Introduction To C

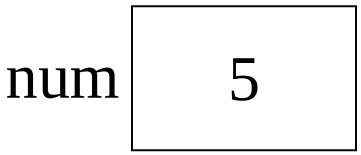
- C Stuff
  - Pointers
  - Parameter Passing
  - Input & Output
  - Arrays
  - Strings

Functions in C

```
#include <stdio.h>

int addUp(int a, int b)
{
   int result;
   result = a + b;
   return result;
}

int main()
{
   int r;
   r = addUp(3, 5);
   printf(“%d”, r);
}
```

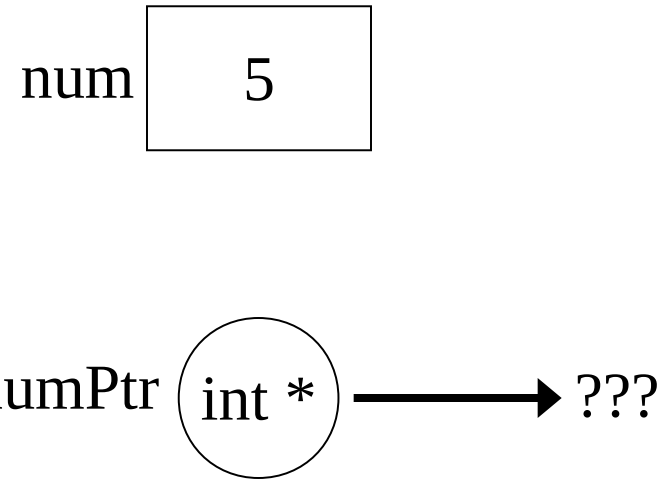
C Pointers

- Declaring an int:
  - `int num = 5;`



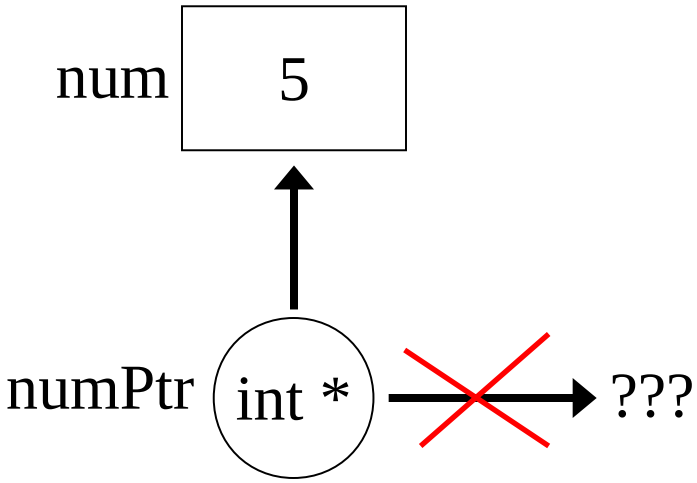
C Pointers

- Declaring an int:
  - `int num = 5;`
- Declaring a pointer to an int:
  - `int* numPtr;`



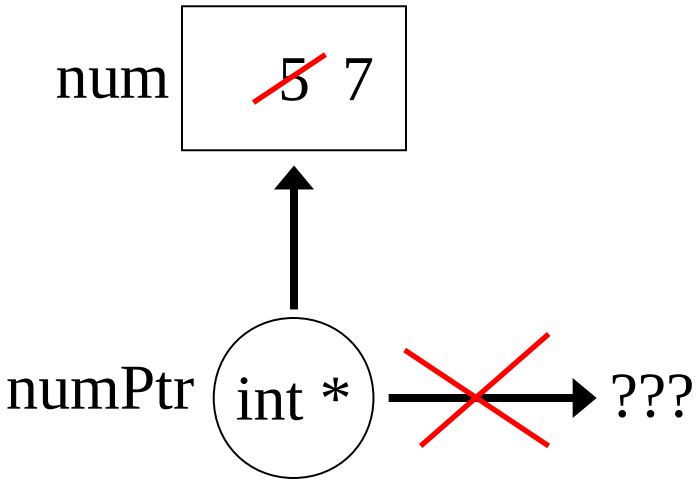
C Pointers

- Declaring an int:
  - `int num = 5;`
- Declaring a pointer to an int:
  - `int * numPtr;`
- Assigning the pointer
  - `numPtr = &num;`
  - &num is the address of the int num



C Pointers

- Dereferencing a pointer
  - Use the * operator
  - `*numPtr = 7;`



BINKY!

