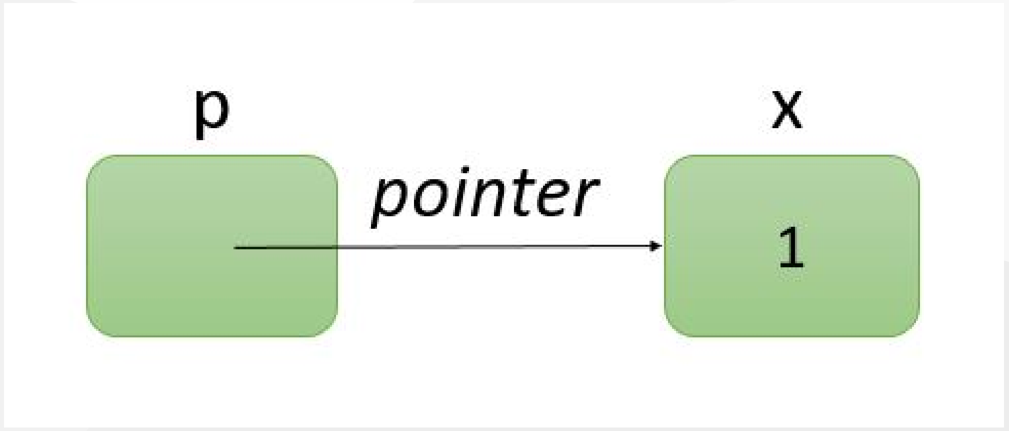
Pointers

- Passing arguments to functions by reference
- Create and manipulate dynamic data structures
    - Such as linked lists, queues, stacks and trees
- Pointer variables values are memory addresses
- Pointers indirectly reference a value



Create a Pointer

- Can only point to an object to the same type
- * in the definition, indicates the variable is a pointer
- When defined, should be initialized with NULL or assigned a value

```
// ptr_p is a pointer of type int
int *ptr_p;
// count is a variable of type int
int count;
// assign the pointer the count variable
ptr_p = &count;
```

Pointer Operators

- * indrection or reference operator
- & a unary operator that returns the address

```c
int num = 5;
int *pnum;
pnum = &num;
int *ppnum;
ppnum = &pnum;
printf("%d", num);
printf("%d", *pnum);
printf("%d", **ppnum);
printf("%p", &num);
printf("%p", &pnum);
printf("%p", &ppnum);
printf("%p", pnum);
printf("%p", ppnum);
```

Operators in Action

```
int x = 2, y = 4;
int *ip;
ip = &x;
y = *ip;
x += y;
y = *ip;
```

Var	Add	Value	Var	Add	Value	Var	Add	Value
x	x100	2	y	x112	4	ip	x102	
x	x100	2	y	x112	4	ip	x102	x100
x	x100	2	y	x112	2	ip	x102	x100
x	x100	4	y	x112	2	ip	x102	x100
x	x100	4	y	x112	4	ip	x102	x100

Pointers and Functions

- C Language conventionally passes arguments to functions using Pass-by-Value
- Functions may require the ability to modify variables
- Pointers **simulate** Pass-by-Reference
- Function should receive the **addresses** of arguments
- Referencing arguments avoids the memory overhead of copying variables to a function and copying them back at the functions conclusion

```c
int a = 9;                          // &a
char ch;                            // &ch
int array[] = {74,52,1,32};         // array or &array[0]
char *pch;
pch = &ch;                          // pch
```

```c
void main () {
  int number1 = 3, number2 = 9;
  int *pnumber;
  pnumber = &number2;
  cubethenumber(number1);
  printf("%d\n", number1);
  number1 = cubethenumber(number1);
  printf("%d\n", number1);
  cubethenumber(number1);
  printf("%d\n", number1);
  cubebyreference(pnumber);
  printf("%d, %d\n", number2, *pnumber);
}

int cubethenumber(int n)
{ return n * n * n; }

int cubebyreference(int *n)
{ return *n * *n * *n; }
```

Convert a String to Uppercase

```c
#include <stdio.h>
#include <ctype.h>

void convertToUppercase(char *pString);

void main() {
  char string[] = "cHaRaCters and $54.69";
  convertToUppercase(string);
}

void convertToUppercase(char *pString) {
  while (*pString != '\0') {
    *pString = toupper(*pString);
    ++pString;
  }
}
```

Print a String One Character at a Time

```c
#include <stdio.h>

void printCharacters(const char *pString);

void main() {
  char string[] = "print characters for a string";
  printCharacters(string);
  printf("\n");
}

void convertToUppercase(char *pString) {
  for (; *pString != '\0'; ++pString) {
    printf("%c", *pString);
  }
}
```

Bubble Sort Using Pass-by-Reference

```c
{
  int hold = array[j];
  array[j] = array[j + 1];
  array[j + 1] = hold;
}

swap(&array[j], &array[j+1]);

void swap(int *element1, int *element2) {
  int hold = *element1;
  *element1 = *element2;
  *element2 = hold;
}
```

sizeof Operator

- Special unary operator used to find the size
- Variables type determine the size
- Array size can be calculated from this information

```
#define SIZE 20
float array1[SIZE];           double array2[SIZE];

numelements1 = sizeof(array1) / getFloatSize(array1);
numelements2 = sizeof(array2) / getDoubleSize(array2);

size_t getFloatSize(float *ptr)
{ return sizeof(ptr); }

size_t getDoubleSize(double *ptr)
{ return sizeof(ptr); }
```