altered form. This method of passing values to a function is called pass (call) by reference/address.

Example 14:

```
#include<stdio.h>
void modify1(int x, int y);
void modify2(int *px, int *py);
void main(void) {
int x,y;
int *px, *py;
x = 10;
y = 20;
px = &x;
py = &y;
printf("value of x any y before calling modify1  %d  %d\n",x,y);
modify1(x,y);
printf("value of x any y after calling modify1  %d  %d\n",x,y);
printf("value of x any y before calling modify2  %d  %d\n",x,y);
/* Note for modify the values are passed as addresses */
modify2(&x,&y);
printf("value of x any y after calling modify2  %d  %d\n",x,y);
return;}
```

```
void modify1(int x, int y)
{
  x = 33;
  y = 66;
  return;
}
void modify2(int *px, int *py)
{
  *px = 33;
  *py = 66;
  return;
}
```

value of x any y before calling modify1 10 20
value of x any y after calling modify1 10 20
value of x any y before calling modify2 is 10 20
value of x any y after calling modify2 is 33 66

The following program reads the side of a square and computes it's perimeter and area using a function.

```
#include<stdio.h>
void square(float s, float *a, float *p);
void main(void) {
float side,area,perimeter;
printf("Please enter the side ");
scanf("%f",&side);
square(side,&area,&perimeter);
printf("Side = %6.2f\n",side);
printf("Area = %8.2f\n",area);
printf("Perimeter = %6.2f\n",perimeter);
return; }

void square(float s, float *a, float *p) {
*a = s*s;
*p = 4*s;
return; }
```

```
Side = 10.00;
Area = 100.00;
Perimeter = 40.00
```

Pointer to a pointer

Pointer variables can point to a numeric or character variables, arrays, structures, or another pointer variable. That is a pointer variable can be assigned the value of another pointer variable. In this case both the pointer variables must point to data items of same type.

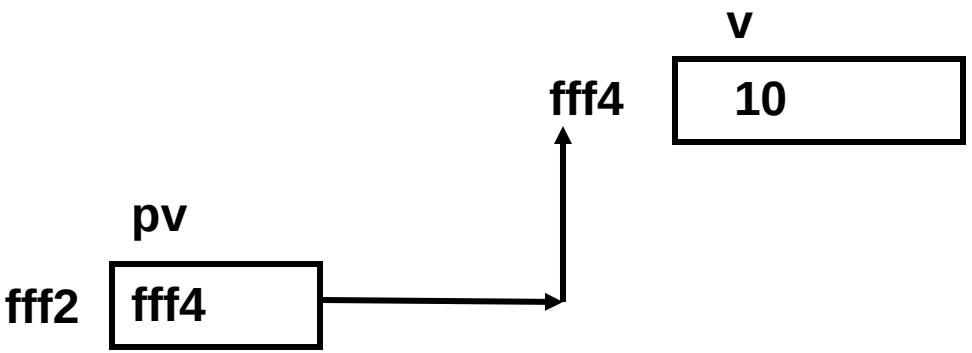
Consider the following program:

```
#include<stdio.h>
void main(void)
{
int num = 10;
int *numptr1,**numptr2;
numptr1 = &num;
numptr2 = &numptr1;
printf("The number is %d\n",num);
printf("The number pointed by numptr1 is %d\n",*numptr1);
printf("The number pointed by numptr2 is %d\n",**numptr2);
return;
}
```

The output of the above program is

```
The number is 10
The number pointed by numptr1 is 10
The number pointed by numptr2 is 10
```

