Pointers

- What are pointers?
- Uses of pointers
- Pointer syntax
- Pointer operators
- Null Pointer
- Pointer to pointer
- Pointer arithmetic's

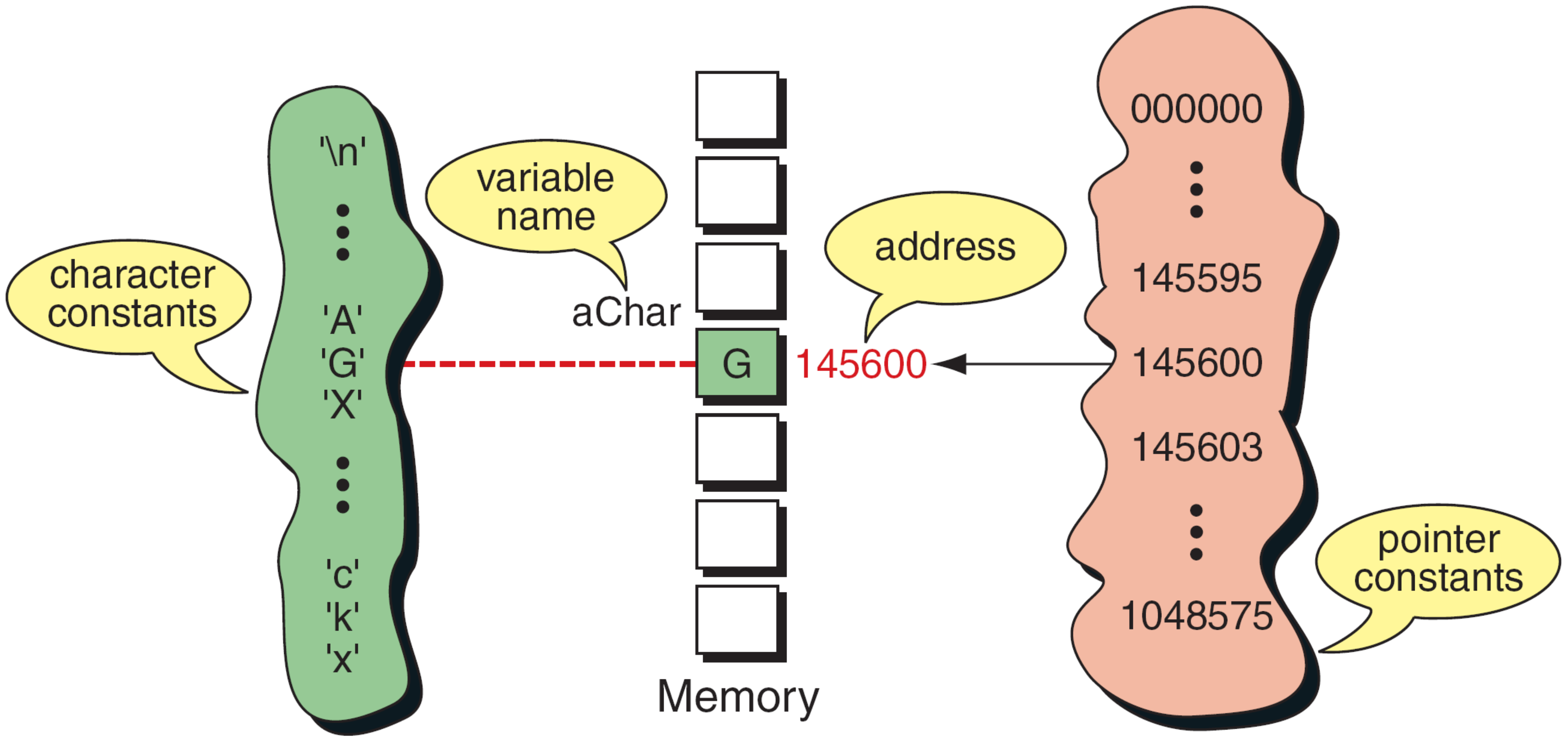


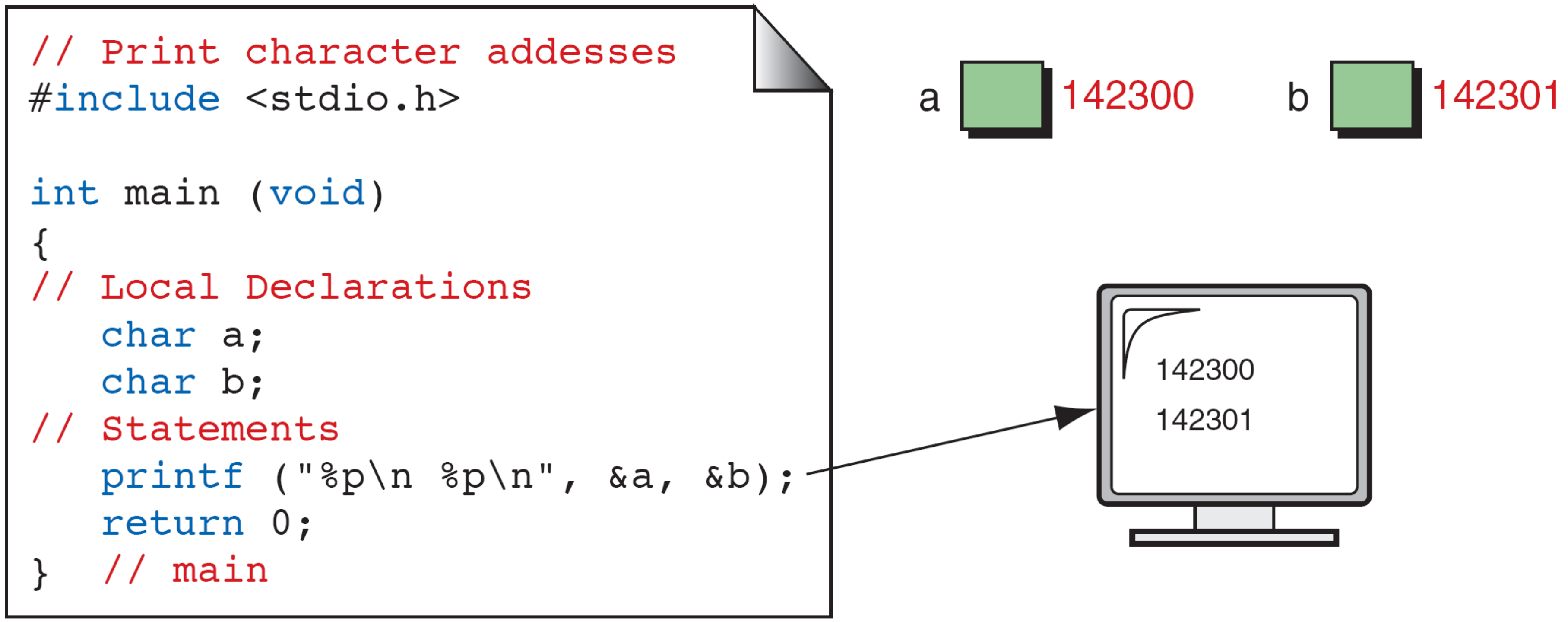
FIGURE 9-3 Pointer Constants

Note

Pointer constants, drawn from the set of addresses for a computer, exist by themselves. We cannot change them;
we can only use them.

Note

An address expression, one of the expression types in the unary expression category, consists of an ampersand (&) and a variable name.



```
// Print character addesses
#include <stdio.h>

int main (void)
{
// Local Declarations
   char a;
   char b;
// Statements
   printf ("%p\n %p\n", &a, &b);
   return 0;
}  // main
```

```
142300
142301
```

FIGURE 9-4 Print Character Addresses

Note

A variable’s address is the first byte occupied by the variable.

