Addresses

- Each user-defined variable is stored at a location in the memory and has a value. The location is uniquely identified by an address.
- The address is retrieved by the “&” sign.

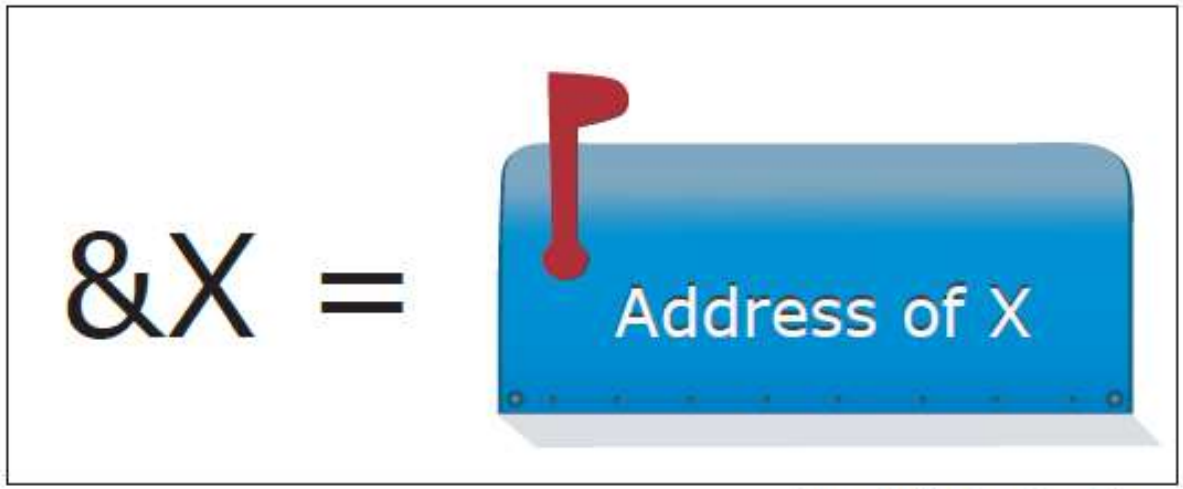


Image by MIT OpenCourseWare.

Pointers

- A pointer is a variable that “points” to the block of memory that another variable represents. In other words, it stores the memory address of another variable.
- The other variable may be of type int, char, array, structure, function or any other.

Variables

- Normal variables
  - Contain a specific value (direct reference)
  - Example : count

- Pointer variables
  - Contain memory addresses as their values
  - Example: countPtr

Pointer Variable Declarations

- Pointer declarations
- Declaration: data_type *pointer_name;
  - '*' used with pointer variables

```
int  *myPtr;
float *Ptr;
Char *strPtr;
```

  - Multiple pointers require using a * before each variable declaration

```
int *myPtr1, *myPtr2;
```

Pointer Variable Initialization

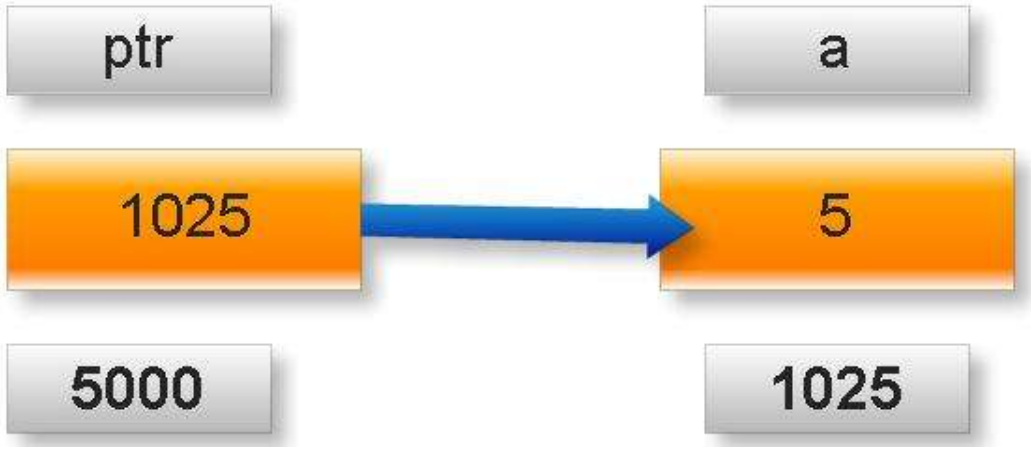
- Initialize pointers to:
  - **0 or NULL,**
    - points to nothing (**NULL** preferred)
    - **Example: int *yPtr;**
  - an address of a variable
    - points to an address of a user defined variable in the memory
    - **Example:**

```
int y = 5;
int *yPtr;
yPtr = &y;
```

