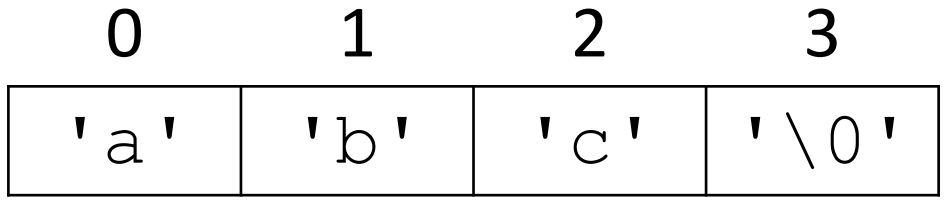
CSC111

Admin

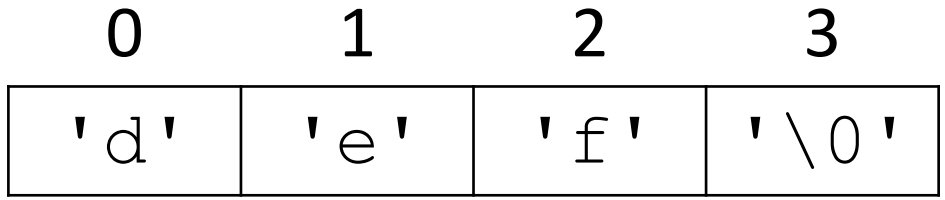
- Reference/data sheets can be picked up in office hours (until next Friday)
- Exams handed back in labs
  - Please check your addition of marks
  - Mark appeals accepted until Friday April 1st, 2022
    - Please come to office hours
- Assignment 6 due this weekend
- Assignment 7 will be available next week

Strings = null terminated char array…

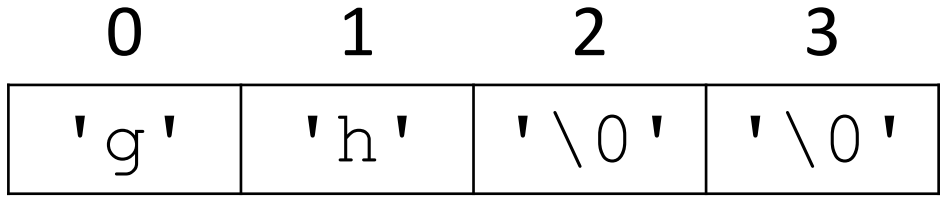
```
char my_string_1[] = {'a', 'b', 'c', '\0'};
```




```
char my_string_2[] = "def";
```




```
char my_string_2[4] = "gh";
```



```
void print_array ( char array[], int len ) {
    int i;
    for (i=0; i<len; i++) {
       printf(" %c  ", array[i]);
    }
}
```

```
void print_string (char str[]) {
    int i=0;
    while (str[i] != '\0') {
       printf("%c ", str[i++]);
    }
}
```

```
void print_string (char str[]) {

    while (*str != '\0') {
       printf("%c ", *str++);
    }
}
```

`string.h`

- The `string` library contains functions that operate on `string` types (null terminated `char` arrays)
- To use these functions in your program include the header file:
  `#include <string.h>`
- `strlen, strcpy, strcmp`

Defining and using Compound Data

structs – defining new data types

```
struct student
    char name[20];
    char vnum[8];
    int gpa;
};
```

Given the following new type is defined:

```
struct student {
    char name [50];
    char vnum [10];
    int gpa;
};
```

Declaring a variable of a struct student type:

```
struct student my_student_0;
struct student my_student_1;
struct student my_student_2;
```

type

identifier/
variable name

Declaring
primitive type variables:

```
int my_int;
double my_double;
```

type

identifier/
variable name

Allocates space in memory
with initial **garbage** values

Declaring AND initializing
primitive type variables:

