Pointer Lesson 2 Outline

Pass by Reference Bad Example

```
% cat henrys_house_bad.c
#include <stdio.h>
int main ()
{ /* main */
    int henrys_house;
    void who(int  dr_neemans_house);

    who( henrys_house);
    printf("%d people live in Henry's house.\n",
        henrys_house);
} /* main */

void who (int  dr_neemans_house)
{ /* who */
    printf("How many people live in Dr Neeman's house?\n");
    scanf("%d", &dr_neemans_house);
} /* who */
% gcc -o henrys_house_bad henrys_house_bad.c
% henrys_house_bad
How many people live in Dr Neeman's house?
4
134513624 people live in Henry's house.
```

Pass by Reference Good Example

```
% cat henrys_house_good.c
#include <stdio.h>
int main ()
{ /* main */
    int henrys_house;
    void who(int* dr_neemans_house);

    who (&henrys_house);
    printf("%d people live in Henry's house.\n",
        henrys_house);
} /* main */

void who (int* dr_neemans_house)
{ /* who */
    printf("How many people live in Dr Neeman's house?\n");
    scanf("%d", dr_neemans_house);
} /* who */
% gcc -o henrys_house_good henrys_house_good.c
% henrys_house_good
How many people live in Dr Neeman's house?
4
4 people live in Henry's house.
```

Is Pass by Reference Really by Reference?

In C, the **only** passing strategy is pass by copy.

To pass by reference, we have to piggyback on top of pass by copy – because in C, **everything** is pass by copy.

So, the **value** that we have to pass by copy is the **address** of the argument whose value we want to change, which we achieve using the **address operator** `&`.

In other words, in C pass by reference is actually pass by copy: you copy the address.

More on the Address Operator &

```
% cat addr.c
#include <stdio.h>
int main ()
{ /* main */
    double dub = 5.0;
    float  flo = 4.0;
    int    in  = 3;

    printf("dub = %f, &dub = %d\n", dub, &dub);
    printf("flo = %f, &flo = %d\n", flo, &flo);
    printf("in  = %d, &in  = %d\n", in,  &in);
} /* main */
% gcc -o addr addr.c
% addr
dub = 5.000000, &dub = 536869704
flo = 4.000000, &flo = 536869696
in  = 3, &in  = 536869688
```

Pass by Reference via Pass by Copy?

How does this help us in converting from pass by copy to pass by reference?

Well, the value of the expression `&dub` is the address of `dub`.

If we pass a copy of the value of `&dub`, then we're passing the address of `dub`, so we're passing `dub` by reference.

Huh?

How Pass by Reference Works in C

Okay, so we've decided that, if we pass the value of `&dub`, then we're passing `dub` by reference, because we're passing the address of `dub`.

What's that all about?

Well, **pass by reference** means that the formal argument **refers** to the actual argument, in the sense that the formal argument has the same memory address as the actual argument.

But **pass by value** means that the value of the actual argument is **copied** into a new memory location, which is the memory location of the formal argument.

Pass by Reference in C

So let's say we're doing pass by value.
If the value that we pass is the **address** of the actual argument, then the formal argument **knows** the memory location of the actual argument.

In which case, if we can figure out how to ***dereference*** the address contained in the formal argument – to use it to get to the contents of that address – then we'd have the address of the actual argument.

Which would be pass by reference.

So, what we need is a way to dereference an address.

Happily, C provides a ***dereference operator***:

*

We use the dereference operator with pretty much the same syntax that we use for the address operator:

```
*dub
```

Pass by Reference Bad Example

```
% cat henrys_house_bad.c
#include <stdio.h>
int main ()
{ /* main */
    int henrys_house;
    void who(int  dr_neemans_house);

    who( henrys_house);
    printf("%d people live in Henry's house.\n",
        henrys_house);
} /* main */

void who (int  dr_neemans_house)
{ /* who */
    printf("How many people live in Dr Neeman's house?\n");
    scanf("%d", &dr_neemans_house);
} /* who */
% gcc -o henrys_house_bad henrys_house_bad.c
% henrys_house_bad
How many people live in Dr Neeman's house?
4
134513624 people live in Henry's house.
```

Pass by Reference Good Example

```
% cat henrys_house_good.c
#include <stdio.h>
int main ()
{ /* main */
    int henrys_house;
    void who(int* dr_neemans_house);

    who (&henrys_house);
    printf("%d people live in Henry's house.\n",
        henrys_house);
} /* main */

void who (int* dr_neemans_house)
{ /* who */
    printf("How many people live in Dr Neeman's house?\n");
    scanf("%d", dr_neemans_house);
} /* who */
% gcc -o henrys_house_good henrys_house_good.c
% henrys_house_good
How many people live in Dr Neeman's house?
4
4 people live in Henry's house.
```

More on Pointers

So, a ***pointer*** is a variable whose value is a reference (that is, an address of a location in memory).
It **points** to the location in memory.

Notice that, to assign a value to a pointer, we apply the dereference operator `*` to the pointer:

```
*dr_neemans_house = 4;
```

Likewise, to use the value of the variable pointed to by a pointer, we also apply the dereference operator `*` to the pointer:

```
printf("%d people\n", *dr_neemans_house);
```

Pointer Variables

```
% cat pointer.c
#include <stdio.h>
int main ()
{ /* main */
    int q; int *p;

    q = 5; p = &q;
    printf("q = %d, &q = %d\n", q, &q);
    printf("p = %d, *p = %d\n", p, *p);
} /* main */
% gcc -o pointer pointer.c
% pointer
q = 5, &q = 536869704
p = 536869704, *p = 5
```

An Array Variable Is a Pointer

In C, when we declare an array statically

```
float static_array[100];
```

we are setting up a block in memory,
but we're doing it at compile time instead of at runtime.

Otherwise, an array is identical to a pointer. Specifically,
it's a **pointer** to the block of memory that holds the array.

In fact, you can think of a statically allocated array as
a **pointer constant**:
its value (the address that it points to)
is set at compile time and cannot change at runtime.