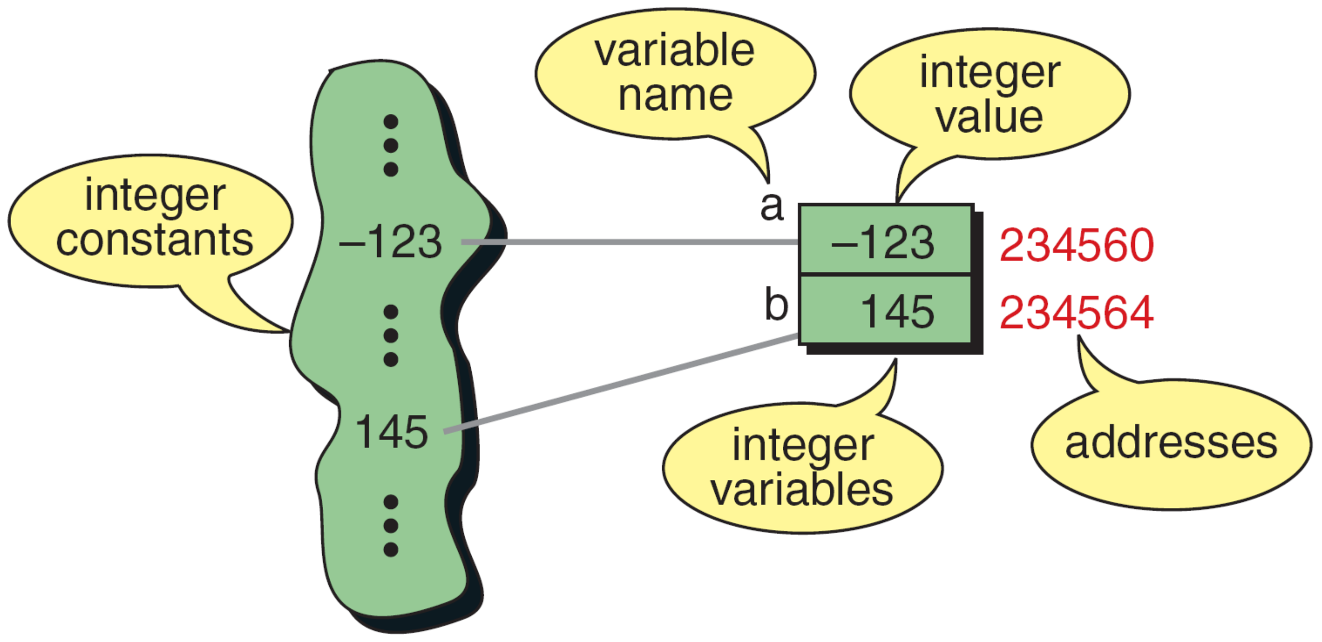
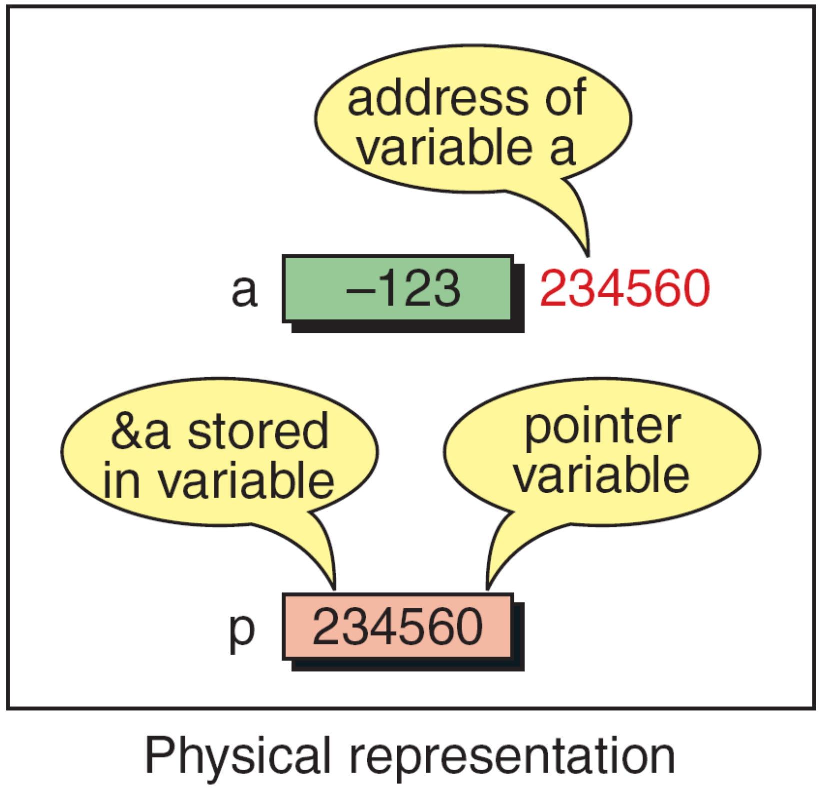