C Parameter Passing

- All parameter passing in C is **pass-by-value.**
- This means a new (temporary) variable is created when the function is called, and the passed parameter's value is copied into this new variable.

C Parameter Passing

- Example:

```
void addOne(int x) // takes an int variable
{
  x++;        //3
}

void main()
{
  int y = 3; //1
  addOne(y); //2
}            //4
```

What is y's value at step 4???

C Parameter Passing

- Example:

```
void addOne(int x) // takes an int variable
{
  x++;          //3
}

void main()
{
  int y = 3; //1
  addOne(y); //2
}            //4
```

What is y's value at step 4???

C Parameter Passing

- Example:

```
void addOne(int x) // takes an int variable
{
  x++;          //3
}

void main()
{
  int y = 3; //1
  addOne(y); //2
}            //4
```

x | 3 After function call (2)

y | 3

What is y's value at step 4???

C Parameter Passing

- Example:

```
void addOne(int x) // takes an int variable
{
  x++;          //3
}

void main()
{
  int y = 3; //1
  addOne(y); //2
}            //4
```

x | 4 After increment (3)

y | 3

What is y's value at step 4???

C Parameter Passing

- Example:

```
void addOne(int x) // takes an int variable
{
  x++;        //3
}

void main()
{
  int y = 3; //1
  addOne(y); //2
}            //4
```

y | 3 | At step (4)

What is y's value at step 4???

C Parameter Passing

- By using pointers, we can implement **pass-by-reference**.
- **Pass-by-reference** means the address of the variable is passed to the function

C Parameter Passing

- Example:

```
//takes a pointer to an int
void addOneByRef(int* x)
{
     (*x)++;
}

void main()
{
     int y = 3;
     addOneByRef(&y);  //pass the address of y
}
```

What is y's value now?

C Parameter Passing

- Example:

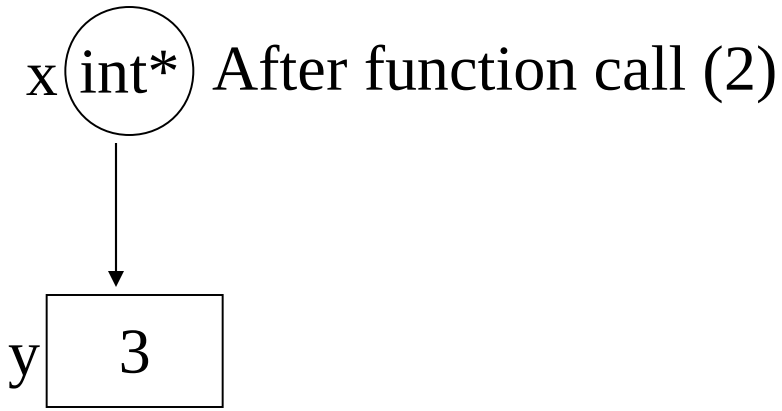
```
//takes a pointer to an int
void addOneByRef(int* x)
{
     (*x)++;              //3
}

void main()
{
     int y = 3;           //1
     addOneByRef(&y); //2
}
```

What is y's value now?

C Parameter Passing

- Example:

```
//takes a pointer to an int
void addOneByRef(int* x)
{
     (*x)++;                  //3
}

void main()
{
     int y = 3;               //1
     addOneByRef(&y); //2
}
```

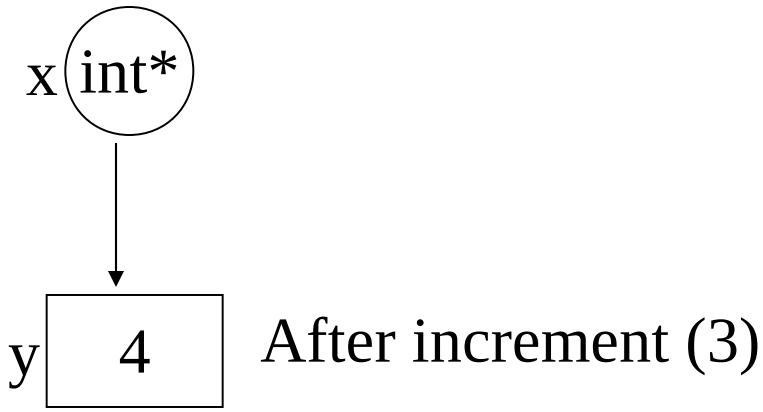


What is y's value now?