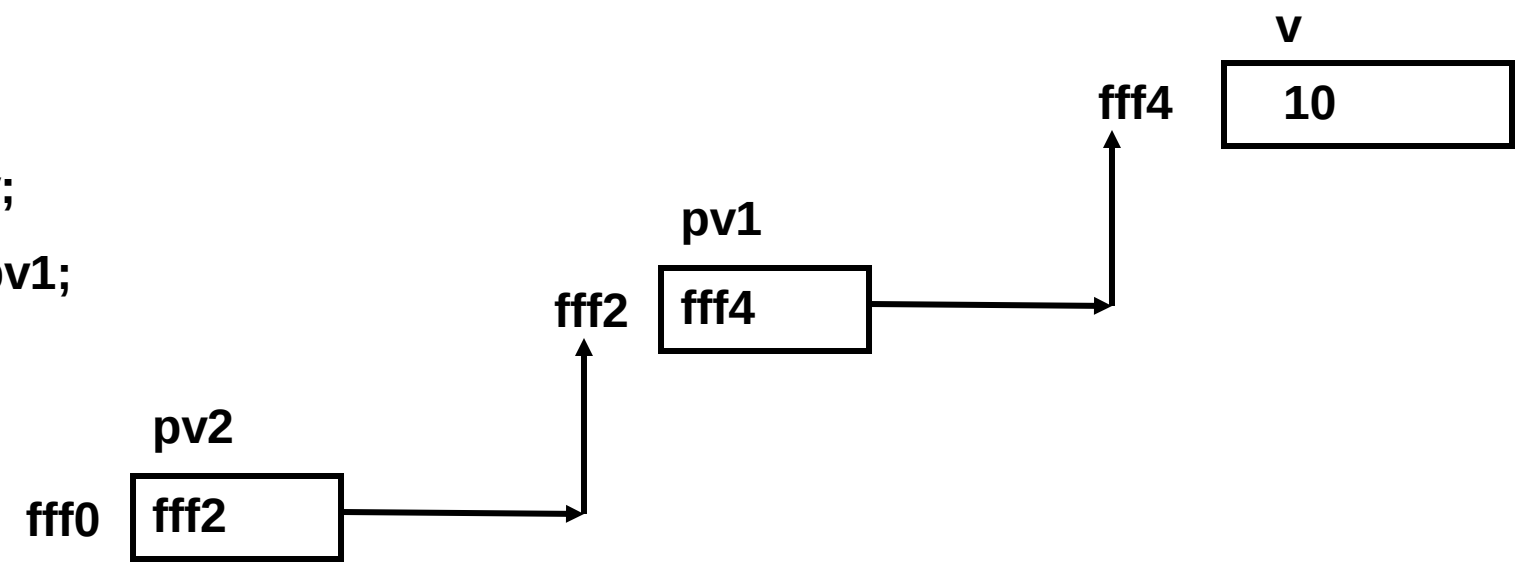
```
Int v = 10;
Int *pv;
pv = &v;
```




```
printf("%d\n", v);          /* This will display 10 */
printf("%x\n",&v);          /* This will display fff4 */
printf("%x\", pv);          /* This will display fff4 */
printf("%x\n", &pv);        /* This will display fff2 */
printf("%d\n", *pv);        /* This will display 10 */
printf("%d\n", *(&v));      /* This will display 10 */
```

Pointer to a Pointer

```
Int v = 10;
Int *pv1;
Int *pv2;
pv1 = &v;
pv2 = &pv1;
```



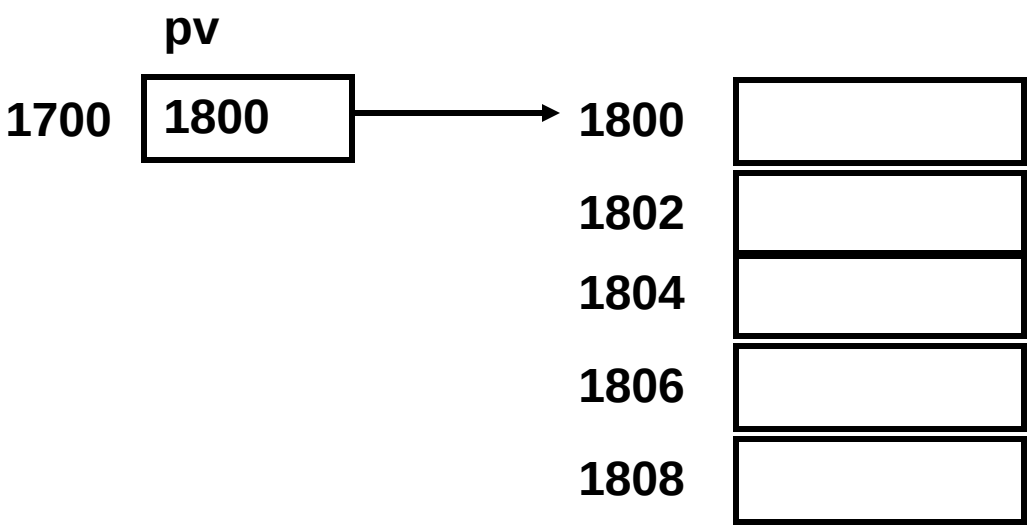
```
#include<stdio.h>
int main(void) {
int v, *pv1, **pv2;
v = 10;
pv1 = &v;
pv2 = &pv1;

printf("Address of v is %x\n", &v);
printf("Address of pv1 is %x\n",&pv1);
printf("Address of pv2 is %x\n", &pv2);
printf("Value at v is %d\n", v);
printf("Value at v is %d\n", *pv1);
printf("Value at v is %d\n", **pv2);
return 0; }
```

```
Address of v is fff4
Address of pv1 is fff2
Address of pv2 is fff0
Value at v is 10
Value at v is 10
Value at v is 10
```