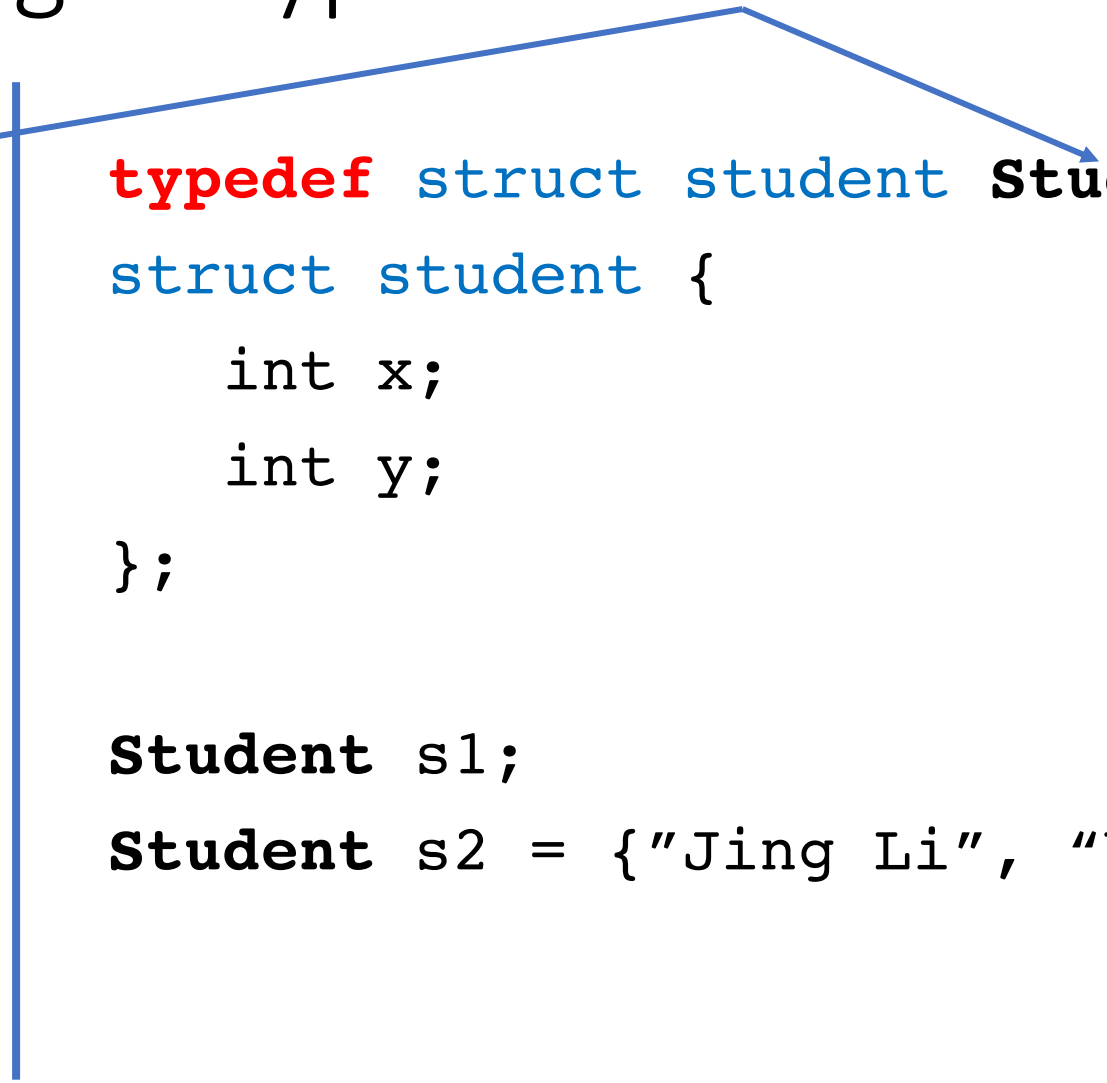
```
int my_int = 5;
double my_double = 5.6;
```

Declaring AND initializing variable of a struct type:

```
struct student student_3 = {"Rajyn Singh", "V00123456", 9};
struct student student_4 = {"Sally Reader", "V00723456", 8};
struct student student_5 = {"Jing Li", "V00523456", 8};
```

Allocates space in memory
with initial **given** values

`typedef` to give types an alias



```
typedef int Integer;

int x;
int y = 10;

Integer a;
Integer z = 15;
```

```
typedef struct student Student;
struct student {
    int x;
    int y;
};

Student s1;
Student s2 = {"Jing Li", "V00523456", 8};
```

typedef (cont)