C Parameter Passing

- Example:

```
//takes a pointer to an int
void addOneByRef(int* x)
{
     (*x)++;              //3
}

void main()
{
     int y = 3;           //1
     addOneByRef(&y); //2
}
```



What is y's value now?

Input / Output

- Displaying to screen:

  printf(“This will show up on screen”);
- Displaying variable values

  int x = 5;

  printf(“ x has the value %d”, x);
- %d indicates a decimal integer
- Several others (%c is char, etc.)

Input / Output

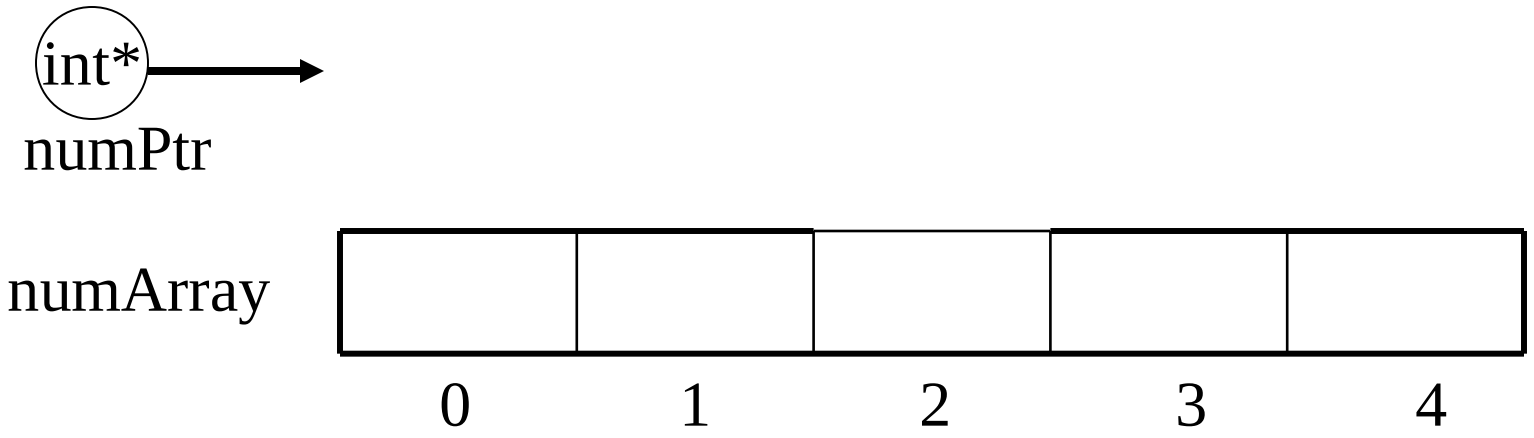
- Getting user input
- Use scanf – similar to printf

```
int x;
scanf("%d", &x);
```

- Why use the address of x???