  - an address (absolute value)
    - points to a certain place/address in the memory
    - **Example: int *ptr=(int *)1000;**

Variables vs Pointer Variables

```
int a=5;
int * ptr;
ptr=&a;
```



About variable a:

- 1. Name of variable : a
- 2. Value of variable: 5
- 3. Address: 1025 (assume)

About variable ptr:

- 4. Name of variable: ptr
- 5. Value of variable: 1025
- 6. Address: 5000 (assume)

Dereferencing = Using Addresses

- *Dereferencing* a *pointer means* getting the value that is stored in the memory location pointed by the *pointer*. The operator * is used to do this, and is called the *dereferencing* operator.
- Given pointer ptr, to get value at that address, write *ptr after the declaration.
- Example:

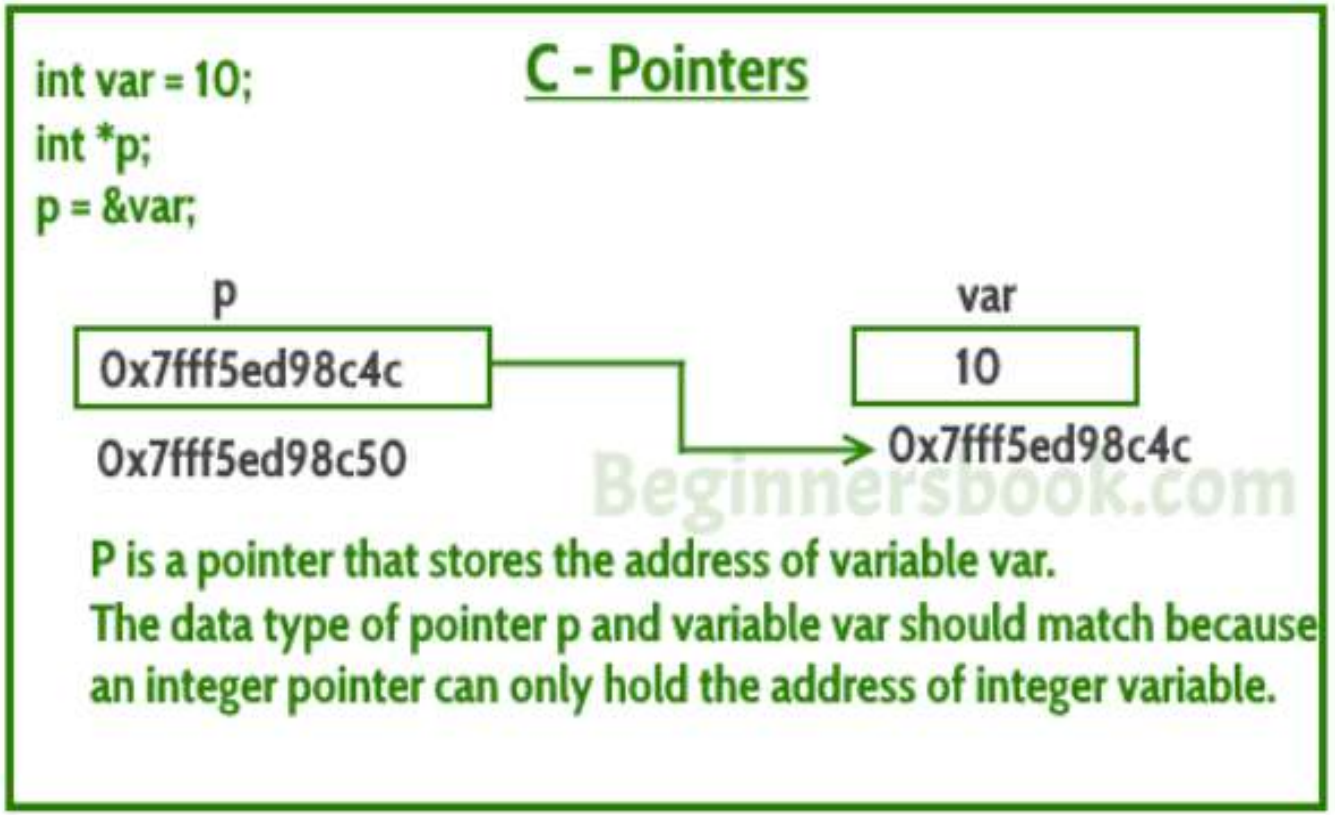
```
int x = 5;
int *ptr = &x;
*ptr = 6;
/* Access x via ptr, and changes it to 6,
equivalent to x = 6 */
printf("%d", x); // Will print 6 now
```

