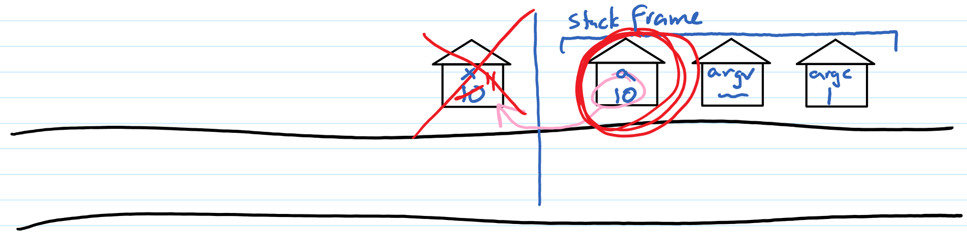
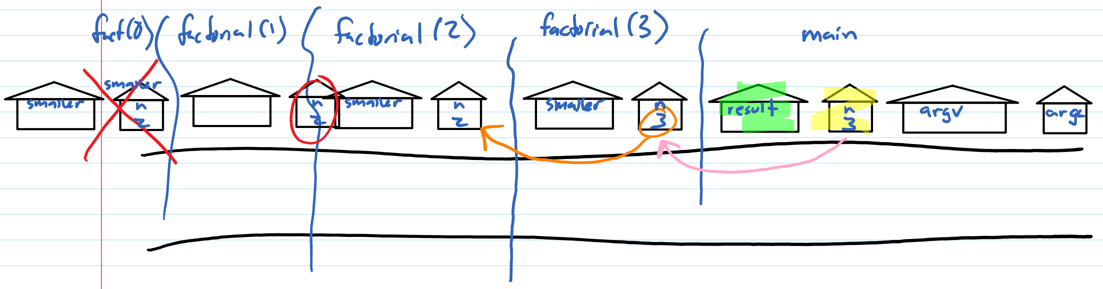
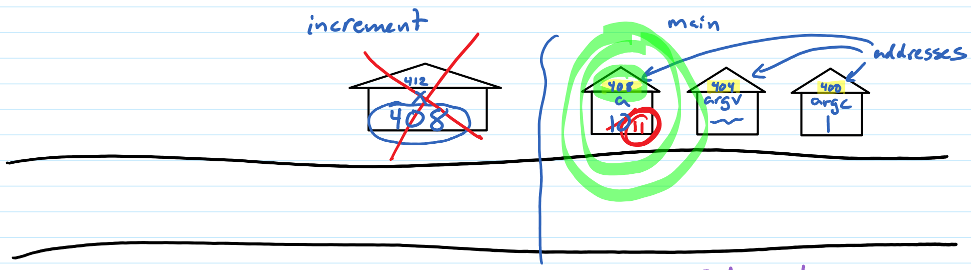


system call stack
(local variable,
parameters)

```
int main(int argc, char *argv[]) {        int increment(int x) {
  int a = 10;                               x = x + 1;
  increment(a);                           }
  printf("a=%d\n", a);
}
```



```
int main(int argc, char *argv[]) {     long factorial(int n) {
  int n = atoi(argv[1]);                 if (n == 0)
  long result = factorial(n);                return 1;
  printf("%d! = %ld\n", n, result);      else {
}                                            long smaller = factorial(n - 1);
                                             return n * smaller;
                                           }
                                       }
```



int pointer

```
int main(int argc, char *argv[]) {      void increment(int *x) {
  int a = 10;                             *x = *x + 1;
  increment(&a);                        }
  printf("a=%d\n", a);
}
```

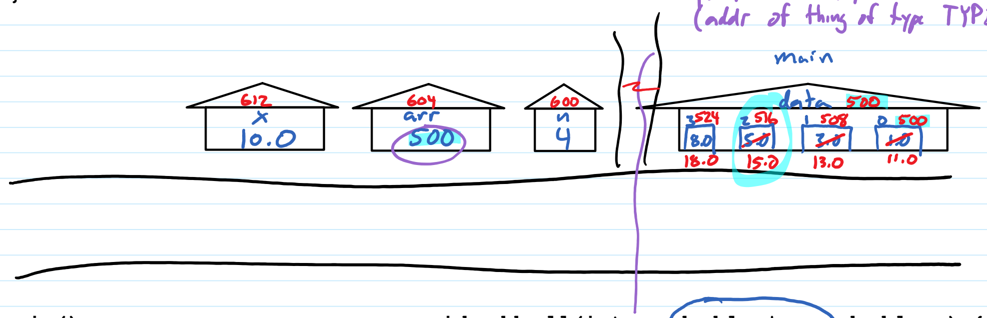
address-of

dereference

x is an int pointer
*x is an int

pointer to TYPE
(addr of thing of type TYPE)

TYPE * name



```
int main()                                 void add_all(int n, double *arr, double x) {
{                                            for (int i = 0; i < n; i++) {
  double data[] = {1.0, 3.0, 5.0, 8.0};        arr[i] += x;
  // …                                       }
  add_all(4, data, 10.0);                  }
  data[2] = 56.961247;
```