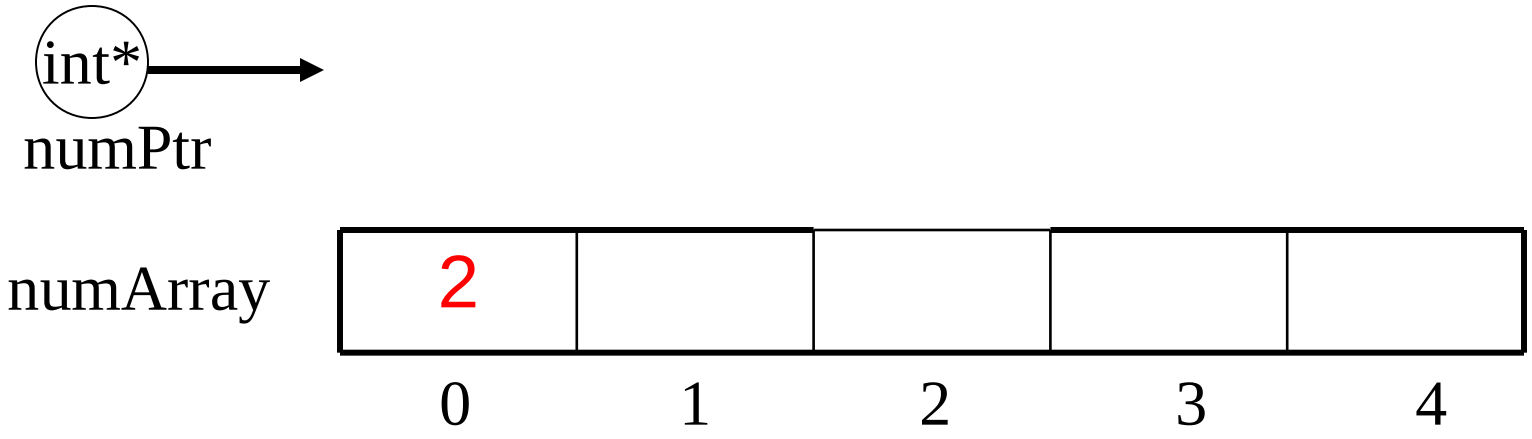
Arrays in C

```
int numArray[5];     //array of 5 ints
int * numPtr;        //pointer to an int
```



Arrays in C

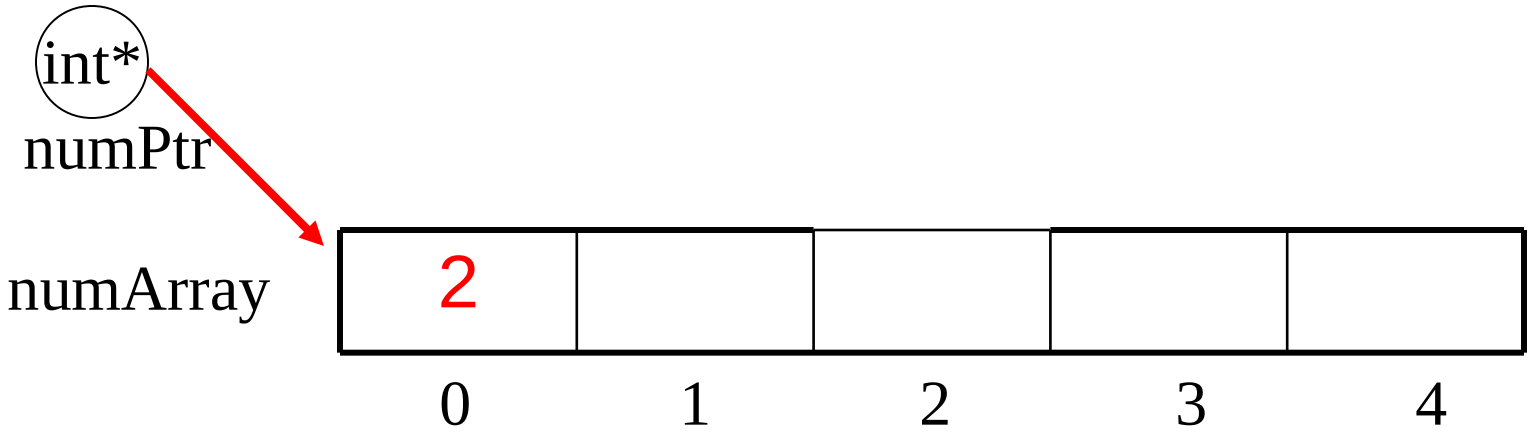
```
int numArray[5];        //array of 5 ints
int * numPtr;           //pointer to an int

numArray[0] = 2;     //Assign a value
```



Arrays in C

```
int numArray[5];        //array of 5 ints
int * numPtr;           //pointer to an int

numArray[0] = 2;        //Assign a value
```

```
numPtr = numArray;  //assign pointer
//Now, *numPtr == numArray[0] == 2
//          == numPtr[0]
```