FIGURE 9-5 Integer Constants and Variables



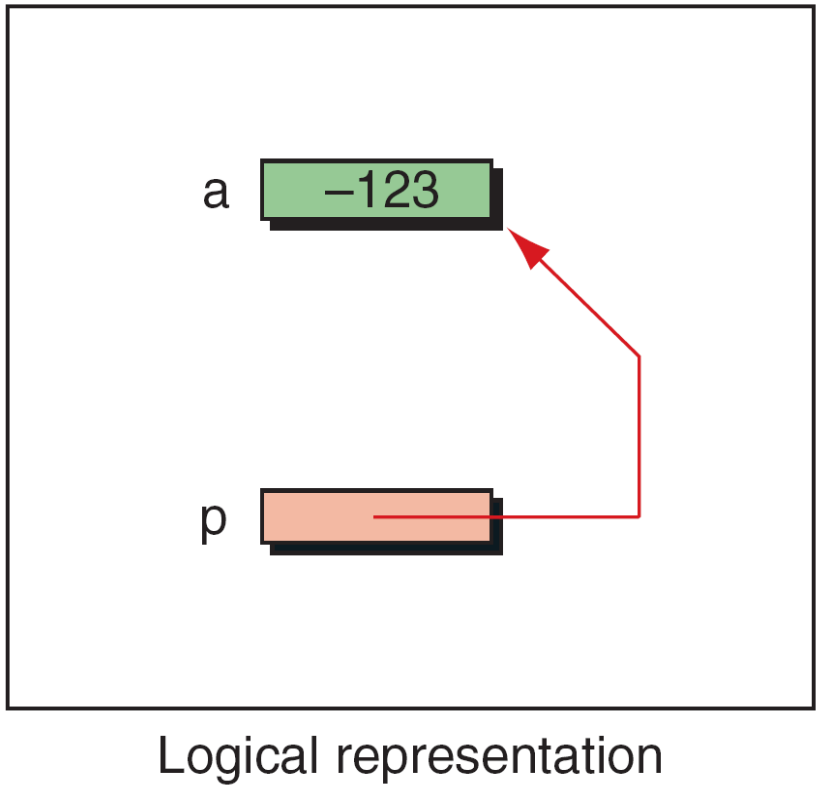


FIGURE 9-6 Pointer Variable

Let’s Start with Addresses

Sample Code

```
#include <stdio.h>
int main()
{
int var = 5;
printf("var: %d\n", var);
printf("address of var: %p", &var);
return 0;
}
```

Output

```
var: 5
address of var: 2686778
```

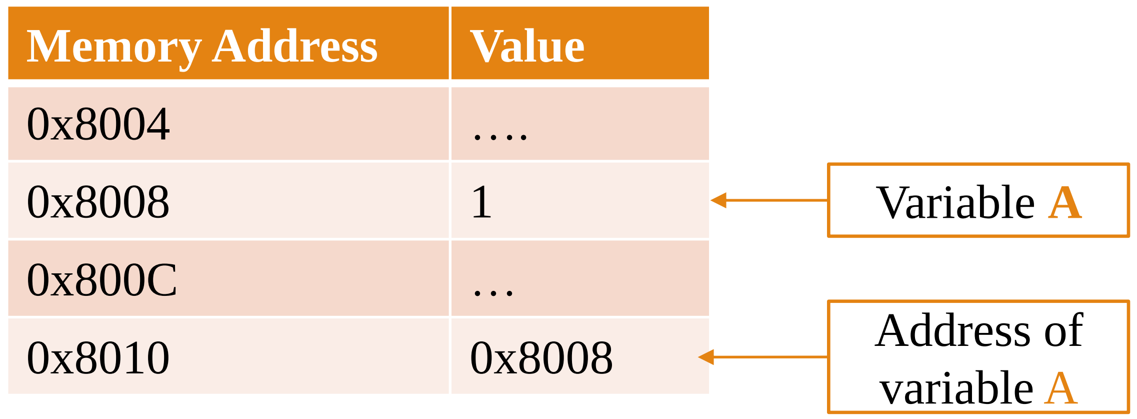
- **var** is the variable
- **&var** will give the address of the variable **var**
- A special variable that can store address is a pointer variable.

%p %u –format specifiers

What Are Pointers?

A Place holder to hold the address of the memory location.

- Address is also a number



Memory Address	Value
0x8004	….
0x8008	1
0x800C	…
0x8010	0x8008

Hence,

- A pointer is a variable whose value is the address of another variable.

Uses Of Pointers

They have a number of useful applications.

- Enables us to access a variable that is defined outside the function.
- Can be used to pass information back and forth between a function and its reference point.
- More efficient in handling data tables.
- Reduces the length and complexity of a program.
- Also increases the execution speed.

Pointer syntax

Different ways to initialize a pointer variable

int* p;

int *p; (Here **p** is a pointer)

