Pointers

Davood Rafiei

Addresses

```
drafiei@ug20:~/201>cat address.c
#include <stdio.h>
int ar[10];
int b;
short s;

int main()
{
  printf("&s=%p\n", &s);
  printf("&b=%p\n", &b);
  printf("ar=%p\n", ar);
  printf("&ar[0]=%p\n", &ar[0]);
  printf("&ar[3]=%p\n", &ar[3]);
}
drafiei@ug20:~/201>gcc -Wall -std=c99 address.c
drafiei@ug20:~/201>./a.out
&s=0x8049600
&b=0x8049648
ar=0x8049620
&ar[0]=0x8049620
&ar[3]=0x804962c
```

Call by Value vs. Call by Reference

```
drafiei@ug20:~/201>cat square-val-ref-printp.c
#include <stdio.h>

int squareByValue(int x) {
  printf("In squareByValue, &x: %p , x: %d\n", &x, x);
  return x*x;
}
void squareByRef(int *x) {
  *x = (*x)*(*x);
  printf("In squareByRef, x: %p , *x: %d\n", x, *x);
}
int main() {
  int v = 5;
  printf("In main befor call, &v: %p , v: %d\n", &v, v);
  v=squareByValue(v);
  squareByRef(&v);
  printf("In main after call, &v: %p , v: %d\n", &v, v);
}
```

```
drafiei@ug20:~/201>gcc -Wall -std=c99 square-val-ref-printptr.c
drafiei@ug20:~/201>./a.out
In main befor call, &v: 0xbff20970 , v: 5
In squareByValue, &x: 0xbff20950 , x: 5
In squareByRef, x: 0xbff20970 , *x: 625
In main after call, &v: 0xbff20970 , v: 625
```

Array Example

```
drafiei@ug20:~/201>cat array-printp.c
#include <stdio.h>

int main() {
  int ar[10] = {5, 6, 8, 7};
  int* ptr = &ar[1];
  printf("ar: %p\n&ptr: %p\nptr: %p\n", ar, &ptr, ptr);
  return 0;
}

drafiei@ug20:~/201>gcc -Wall -std=c99 array-printp.c
drafiei@ug20:~/201>./a.out
ar: 0xbfa91cc8
&ptr: 0xbfa91cc4
ptr: 0xbfa91ccc
```

Equivalent declarations

left is equivalent to right

`int* p1, p2;`	`int* p1;` `int  p2;`
`int *p1, *p2;`	`int* p1;` `int* p2;`
`int  *p`	`int*  p`

Size of Pointers

```
drafiei@ug20:~/201>cat pointer-size.c
#include <stdio.h>

int main()
{
  char    d1;
  int     d2;
  double  d3;
  char*   p1; // a pointer to a character
  int*    p2; // a pointer to an integer
  double* p3; // a pointer to a double

  printf("sizeof(d1): %d\n", sizeof(d1));
  printf("sizeof(d2): %d\n", sizeof(d2));
  printf("sizeof(d3): %d\n", sizeof(d3));
  printf("sizeof(p1): %d\n", sizeof(p1));
  printf("sizeof(p2): %d\n", sizeof(p2));
  printf("sizeof(p3): %d\n", sizeof(p3));
}
drafiei@ug20:~/201>gcc -Wall -std=c99 pointer-size.c
drafiei@ug20:~/201>./a.out
sizeof(d1): 1
sizeof(d2): 4
sizeof(d3): 8
sizeof(p1): 4
sizeof(p2): 4
sizeof(p3): 4
```

Pointer Dereferencing

```
drafiei@ug20:~/201>cat deref.c
#include <stdio.h>

Int main()
{
  int ar[10] = {1, 12, 20, 33, 42};
  int* p;

  p = &ar[3];  //two references to the same location
  printf("ar[3]: %d\n", ar[3]);
  printf("*p: %d\n", *p);
  *p = 90;
  printf("ar[3]: %d\n", ar[3]);
  ar[3] = 120;
  printf("*p: %d\n", *p);
}
drafiei@ug20:~/201>gcc -Wall -std=c99 deref.c
drafiei@ug20:~/201>./a.out
ar[3]: 33
*p: 33
ar[3]: 90
*p: 120
```

Pointer Assignments

int *p1, *p2;

p2 = p1;

- Pointer assignment
- “Make p2 point to where p1 points”

*p1 = *p2;

- Value assignment
- Assigns ‘value pointed to’ by p1, to ‘value pointed to’ by p2

- Pointers should always be initialized before dereferencing
  - Otherwise, they can be dangerous!!