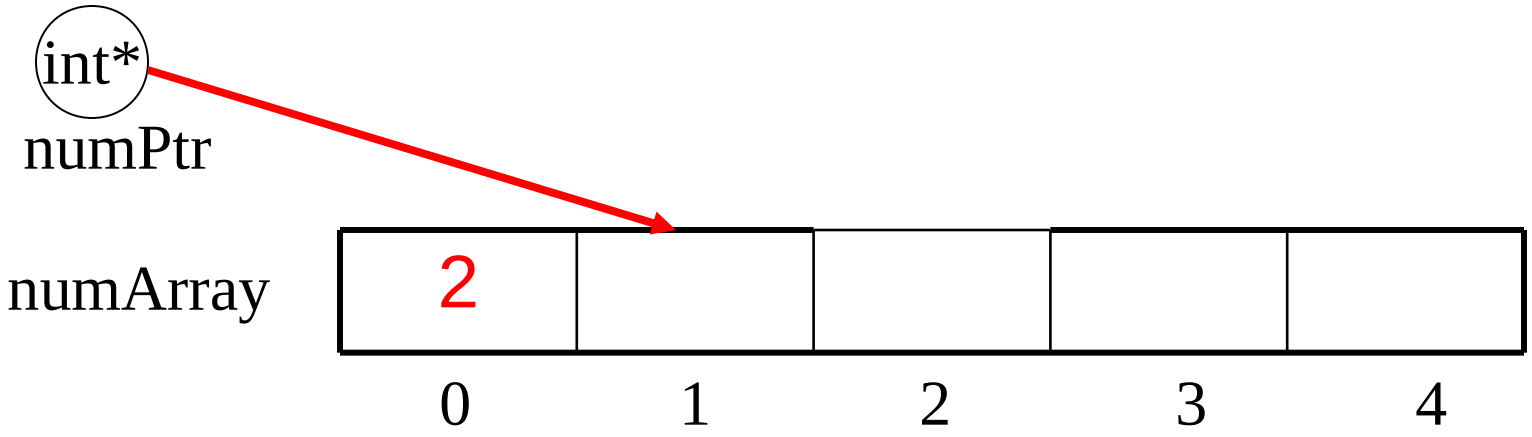


Pointer Arithmetic

What happens if we add to a pointer?

```
numPtr++;
```

Well, pointers have a type associated with them, so it'll 'do the right thing'!



Pointers and Arrays Are not The Same Thing

- For example:

```
int numArray[5];        //array of 5 ints
int * numPtr;           //pointer to an int
numPtr = numArray;      //OK!!!!
```

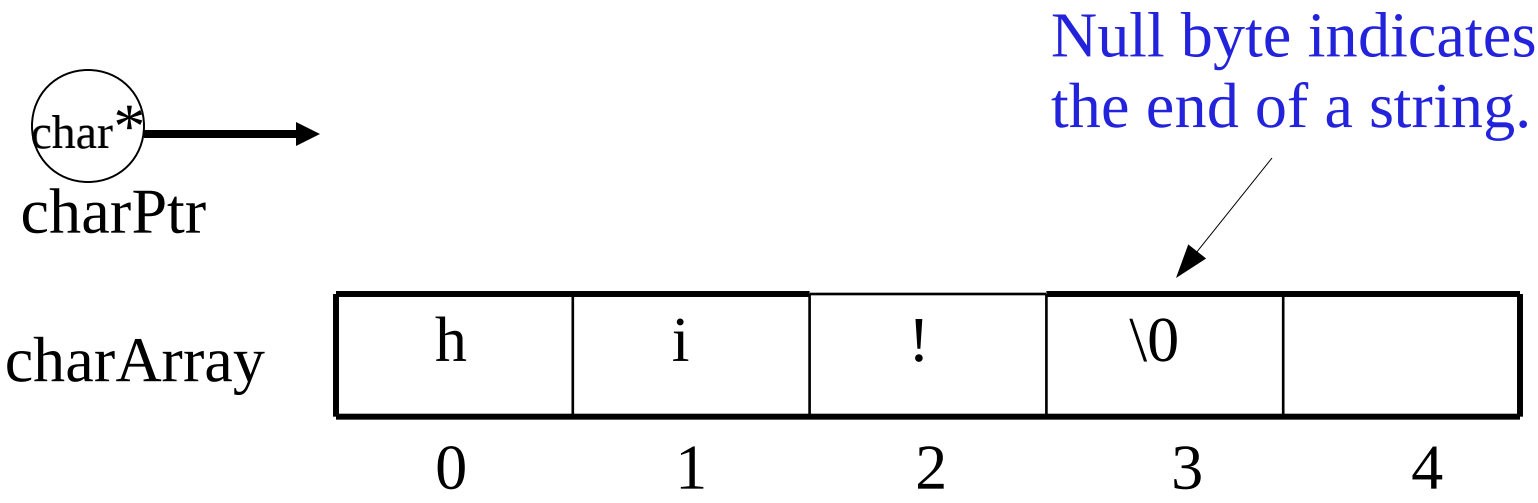
```
int numArray[5];        //array of 5 ints
int * numPtr;           //pointer to an int
numArray = numPtr;      //BAD!!!!
```