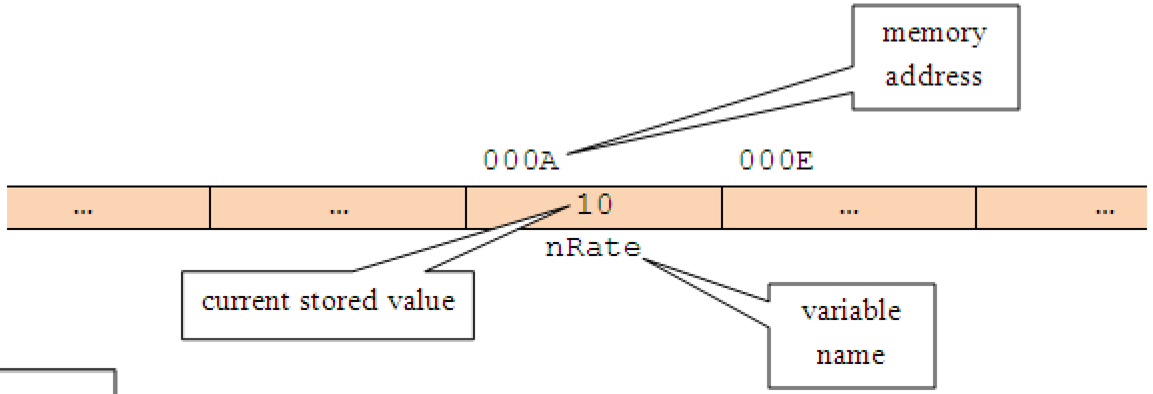
int * p;

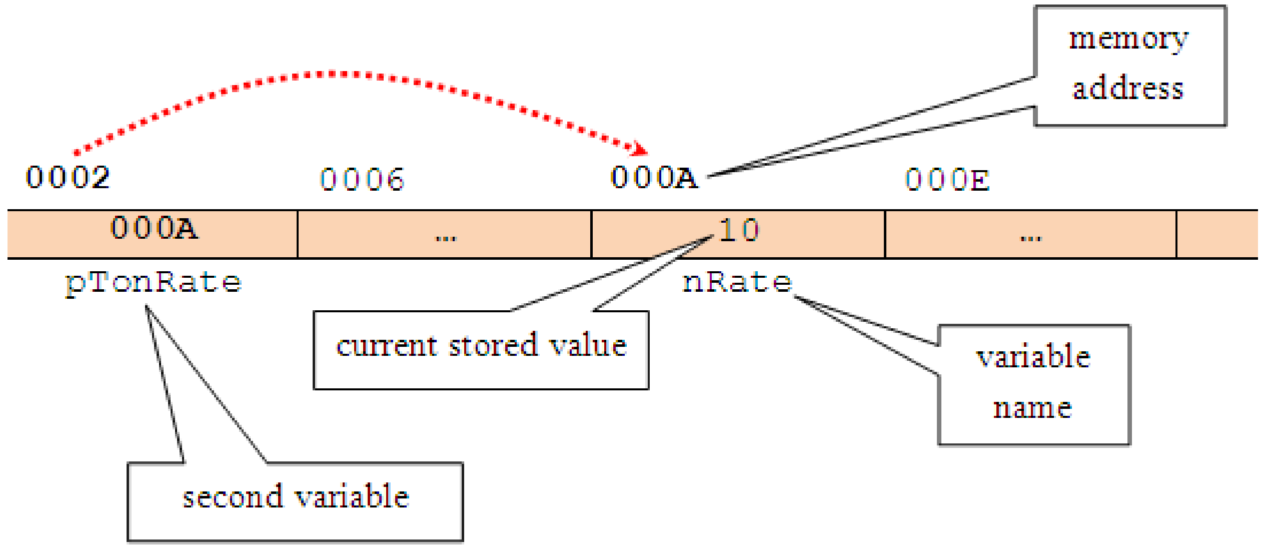
int* p1, p2; (Here **p1** is a pointer, **p2** is a normal variable)

p = &a; (Assigning the pointer **p** with address of **a)**

Contd…

```
int nRate;
nRate = 10;
```





```
int *pTonRate;
pTonRate = &nRate
```

Description

#Sample code

```
void main()
{
int a = 6;
int *p;
p = &a;
printf("Value at *p = %d\n",*p);
printf("Address stored in p =%x\n", p);
printf("Address of a =%p\n", &a);
}
```

Variable Initialization

Pointer Variable Declaration
p is a pointer
***** is a the value at operator

Store address of **a** in pointer variable **p**
& 'address of' operator

Output

```
Value at *p = 6
Address stored in p =aef13dd4
Address of a =aef13dd4
```