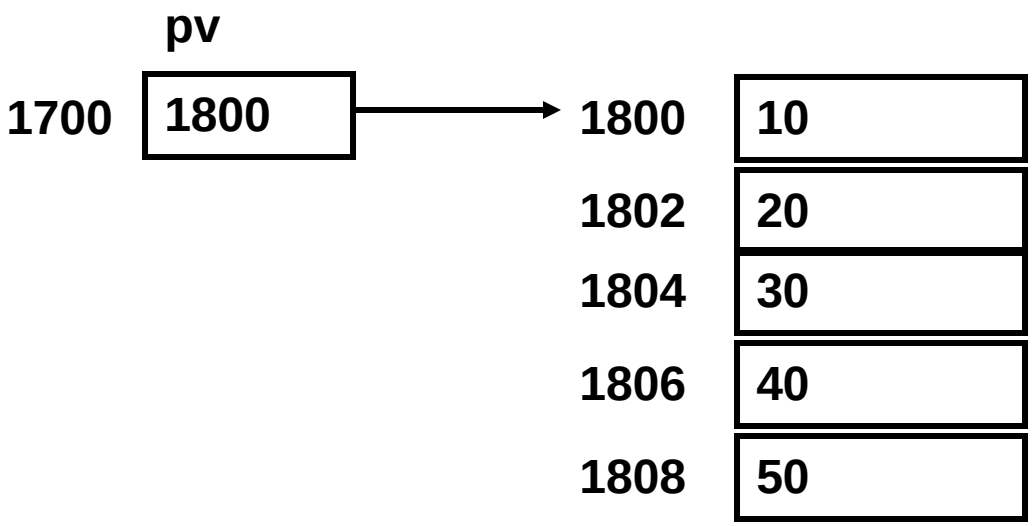
One Dimensional array and Pointer

```
int I, *pv;
pv = (int *) malloc(5 * sizeof(int) );
```

pv represents 1800
pv + 1 represents 1802
pv + 2 represents 1804
pv + 3 represents 1806
pv + 4 represents 1808




```
*pv = 10;
*(pv+1) = 20;
*(pv+2) = 30;
*(pv+3) = 40;
*(pv+4) = 50
```



Array of pointers

One dimensional array can be represented in terms of a pointer and an offset. A two dimensional array is a collection of one dimensional arrays. Hence a two dimensional array can also be represented using pointers.

We can represent a multi-dimensional arrays using

Pointer to a group of array : A pointer is defined to point a group of contiguous one dimensional arrays.

Array of pointers An array of pointers pointing a contiguous one dimensional arrays.

Pointer to a Group of arrays

Syntax:

Data_type (*pt_name)[expression 2];