Strings

- There is no string type in C. Strings are just arrays of ASCII characters.

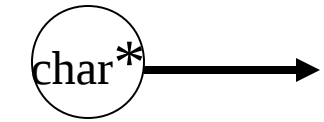
Strings in C

```
char charArray[5] = “hi!”;  //array of 5 chars
char * charPtr;             //pointer to a char
```



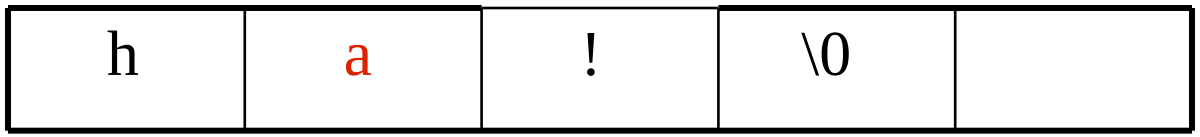
Strings in C

```
char charArray[5] = “hi!”;  //array of 5 chars
char * charPtr;             //pointer to a char

charArray[1] = 'a';     //assign a value
```



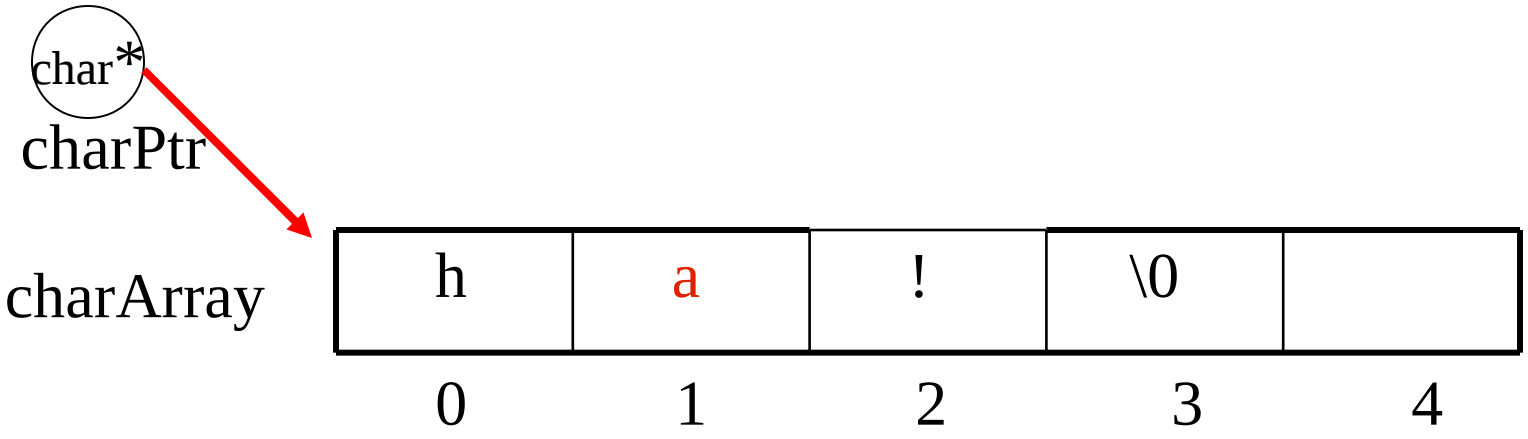
charPtr

charArray



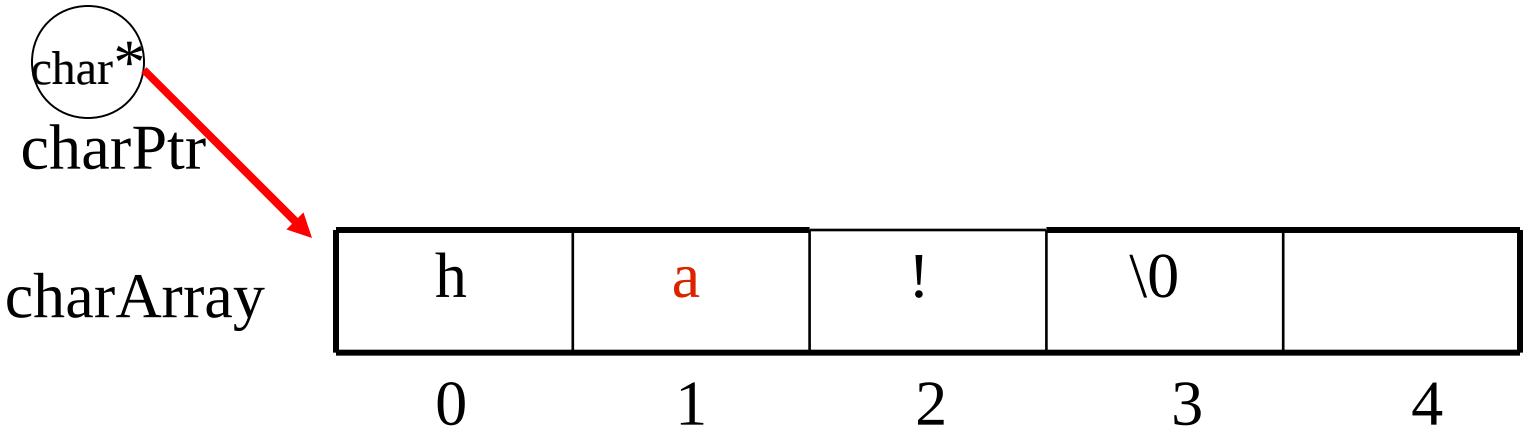
Strings in C

```
char charArray[5] = “hi!”;  //array of 5 chars
char * charPtr;             //pointer to a char

charArray[1] = 'a';      //assign a value

charPtr = charArray;     //assign pointer
```



Strings in C

```
char charArray[5] = “hi!”;  //array of 5 chars
char * charPtr;             //pointer to a char
charArray[1] = 'a';     //assign a value
charPtr = charArray;    //assign pointer

printf(“%s”, charPtr);  //print out ha!
```



Working with strings

- Nothing “Just Works”.

```
char charArray[5] = “hi!”;  //array of 5 chars
char otherArray[5] = “hi!”;

//charArray == otherArray ?? NO!

printf(“hello “ + “world”); //NO WAY!

CharArray = “ha!” //NOT A CHANCE!
```

Working with strings

- The solution: #include <string.h>

```
char charArray[5] = “hi!”;  //array of 5 chars
char otherArray[5] = “hi!”;

//charArray == otherArray ?? NO! strcmp

printf(“hello “ + “world”); //NO WAY! strcat

CharArray = “ha!” //NOT A CHANCE! strcpy
```

Some Very Bad Code

```
char * returnString()
{
   char * theString = “Hello World!”;
   return theString;
}
```

malloc()

- What if you don't know how large an array should be when you are writing your program?
- What if you want a function that returns a new array?
- malloc (memory allocate) sets aside memory dynamically:
- From stdlib.h

```
char* bigString =
     (char*)malloc(1024 * sizeof(char));
```

Freeing Memory

- Once memory is allocated it STAYS allocated.
- Unless you use the free function:

```
char* bigString =
     (char*)malloc(1024 * sizeof(char));
free(bigString);
```

- Not freeing memory results in “memory leaks”.