Note

The value of **a** can be set using **p** -

***p = 6** (after pointing p to &a)

Pointer Operators

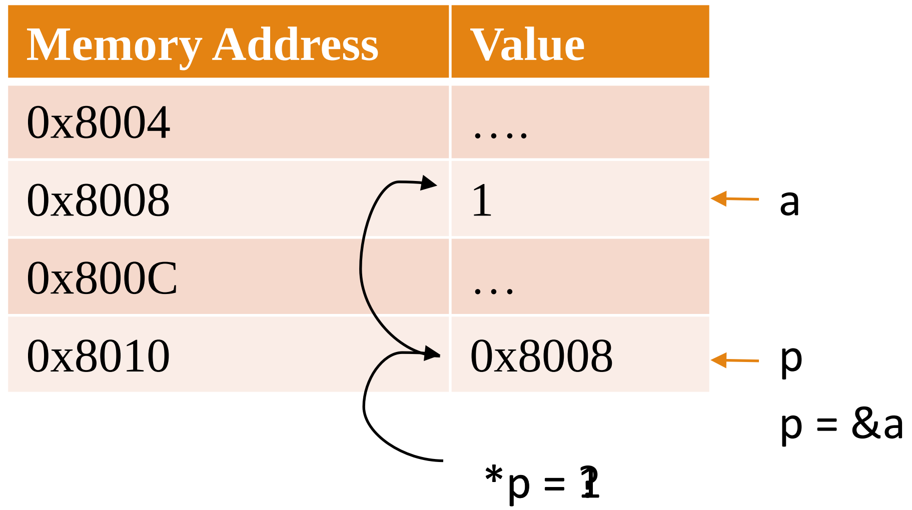
&

- *“Address of operator”*
- Provides the address of the variable

*

- *“De-referencing/indirection Operator”*

  Or
- *“Value at Operator”*
- Accesses the memory location this pointer holds the address of



Memory Address	Value
0x8004	….
0x8008	1
0x800C	…
0x8010	0x8008

Example

EX -1

```
int* pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);  // Output: 1
printf("%d", *pc);// Output: 1
```

The value of c is set to 1.
Since pc and the address of c is the same, *pc gives us 1.

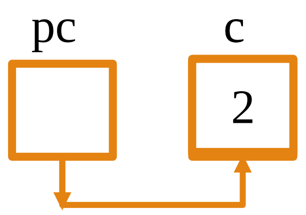
- EX -2

```
int* pc, c;
c = 5;
pc = &c;
*pc = 1;
printf("%d", *pc); // Output: 1
printf("%d", c);   // Output: 1
```

The value of *pc is set to 1.
Since pc and the address of c is the same, c gives us 1.

Example

```
#include <stdio.h>
int main()
{
   int* pc, c;
   c = 22;
   printf("Address of c: %p\n", &c);
   printf("Value of c: %d\n\n", c);
   pc = &c;
   printf("Address of pointer pc: %p\n", pc);
   printf("Content of pointer pc: %d\n\n", *pc);
   c = 11;
   printf("Address of pointer pc: %p\n", pc);
   printf("Content of pointer pc: %d\n\n", *pc);
   *pc = 2;
   printf("Address of c: %p\n", &c);
   printf("Value of c: %d\n\n", c);
   return 0; }
```



```
Address of c : 2686784
Value of c : 22

Address of pointer pc : 2686784
Content of pointer pc : 22

Address of pointer pc : 2686784
Content of pointer pc : 11
Address of c : 2686784
Value of c : 2
```

Things to Remember

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

```
float   x;
int    *p;
:                          ➔  will result in erroneous output
p = &x;
```

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

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

- Once a pointer has been assigned the address of a variable, the value of the variable can be accessed using the indirection operator (*).

```
int   a, b;
int   *p;
:
p = &a;
b = *p; // here b=a
```

NULL Pointer

- If an exact address to be assigned to a pointer is not known then assign a **NULL** value.
- This is done at the time of variable declaration.
- A pointer that is assigned NULL is called a **null** pointer.
- Example :-

```
int *ptr = NULL
```