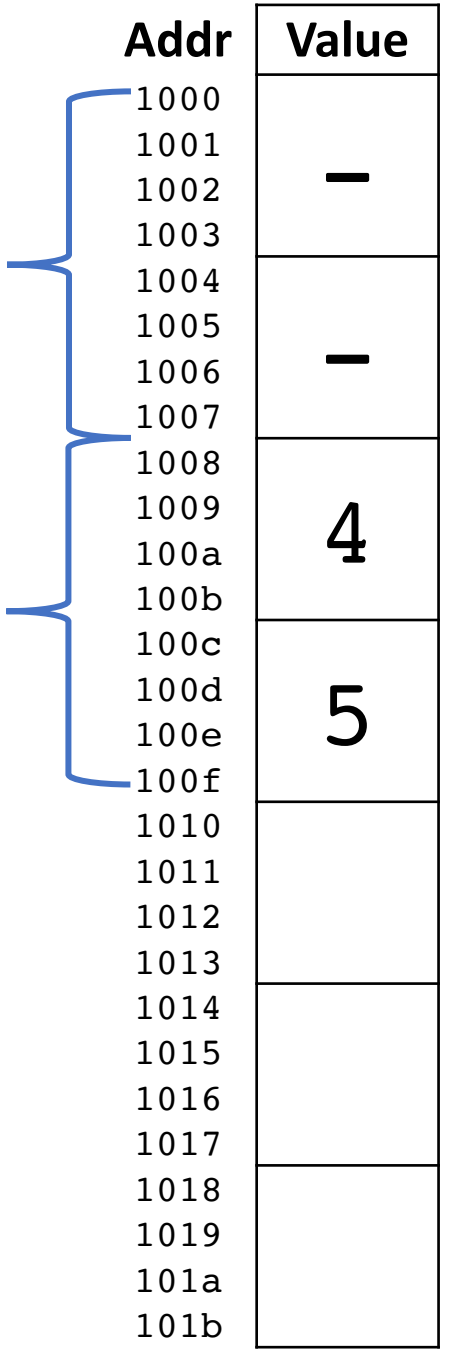
```
typedef struct {
    int x;
    int y;
} student;
```

```
typedef struct student Student;
struct student {
    int x;
    int y;
};
```

structs in memory…

```
typedef struct point Point;
struct point {
    int x;
    int y;
};

Point p1;
Point p2 = {4, 5};
```

- A contiguous block of memory is allocated
- space for values of specified type and size of the fields in Point



accessing values…

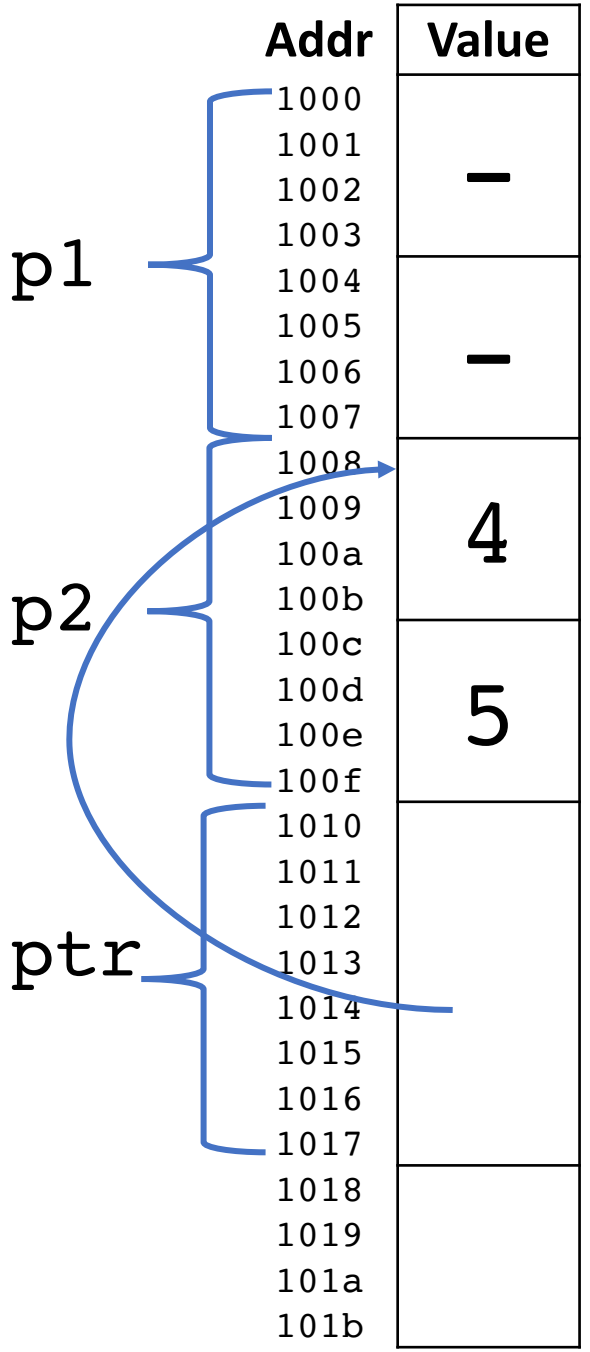
```
Point p1;
Point p2 = {4, 5};

printf("%d, %d\n",      p2.x,      p2.y);

Point* ptr = &p2;

printf("%d, %d\n", (*ptr).x, (*ptr).y);
printf("%d, %d\n",  ptr->x,   ptr->y);
```

