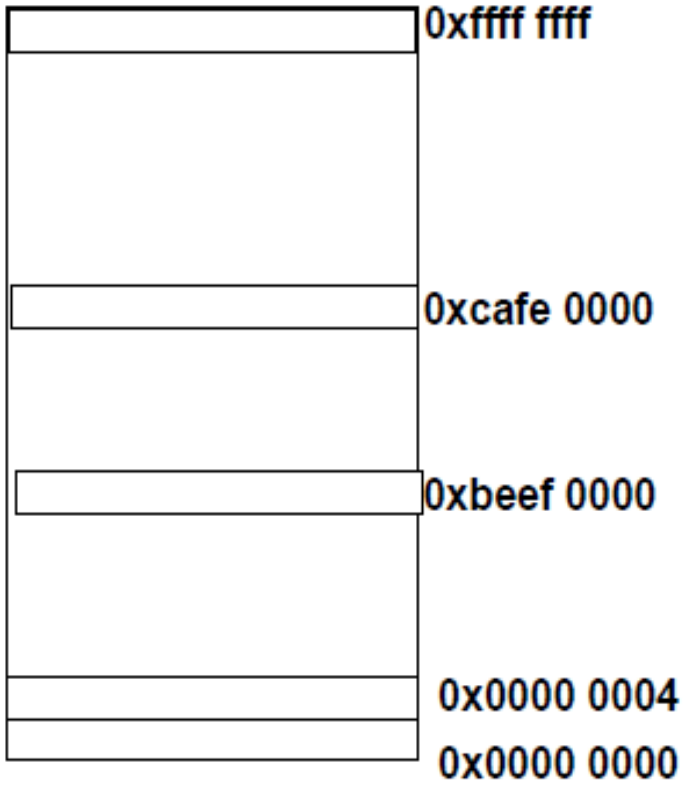
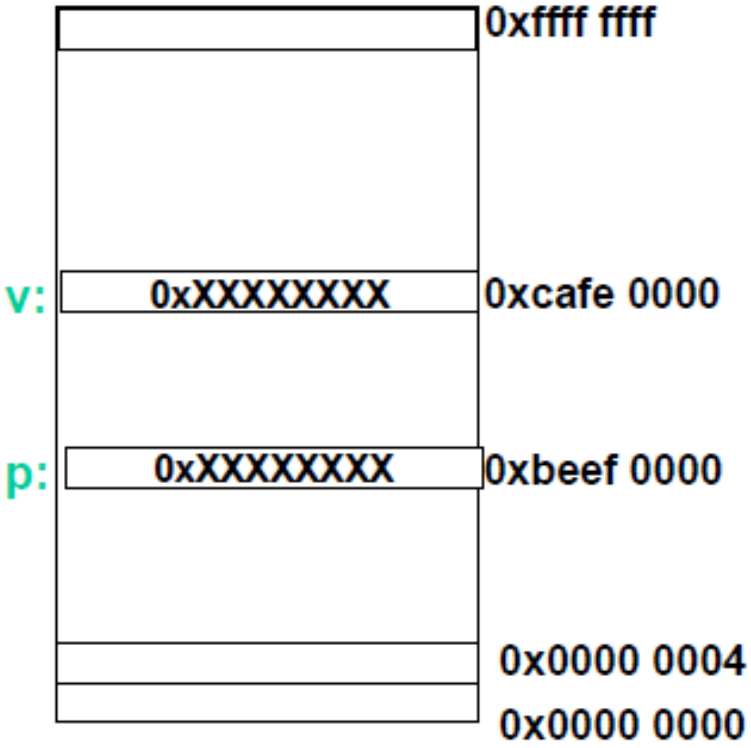
Pointer Usage Example



Memory and Pointers:

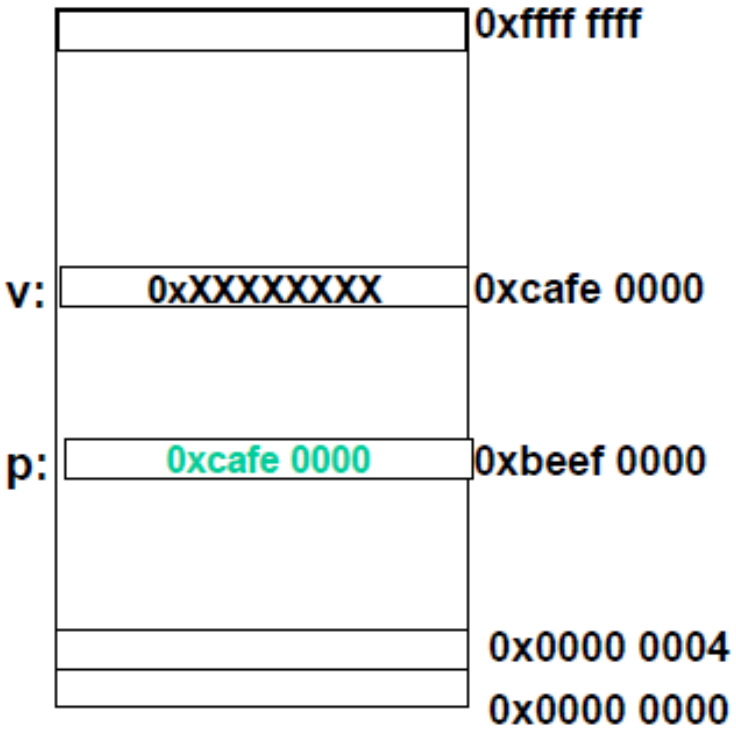
Pointer Usage Example



Memory and Pointers:

int *p, v;