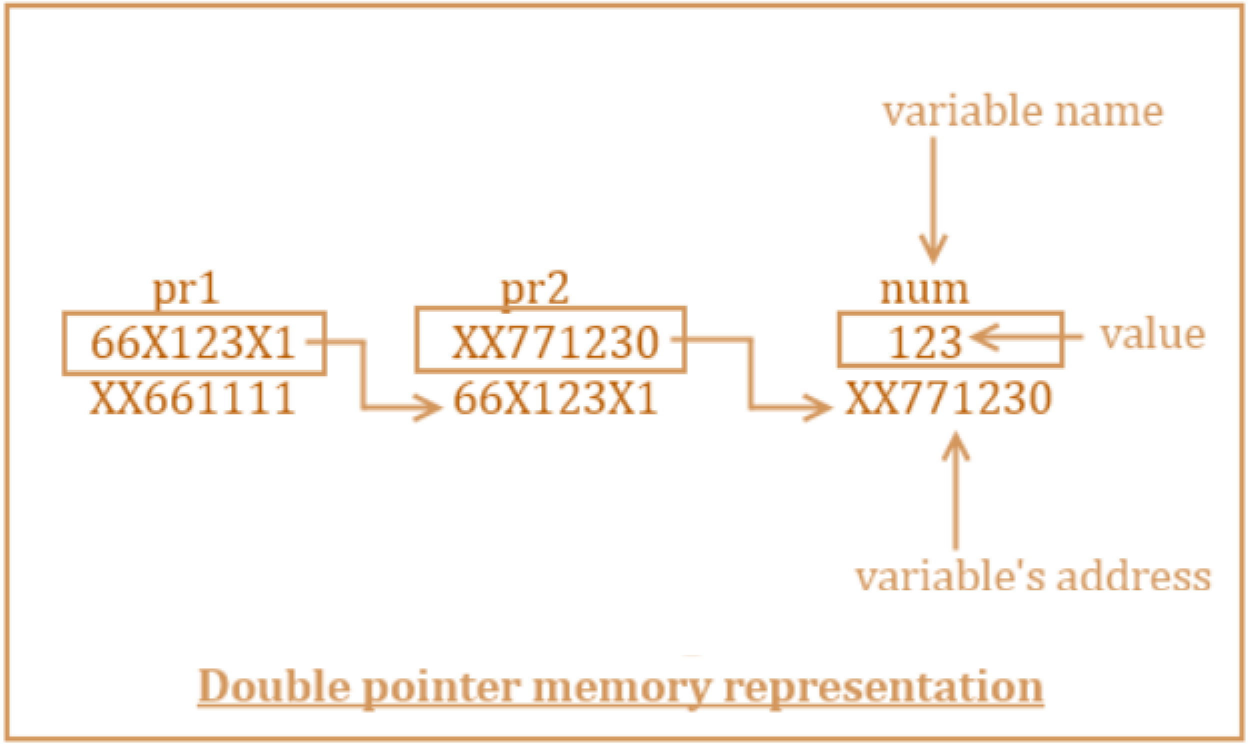
(The value of ptr is 0)

Pointer to Pointer

Pointer to pointer is a chain of pointers.
It is declared using an extra ‘*’.

int **var;

```
int num = 123
int *pr2;
int **pr1;
pr2 = &num
pr1 = &pr2
```

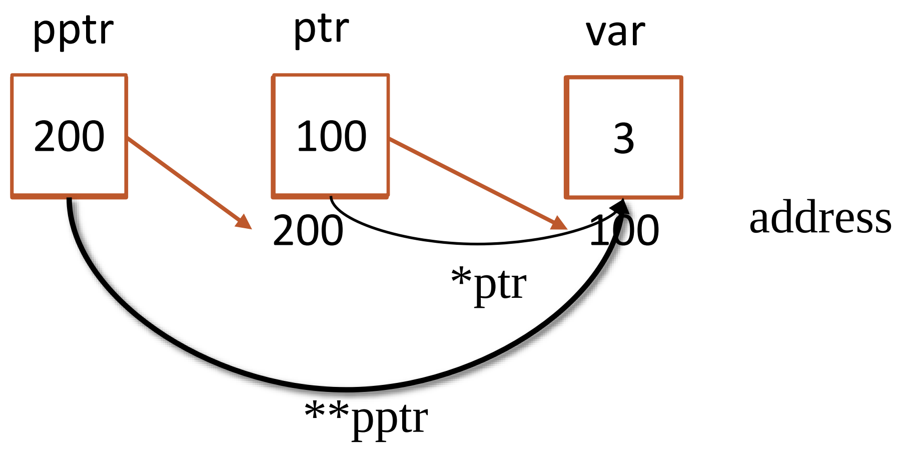


Example

Sample Code

Let 100 be the address of var
Let 200 be the address of ptr

```
#include <stdio.h>
int main () {
   int  var;
   int  *ptr;
   int  **pptr;
   var = 3 ;
   ptr = &var;
   pptr = &ptr;
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);
   return 0;
}
```



Output

```
Value of var = 3
Value available at *ptr =3
Value available at **pptr =3
```

Pointer Arithmetic

- As pointer’s are numbers arithmetic operations can be performed.
- The four arithmetic operations are:

  ++, --, +, and –
- Incrementing/ Decrementing operator
- – Add/subtract an integer to a pointer to point to a different location
- What are not allowed?
  - Adding two pointers.

    p1 = p1 + p2 ;
  - Multiply / divide a pointer in an expression.