1) start w/ address of arr

i=0 500+0*8
i=1 500+1*8
i=2 500+2*8

```
double data[] = {1.0, 3.0, 5.0, 8.0};      arr[i] += x;
//...                                     }
add_all(4, data, 10.0);                 }
data[2] = 56.961247;
//...
}
```

arrays degrade to pointers

1) data → addr 500

2) 500 + 2 * 8 = 516

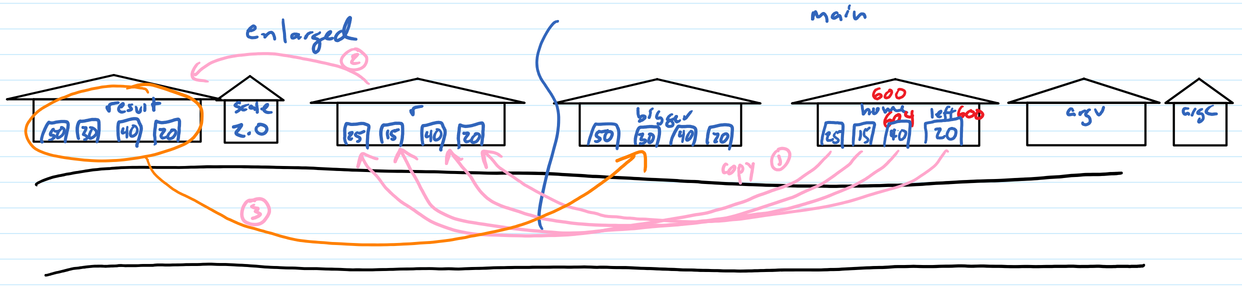
1) start w/ address of arr

2) add i * sizeof(double)

3) use the thing at that addr

i=0	500+0*8
i=1	500+1*8
i=2	500+2*8
i=3	500+3*8

```
typedef struct _rectangle
{
  int left;
  int top;
  int width;
  int height;
} rectangle;

rectangle enlarged(rectangle r, double scale)
{
  rectangle result = {r.left, r.top, r.width * scale, r.height * scale};
  return result;
}

int main(int argc, char *argv[])
{
  rectangle home = {20, 40, 15, 25};
  rectangle bigger = enlarged(home, 2.0);

  printf("%d %d %d %d\n", home.left, home.top, home.width, home.height);
  printf("%d %d %d %d\n", bigger.left, bigger.top, bigger.width, bigger.height);

}
```



&home	600
&(home.left)	600
&(home.top)	604
	⋮

$\&home + 4 \overset{?}{=} \&(home.left)$

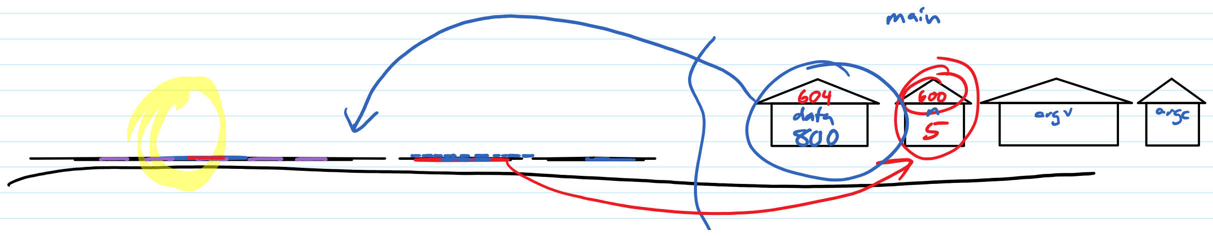
not necessarily true

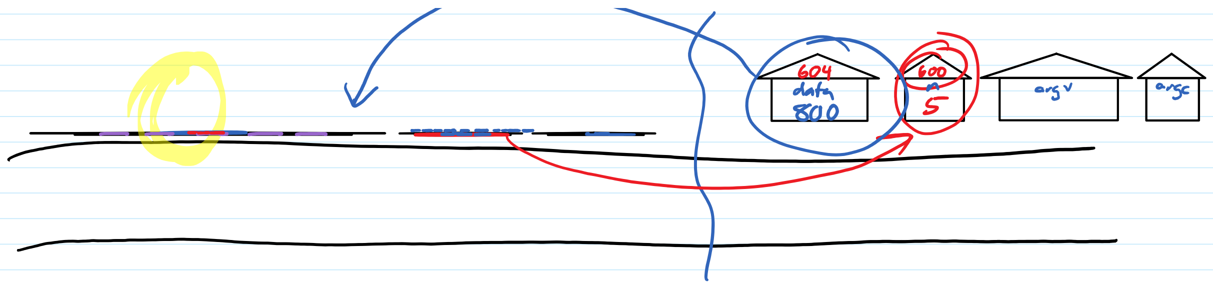
```
int main() {                                double *read_array(FILE *in, int *n) {
  // ...                                      fscanf(in, "%d", n);
  int n;
  double *data = read_array(stdin, &n);       if (n > 0) {
                                                double input[*n];
                                                for (int c = 0; c < *n; c++) {
                                                  scanf("%lf", &input[c]);
                                                }

                                                return input;
                                              }
                                            }
```

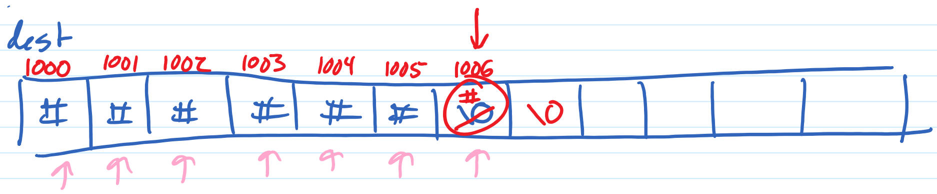
printf("%lf", data[2])

data

800 + 2 * 8 = 816







```
strcat(dest + i * src_len, src);
```

$1000 + (i * src\_len) * sizeof(char)$

1006

$i=6$

$src\_len=1$

$\&(arr[i]) \equiv arr + i$