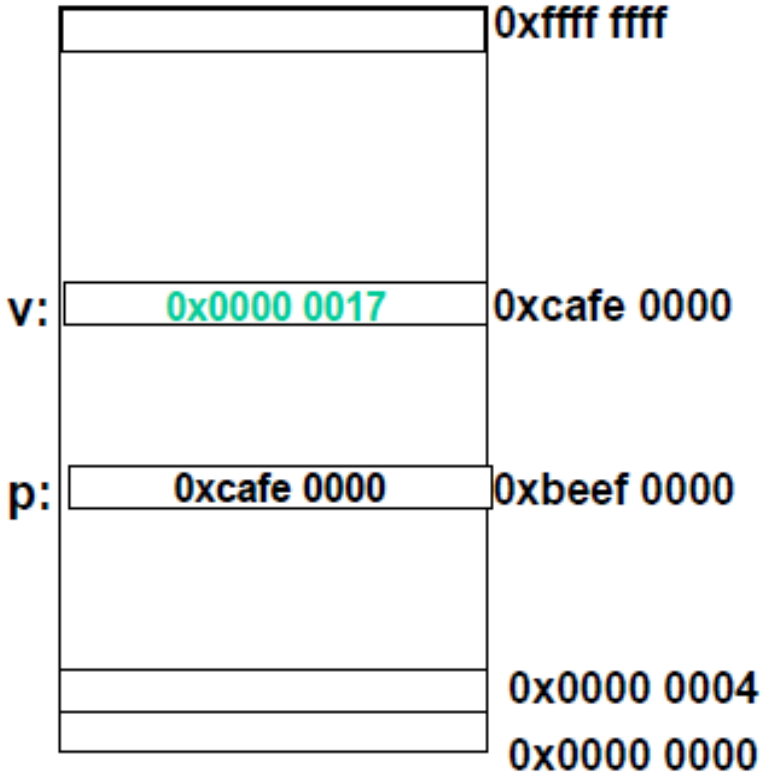
Pointer Usage Example



Memory and Pointers:

```
int *p, v;
p = &v;
```

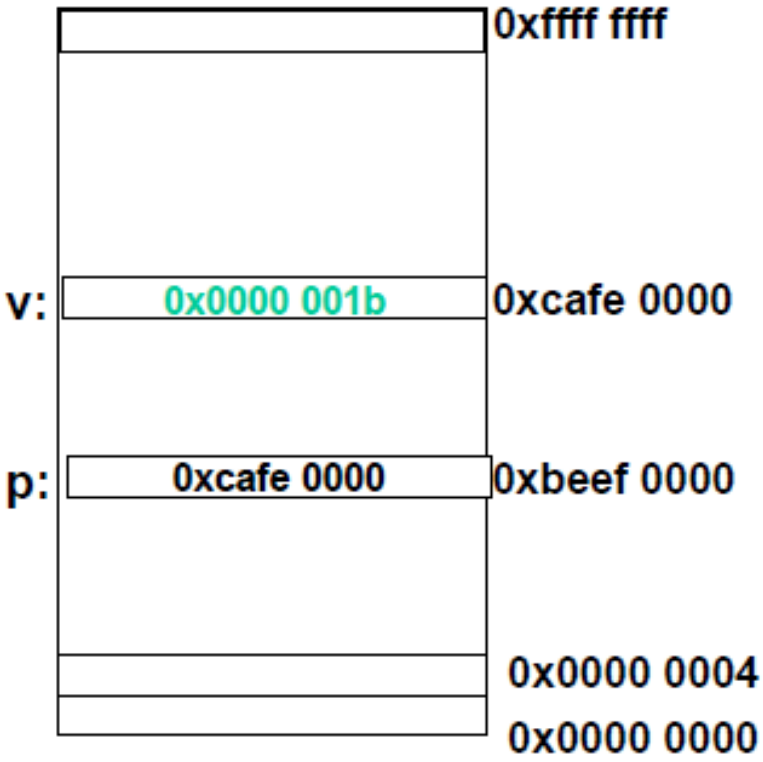
Pointer Usage Example



Memory and Pointers:

```
int *p, v;
p = &v;
v = 0x17;
```

Pointer Usage Example



Memory and Pointers:

```
int *p, v;
p = &v;
v = 0x17;
*p = *p + 4;
```

V = *p + 4

Contd.

- **Following usages are illegal:**
  - **&235**
    - **Pointing at constant.**
  - **&(a+b)**
    - **Pointing at expression.**

```
#include  <stdio.h>
main()
{
    int   a;
    float  b, c;
    double  d;
    char  ch;

    a = 10;   b = 2.5;  c = 12.36;  d = 12345.66;  ch = 'A';
    printf  ("%d is stored in location %u \n",  a,  &a) ;
    printf  ("%f is stored in location %u \n",  b,  &b) ;
    printf  ("%f is stored in location %u \n",  c,  &c) ;
    printf  ("%ld is stored in location %u \n", d,  &d) ;
    printf  ("%c is stored in location %u \n",  ch, &ch) ;
}
```

Output:

```
10 is stored in location 3221224908
2.500000 is stored in location 3221224904
12.360000 is stored in location 3221224900
12345.660000 is stored in location 3221224892
A is stored in location 3221224891
```

Things to Remember

- Pointer variables must always point to a data item of the ***same type***.

```
float   x;
int     *p;
:
p = &x;
```

➔ will result in erroneous output

- Assigning an absolute address to a pointer variable is prohibited.

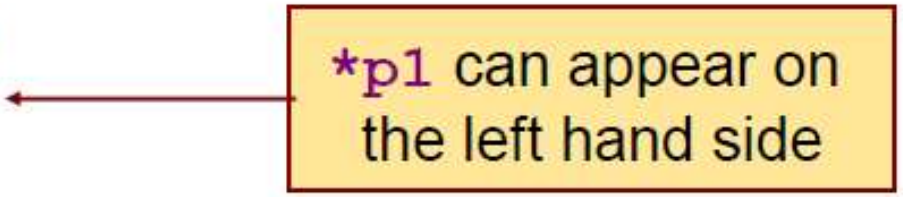
```
int    *count;
:
count = 1268;
```

Pointer Expressions

- Like other variables, pointer variables can be used in expressions.