OUTPUT:

4, 5

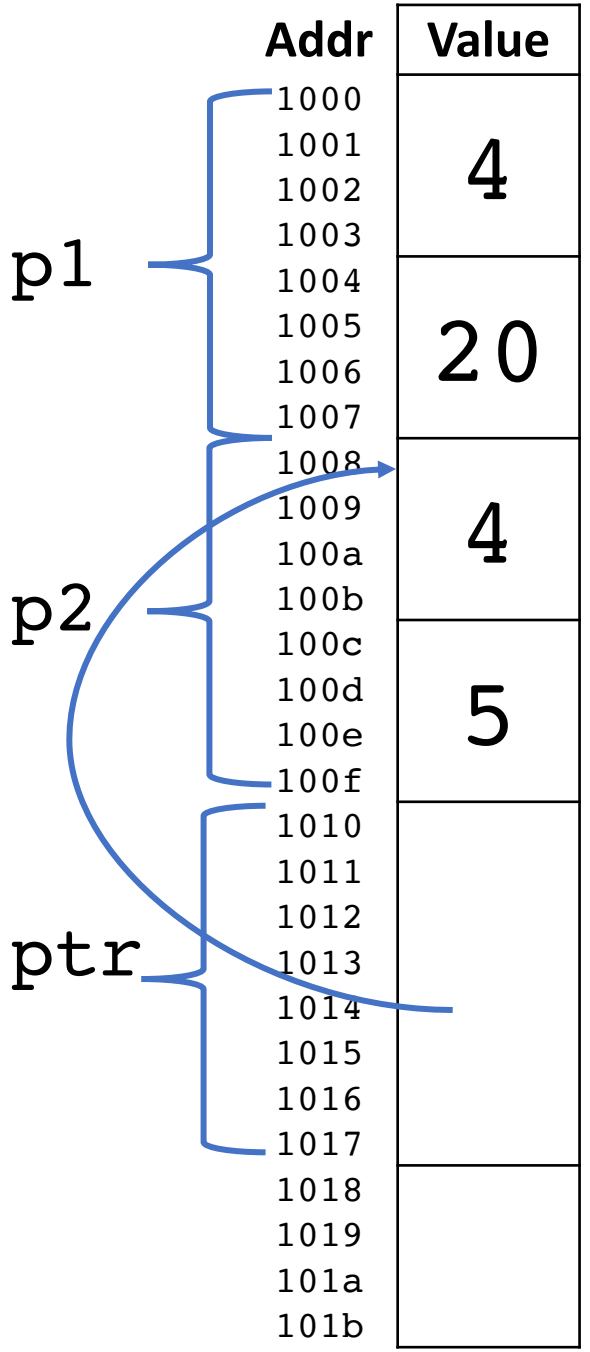
4, 5

4, 5



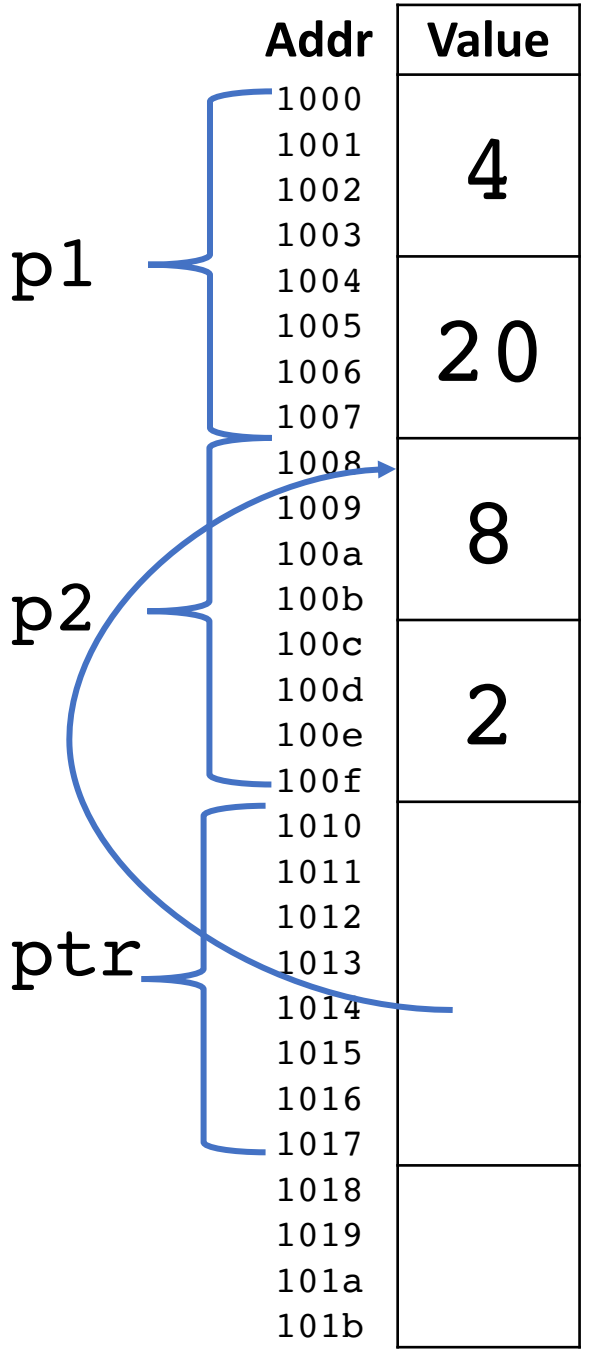
assigning values…

```
Point p1;
Point p2 = {4, 5};
Point* ptr = &p2;
p1 = p2;
p1.y = 20;
```



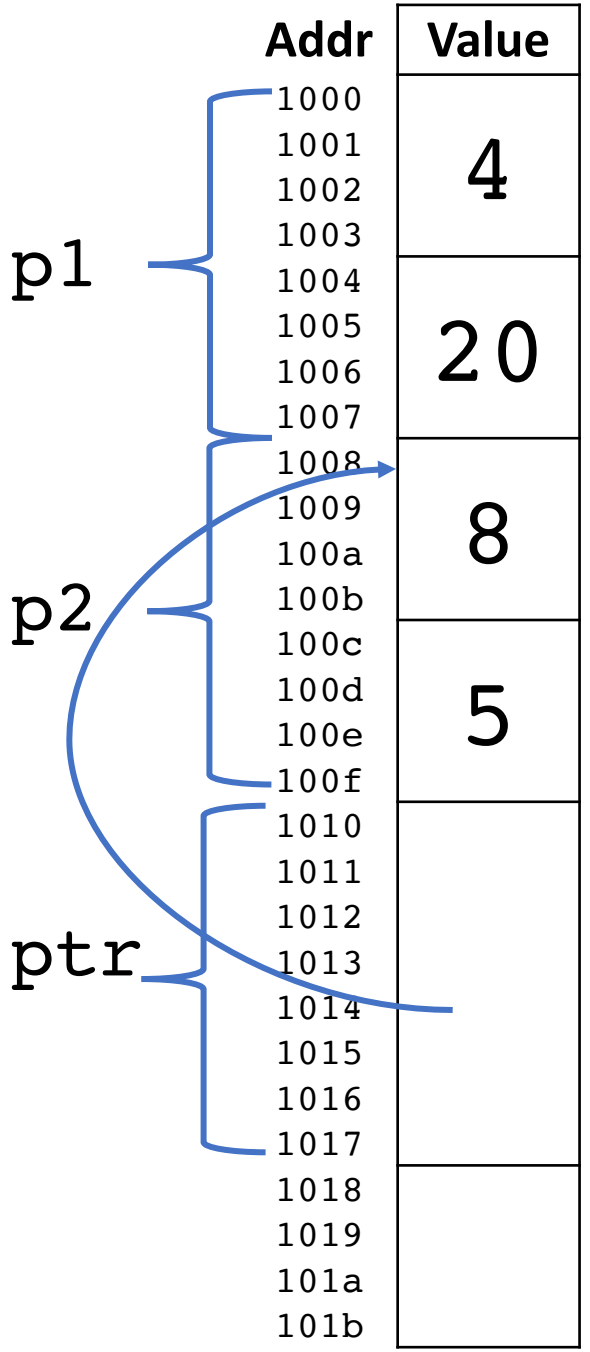
assigning values…

```
Point p1;
Point p2 = {4, 5};
Point* ptr = &p2;
p1 = p2;
p1.y = 20;
p2.x = 8;
ptr->y = 2;
```



assigning values…

```
Point p1;
Point p2 = {4, 5};
Point* ptr = &p2;
p1 = p2;
p1.y = 20;
p2.x = 8;
```



