Pointers
name for a memory location

A variable is nothing more than a convenient

The variable is stored at specific memory address

int nrate = 10;

Variable Memory Snapshot
This statement specifies that the value read is to be stored at the address of x.

Example:

```
scanf("%f", &x);
```

A variable can be referenced using the address operator `&`.
If $a$ is a variable at location 100 in memory, then $p$ would have the value 100 (a's address).

A pointer is a variable that holds the address of a memory location.

If a variable holds the address of another variable, $d$ then is said to point to variable $b$, then $b$ is said to point to $d$.
Integer cannot also point to a floating point variable. (i.e., a pointer defined to point to an integer must be specified, i.e., a pointer defined to point to an integer will point to an integer variable to which it will refer.) пт is a pointer to an integer

Example: *ptr = int a, b; *ptr = a

Referencing operator. The asterisk is called the "indirection operator" or the dereference operator. The pointer variables are declared using an asterisk (*). The

How to declare a pointer variable
holding an address
declaring a pointer creates a variable capable of

example

neither variable has not been initialized in the above
the variable dptr is declared to point to a double

the variable int is declared to point to an int

double *dptr;

int *iptr;

Example
Memory requirements depend on the pointers.

Data items
- They make it easy to represent relationships between
  as your program is running
- They make it possible to get new memory dynamically
  programs
- They facilitate sharing between different parts of
  compact way
- They allow you to refer to large data structures in a

Why Pointers?
Example
an int.
be a pointer to int. q is declared to be
In the above example p is declared to
The * operator does not distribute.
int p, q;
When using the form

More about declaring pointers
The operator * in front of a pointer variable returns the contents stored at that address.

The operator & in front of an ordinary variable produces the address of that variable.

The operator & in front of an ordinary variable produces the address of that variable.

The same type as the left operand can be any expression that evaluates to the right operand.

The assignment operator (=) is defined for pointers.

Assigning values to a pointer
```c
if (x) {
    printf("x = \%d\n", x);
}

int *ptr;
ptr = 8;

int *int = 6;
}```
Exercise
Exercise

Given a memory snapshot after each set of assignment statements,

\[
a = \_ptr, \\
:\text{int } a=1, b=2, \_ptr=\&b;
\]

\[
ptr = \&b, \\
:\text{int } a=1, b=2, \_ptr;
\]
null pointer

When you initialize a pointer, you initialize it to NULL, a pointer variable whose value is NULL is not pointing to anything that can be accessed.

A pointer can be compared or compared to the integer zero, or equivalently, to the symbolic constant `NULL`, which is defined in `<stdio.h>`.

A pointer can be assigned or compared to the integer NULL.
Before you use a pointer make sure it points to null.

```c
double *dPtr = NULL;
char *s = 0;
int *iPtr = 0;
```

**Example**
The memory snapshot after these statements are executed is:

```c
ptr2 = ptr1;
ptr1 = &x;
/* Assign both pointers to point to x. */
int x = 5, y = 8, *ptr1, *ptr2;
/* Declare and initialize variables. */
```

A pointer can point to only one location at a time, but several pointers can point to the same location.

**Pointer Assignments**
next value in memory

when applied to pointers, ++ means increment pointer to point to

if $p$ is defined as double, *$p$ will be incremented by 8 (system dependent)

(if $p$ is defined as int, *$p$ will be incremented by 4 (system dependent)

Example: $p++$; to point to next variable

being pointed to

Arithmetic is performed relative to the variable type

Arithmetic operations are supported

Four arithmetic operations are supported

Pointer Arithmetic
such as an array.

- compare two pointers that are pointing to a common object:
  \( (b_* == d_*) \)
  is this equivalent to:
  \( (b == d) \)

- check if two pointers are pointing to the same object
- check for null pointer \( (p == nullptr) \)

Common comparisons are:

- You may compare pointers using relational operators

Comparing Pointers
Exercise

Show the memory snapshot after the following operations:

```c
ptr3 = &y;  // make ptr3 point to y
ptr1 = &x;  // make ptr1 point to x
int *ptr1, *ptr2, *ptr3;
int x=2, y=3, temp;
temp = y;
```