- If p1 and p2 are two pointers, the following statements are valid:

```
sum = *p1 + *p2;
prod = *p1 * *p2;
prod = (*p1) * (*p2);
*p1 = *p1 + 2;
x = *p1 / *p2 + 5;
```



Contd.

- What are allowed in C?
  - Add an integer to a pointer.
  - Subtract an integer from a pointer.
  - Subtract one pointer from another (related).
    - If p1 and p2 are both pointers to the same array, then p2–p1 gives the number of elements between p1 and p2.

- What are not allowed?
  - Add two pointers.

```
p1 = p1 + p2;
```

  - Multiply / divide a pointer in an expression.

```
p1 = p2 / 5;
p1 = p1 - p2 * 10;
```

Scale Factor

- We have seen that an integer value can be added to or subtracted from a pointer variable.

```
int  *p1, *p2;
int  i, j;
:
p1 = p1 + 1;
p2 = p1 + j;
p2++;
p2 = p2 - (i + j);
```

  - In reality, it is not the integer value which is added/subtracted, but rather the *scale factor* times *the value*.

Contd.

Data Type	Scale Factor
char	1
int	4
float	4
double	8

- If p1 is an integer pointer, then

  p1++

  will increment the value of p1 by 4.

Example: passing arguments by reference

```
#include  <stdio.h>
main()
{
    int  a, b;
    a = 5;  b = 20;
    swap (&a, &b);
    printf ("\n a=%d, b=%d", a, b);
}

void swap (int *x, int *y)
{
    int  t;
    t = *x;
    *x = *y;
    *y = t;
}
```

Output

```
a=20, b=5
```

Pointers and Arrays

- When an array is declared,
  - The compiler allocates a *base address* and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.
  - The *base address* is the location of the first element (*index 0*) of the array.
  - The compiler also defines the array name as a *constant pointer* to the first element.

Contd.

Both `x` and `&x[0]` have the value `2500`.

`p = x;` and `p = &x[0];` are equivalent.

- We can access successive values of x by using `p++` or `p--` to move from one element to another.

- **Relationship between p and x:**

```
p   = &x[0] = 2500
p+1 = &x[1] = 2504
p+2 = &x[2] = 2508
p+3 = &x[3] = 2512
p+4 = &x[4] = 2516
```

*(p+i) gives the value of x[i]

Arrays and pointers

int a[N];

- a is a constant pointer.

- ~~a=p; ++a; a+=2;~~ illegal

- Other valid pointer expressions:
  - p+i
  - ++p
  - p+=i
  - p-q /* No of array elements between p and q */

```
x = *p++ ⇒ x = *p ; p =  p + 1;

x = (*p)++ ⇒ x = *p ; *p = *p + 1;
```

Example: function to find average

```
#include <stdio.h>
main()
{
  int x[100], k, n;

  scanf ("%d", &n);

  for (k=0; k<n; k++)
     scanf ("%d", &x[k]);

  printf  ("\nAverage is %f",
                  avg (x, n));
}
```

```
float avg (array, size)
int array[], size;
{
  int  *p, i , sum = 0;

  p = array;

  for (i=0; i<size; i++)
       sum = sum + *(p+i);

  return ((float) sum / size);
}
```

- An array name is an address, or pointer, that is fixed.

It is a CONSTANT pointer to the first element.

Arrays In Functions

- An array parameter can be declared as an array or a pointer; an array argument can be passed as a pointer.
  - Can be incremented

```
int strlen(char s[])
{


}
```

```
int strlen(char *s)
{


}
```

- **Declared arrays are only allocated while the scope is valid**

```
char *foo() {
    char string[32]; ...;
    return string;
} is incorrect
```

- For further clarification Plz refer lecture class notes and lectures on String.
- Also note the pointer to a function.