Example

```c
int main()
{
   /* Pointer of integer type, this can hold the
    * address of a integer type variable.
    */
   int *p;

   int var = 10;

   /* Assigning the address of variable var to the pointer
    * p. The p can hold the address of var because var is
    * an integer type variable.
    */
   p= &var;

   printf("Value of variable var is: %d", var);
   printf("\nValue of variable var is: %d", *p);
   printf("\nAddress of variable var is: %p", &var);
   printf("\nAddress of variable var is: %p", p);
   printf("\nAddress of pointer p is: %p", &p);
   return 0;
}
```

Example (cont.) – output

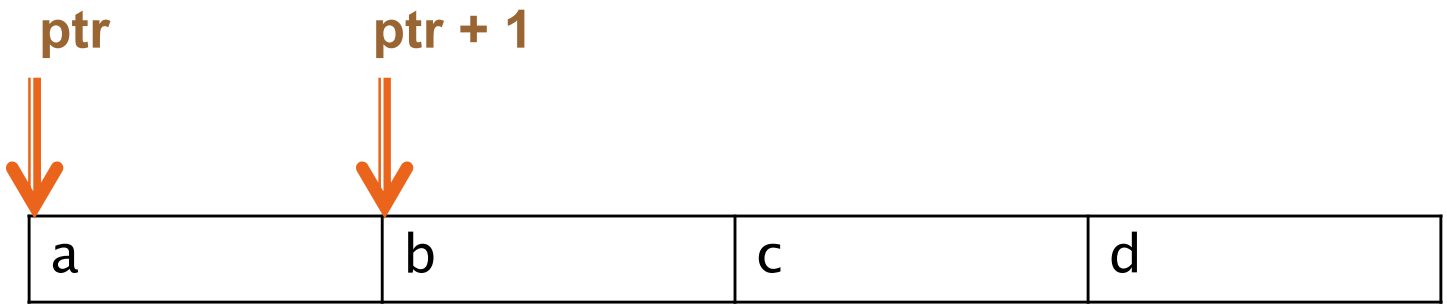
```
Value of variable var is: 10
Value of variable var is: 10
Address of variable var is: 0x7fff5ed98c4c
Address of variable var is: 0x7fff5ed98c4c
Address of pointer p is: 0x7fff5ed98c50
```



Pointer Arithmetic

- Can do math on pointers
- Example:

  char* ptr;



- ptr+i = ptr + i * sizeof(data_type of ptr)

Data type sizes

Name	Description	Size*	Range*
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

Pointer Arithmetic

- Arithmetic operations can be performed on pointers
  - Increment/decrement pointer (++ or --)
  - Add an integer to a pointer
    (+ or += , - or -=)
  - Pointers may be subtracted from each other
  - Operations on a single pointer are meaningless unless performed on an array because it will access unrelated variables randomly.

Pointer Expressions and Arithmetic

- Pointer comparison (<, == , >)
  - See which pointer points to the higher numbered array element (index), subsequently higher address
  - Also, checks if a pointer points to **`nothing`**

Long Form	Short Form
if (ptr == NULL)	if (!ptr)
if (ptr != NULL)	if (ptr)

Pointer Arithmetic-Example

```
#include<stdio.h>
int main()
{
int *ptr=( int *)1000;
ptr=ptr+1;
printf(" %d",ptr);
return 0;
}
```

Output

1004

Pointer Arithmetic-Example

```
#include<stdio.h>
int main()
{
double *p=(double *)1000;
p=p+3;
printf(" %d",p);
return 0;
}
```

Output

```
1024
```

Pointer Arithmetic-Example

```
#include<stdio.h>
int main()
{
int *p=(int *)1000;
int *temp;
temp=p;
p=p+2;
printf("%d %d\n",temp,p);
printf("difference= %d",p-temp);
return 0;
}
```

Output

```
1000 1008
Difference= 2
```

Pointer Arithmetic-Example

```
#include<stdio.h>
int main()
{
float *p=(float *)1000;
float *q=(float *)2000;
printf("Difference= %d",q-p);
return 0;
}
```

Output

```
Difference= 250
```