Pointer Arithmetic

- If **`p`** is a pointer and **`p==3000`**, what is **`p+1`**?
- It depends on the type of **`p`**
  - **`char*`** **`3001`**
  - **`int*`** **`3004`**
  - **`double*`** **`3008`**
  - **`void*`** **`3001`**

lower2upper.c

The following function converts the letters in a string to upper case.
Fill in the blanks.

```
#include <stdio.h>

void lower2upper(char str[])
{
  for (char* p = str; ???; ???) {
    if (*p>='a' && *p<='z')
      *p = *p + 'A' - 'a';
  }
}
int main()
{
  char str[] = "Hello world!";
  lower2upper(str);
  printf("New str: %s\n", str);
}
```

pointer-math.c

```
 drafiei@ug20:~/201>cat pointer-math.c
#include <stdio.h>

int main()
{
 double  a[] = { 3.2, 7.9, 11.3};
 double* aPtr = &a[0]; // or aPtr = a;

 printf(" aPtr: %p\n", aPtr);
 printf("*aPtr: %f\n", *aPtr);
 aPtr += 2;
 printf(" aPtr: %p\n", aPtr);
 printf("*aPtr: %f\n", *aPtr);
 printf("&a[2] - &a[0]: %d\n", &a[2] - &a[0]);
}
drafiei@ug20:~/201>gcc -Wall -std=c99 pointer-math.c
drafiei@ug20:~/201>./a.out
 aPtr: 0xbfdc4fe8
*aPtr: 3.200000
 aPtr: 0xbfdc4ff8
*aPtr: 11.300000
&a[2] - &a[0]: 2
```

Passing Pointer as Argument

```
drafiei@ug20:~/201>cat pointer-arg.c
#include <stdio.h>

void increment(int* p)
{
  *p = *p + 1; // (*p)++
  p = 0;
}
int main()
{
  int i = 9;
  int* ip = &i;
  increment(ip);
  printf("i: %d , &i: %p , ip: %p\n", i, &i, ip);
}
```