Data type represents the type of data

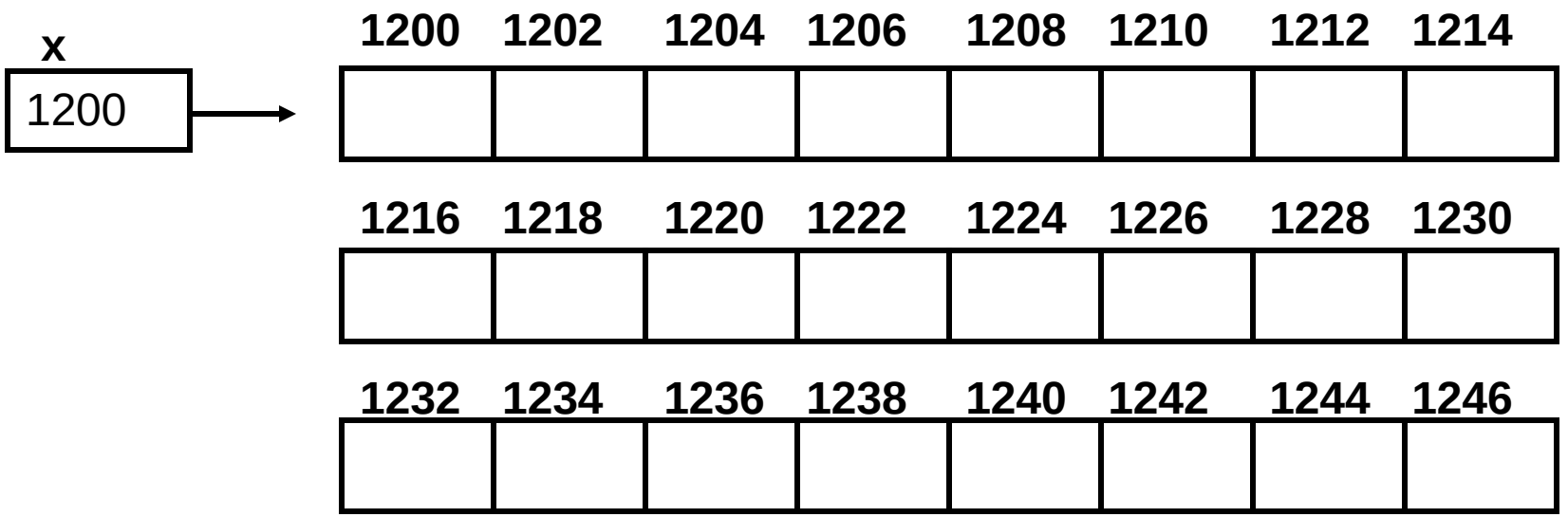
pt_name represents the name of the pointer

Expression 2 indicates the number of column elements.

The above declaration indicates that pt_name is a pointer variable pointing to a group of contiguous, one dimensional array consisting of ‘expression 2’ elements. If we want to have more than one array then use malloc to allocate the contiguous memory locations.

Example:

```
int (*x)[8];
x = (int *) malloc(3*8*sizeof(int));
```



Array of Pointers

Syntax:

Data_type *pt_name[expression 1];

Data type represents the type of data

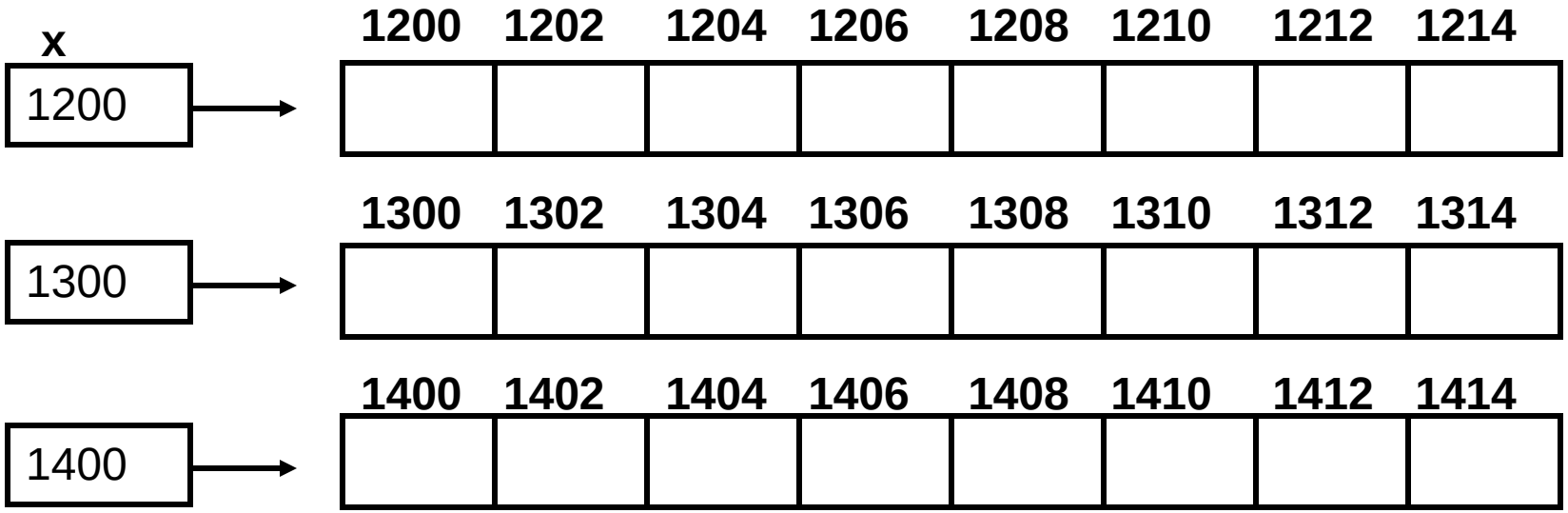
pt_name represents the name of the pointer