Demo

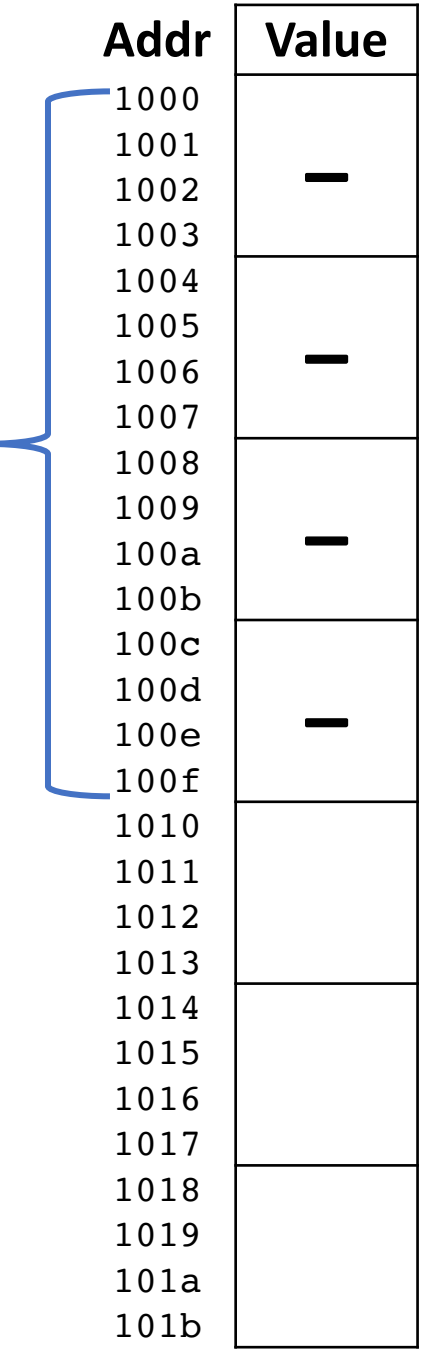
```
typedef struct point Point;
struct point {
    int x;
    int y;
};
Point point_array[2];
```