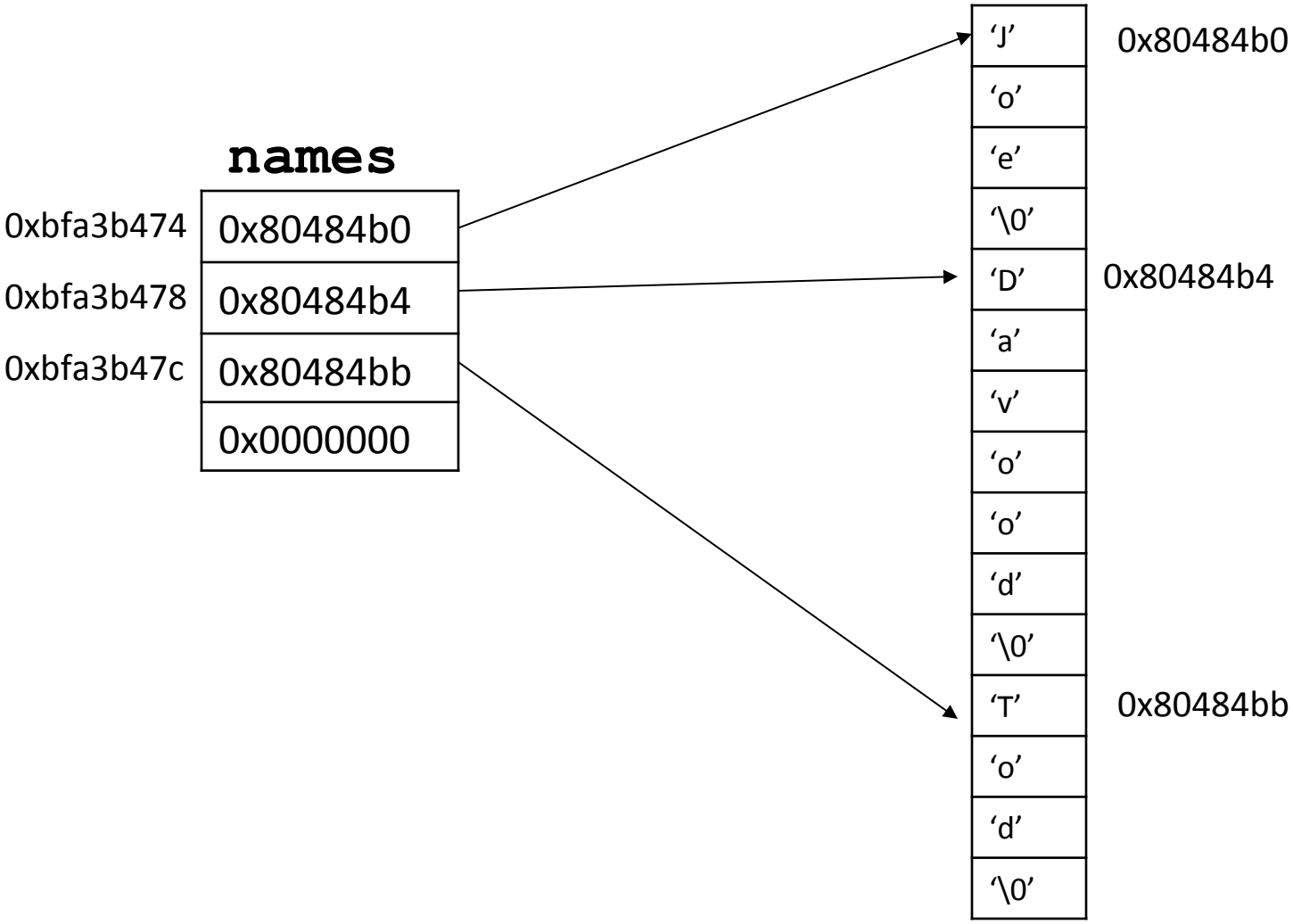
```
drafiei@ug20:~/201>gcc -Wall -std=c99 pointer-arg.c
drafiei@ug20:~/201>./a.out
i: 10 , &i: 0xbffc7220 , ip: 0xbffc7220
```

Array of Pointers

```
drafiei@ug20:~/201>cat array-of-pointers.c
#include <stdio.h>

int main()
{
  char* names[] = {"Joe", "Davood", "Tod", 0};
  printf("names: %p\n", names);
  for (int i = 0; names[i]!=0; i++)
    printf("&names[%d]: %p , names[%d]: %p , names[%d]: %s\n", i, &names[i],
           i, names[i], i, names[i]);
}

drafiei@ug20:~/201>gcc -Wall -std=c99 array-of-pointers.c
drafiei@ug20:~/201>./a.out
names: 0xbfa3b474
&names[0]: 0xbfa3b474 , names[0]: 0x80484b0 , names[0]: Joe
&names[1]: 0xbfa3b478 , names[1]: 0x80484b4 , names[1]: Davood
&names[2]: 0xbfa3b47c , names[2]: 0x80484bb , names[2]: Tod
```


names
0xbfa3b474
0xbfa3b478
0xbfa3b47c
0x80484b0
0x80484b4
0x80484bb
0x0000000
'J'
'o'
'e'
'\0'
'D'
'a'
'v'
'o'
'o'
'd'
'\0'
'T'
'o'
'd'
'\0'
0x80484b0
0x80484b4
0x80484bb

Pointers to Pointers

- A pointer variable occupies a memory location (how many bytes?) and has an address.
  - `T** p` is a pointer to pointer to `T`
  - `*p` is a pointer to `T`
  - `**p` is a `T`

pointer2pointer.c

```
#include <stdio.h>
int main() {
  int ar[] = {1, 2, 3, 4};
  int* ptr = &ar[3];
  int** ptrPtr = &ptr;
  (*ptrPtr)--;
  **ptrPtr = 9;
  for (int i = 0; i<4; i++)
    printf("a[%d]: %d\n", i, ar[i]);
  return 0;
}
```

Type Cast

- C provides casts to convert one data type to another.
- Syntax:
  - (Type) expression
- Examples

```
int   ival = 12;
double dval = (double) ival;

char*  ptr;
int  ar[5] = {1, 2, 3, 4, 5};
ptr = (char*) &ar[0];
```