Expression 1 indicates the number of row elements.

The above declaration indicates that pt_name is a one dimensional array whose contents are pointer variables pointing to a group of contiguous, one dimensional arrays.

Example:

```
int *x[3];
int i;
for(i=0;i<3;i++)
x[i] = (int *) malloc(3*8*sizeof(int));
```



```
#include<stdio.h>

/* A Two dimensional array is a collection
of
   one dimensional arrays.
   A pointer to a group of arrays can be
used to
   declare a two or more dimensional
arrays.
   The general form is (*p)[expression 2] */
int main(void)
{
int i, j, (*p)[3];
p = (int *) malloc(5 * 3 * sizeof(int));

printf("Starting Address ia %x\n",p);
printf("Address of consecutive rows\n");

for(i=0;i<5;i++)
   printf("%x\t", p[i]);
printf("\n\n");
```

```
for(i=0;i<5;i++)
   {
   for(j=0;j<3;j++)
      printf("%x\t", p[i] + j);
   printf("\n");
   }
```

```
for(i=0;i<5;i++)
   for(j=0;j<3;j++)
      *(*(p+i)+j) = i*j;

printf("\n");
for(i=0;i<5;i++)
   {
   for(j=0;j<3;j++)
      printf("%d\t", *(p[i]+j));
   printf("\n");
   }
printf("\n");

for(i=0;i<5;i++)
   {
   for(j=0;j<3;j++)
      printf("%d\t", *(*(p+i)+j));
   printf("\n");
   }

return 0;
}
```

Starting Address ia 818
Address of consecutive rows

818	81e	824	82a	830

818	81a	81c
81e	820	822
824	826	828
82a	82c	82e
830	832	834

0	0	0
0	1	2
0	2	4
0	3	6
0	4	8

0	0	0
0	1	2
0	2	4
0	3	6
0	4	8

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

/* A Two dimensional array is a
collection of
   one dimensional arrays.
   An array of pointers can be used to
   declare a two or more dimensional
arrays.
   The general form is *p[expression 1] */

int main(void)
{
int i, j, *p[5];
for(i=0;i<5;i++)
p[i] = (int *) malloc(3 * sizeof(int));
clrscr();
printf("Address of consecutive rows\n");
for(i=0;i<5;i++)
   printf("%x\t", p[i]);
printf("\n\n");
```

```
for(i=0;i<5;i++)    {
   for(j=0;j<3;j++)
      printf("%x\t", p[i] + j);
   printf("\n");    }

for(i=0;i<5;i++)
   for(j=0;j<3;j++)
      *(*(p+i)+j) = i*j;
printf("\n");
for(i=0;i<5;i++)    {
   for(j=0;j<3;j++)
      printf("%d\t", *(p[i]+j));
   printf("\n");       }
printf("\n");

for(i=0;i<5;i++)
   {
   for(j=0;j<3;j++)
      printf("%d\t", *(*(p+i)+j));
   printf("\n");
   }
return 0;
}
```

Address of consecutive rows

5d2	5dc	5e6	5f0	5fa

5d2	5d4	5d6
5dc	5de	5e0
5e6	5e8	5ea
5f0	5f2	5f4
5fa	5fc	5fe

0	0	0
0	1	2
0	2	4
0	3	6
0	4	8

0	0	0
0	1	2
0	2	4
0	3	6
0	4	8