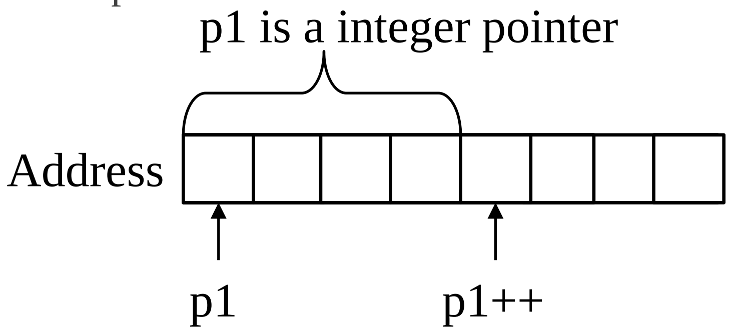
    p1 = p2 / 5 ;

    p1 = p1 – p2 * 10 ;

Contd…

- 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.

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.

Note that only integral values can be added or subtracted from a pointer. We can also subtract or compare two pointers of same type.

Example

Sample Code

```
#include <stdio.h>
int main()
{
float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
float *ptr1 = &arr[0];
float *ptr2 = ptr1 + 3;
printf("%d", ptr2-ptr1);
return 0;
}
```

Output

```
3
```

Explanation

- Generally if an integer value ‘x’ is added to pointer ‘p’ then resultant value is p+ x*(sizeof(p)).
- If ptr is 100, then ptr2 = 100 + 3*(4) = 112, and hence address belongs to arr[3].
- ptr2 – ptr1 = (ptr1+3 –ptr1) = 3

(which basically means no. of elements between ptr2 and ptr1)

Question-1

What is the output ?

Sample Code

```
void main()
{
float a=6, *p;
p = &a;
printf("%f",*p);
}
```

Options

A) 6
B) Error
C) Garbage value
D) None of the above

Solution

D) None of the above
Beacause *p points to a float value , the output will be 6.000000

Question-2

What is the output ?

Sample Code

```
void main()
{
int a=6, *p;
p = &a;
printf("%d",*(&a));
}
```

Options

A) 6
B) Error
C) Garbage value
D) None of the above

Solution

A) 6
First the address of ‘**a**’ is taken then the value present in there is printed

Question-3

What is the output ?

Sample Code

```
void main()
{
int a=6,*p;
p = &a;
printf("%d",**(&p));
}
```

Options

A) 6
B) Error
C) Garbage value
D) None of the above

Solution

A) 6
&(*p) will point to p and *p will point to a that is 6

Question-4

What is the output ?

Sample Code

```
#include <stdio.h>
void main()
{
int a=6,*p;
printf("%d",*p);
}
```

Options

A) 6
B) Address of ‘a’
C) Error
D) None of the above

Solution

D) None of the above
Gives Segmentation fault , because as p is not initialized it gives a segementation fault

Question-5

What is the output ?

Sample Code

```
#include <stdio.h>
void main()
{
int a = NULL,*c=NULL;
c = &a;
printf("%d",*c);
}
```

Options

A) 0
B) Error
C) Garbage value
D) None of the above

Solution

A) 0
The integer equivalent of **NULL** is 0

Question-6

What is the output ?

Sample Code

```
#include <stdio.h>
void main()
{
int a=6,*d, *c;
d = &a;
c = d;
printf("%d",*c);
printf("%d",*d);
}
```

Options

A) 6 6
B) Error
C) Garbage value
D) None of the above

Solution

A) 6 6
Using the assignment operator to initiate the pointer will not change the working of the pointer

Question-7

What is the output ?

Sample Code

```
#include <stdio.h>
void main()
{
int a=2, b=4,*c,*d;
c = &a;
d = &b;
d = c;
printf("%d",*c);
printf("%d",*d);
}
```

Options

A) 2 2
B) Error
C) Garbage value
D) None of the above

Solution

A) 2 2
Updating the pointer to a new location will change it completely to a new memory location. So both **‘c’** and **‘d’** points to **‘a’ .**

Question-8

What is the output ?

Sample Code

```
int main()
{
    int *ptr;
    int x;
    ptr = &x;
    *ptr = 0;
    printf(" x = %dn", x);
    printf(" *ptr = %dn", *ptr);
    *ptr += 5;
    printf(" x  = %dn", x);
    printf(" *ptr = %dn", *ptr);
    (*ptr)++;
    printf(" x = %dn", x);
    printf(" *ptr = %dn", *ptr);
    return 0;
}
```

Solution

```
x = 0
*ptr = 0
x = 5
*ptr = 5
x = 6
*ptr = 6
```