type of elements

name of the array

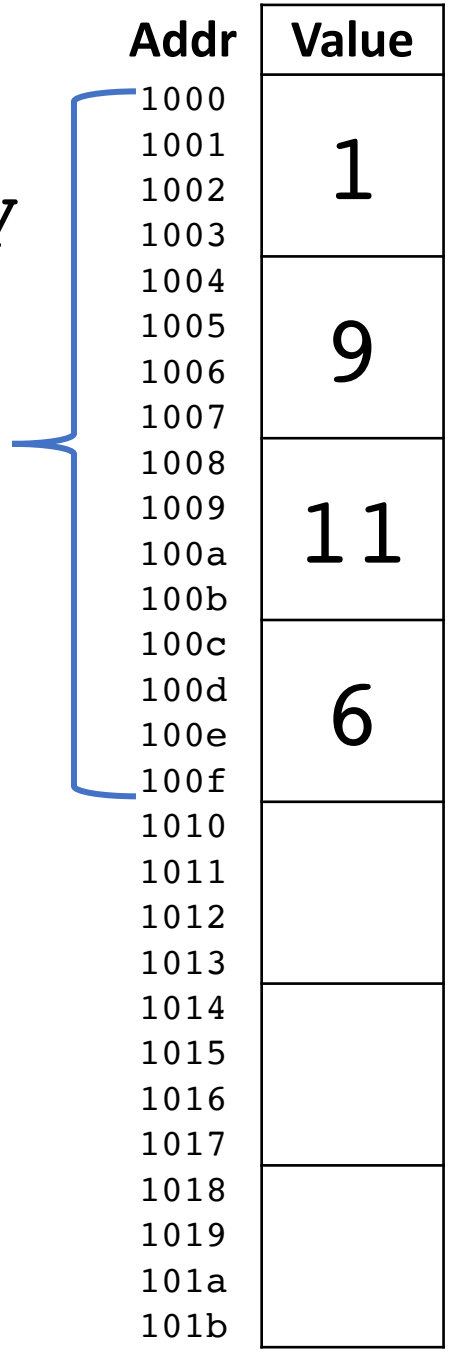
size of the array

point_array

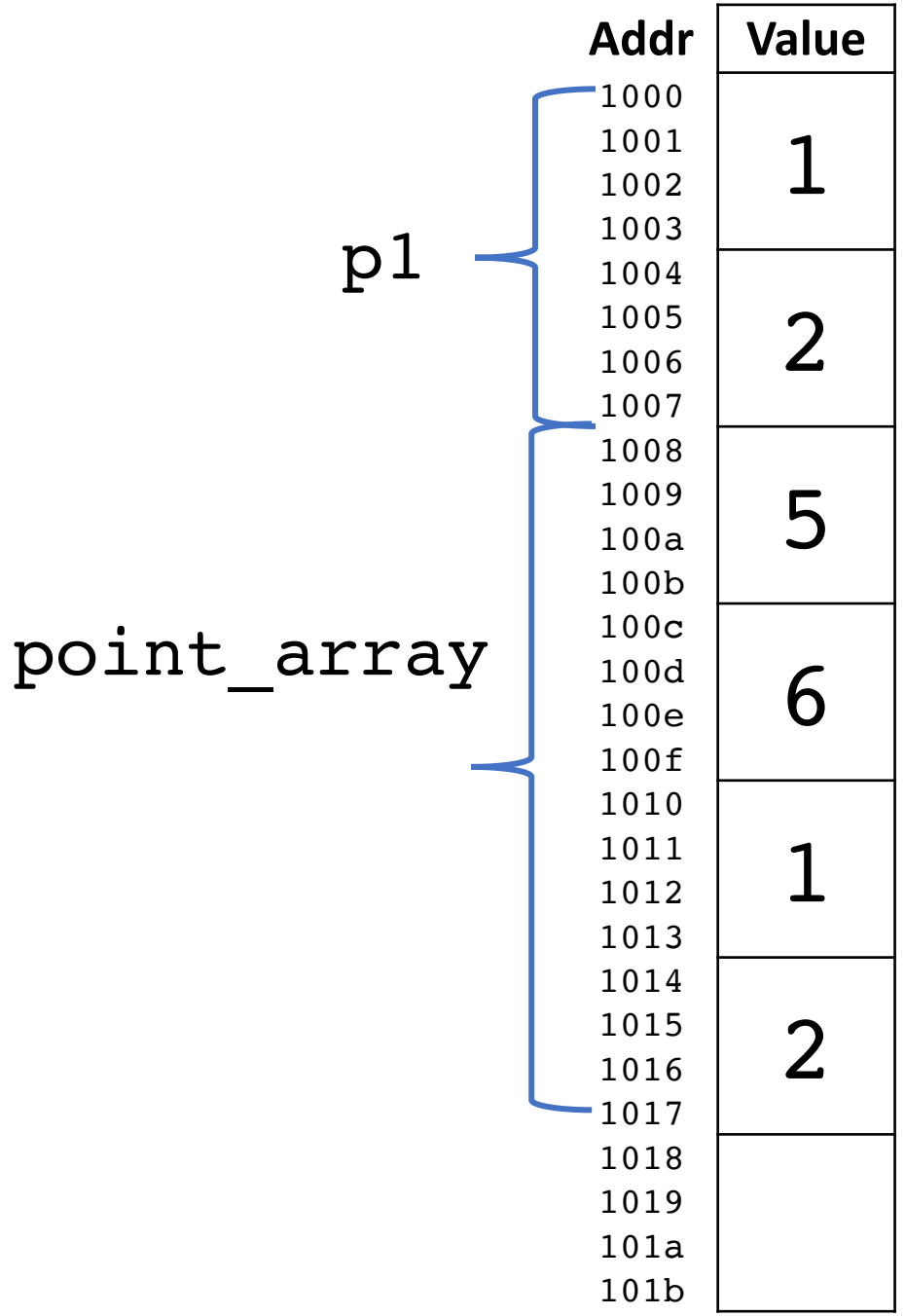


```
Point point_array[2] = {{1,2}, {5,6}};
point_array[0].y = 9;
point_array[1].x = 11;
```

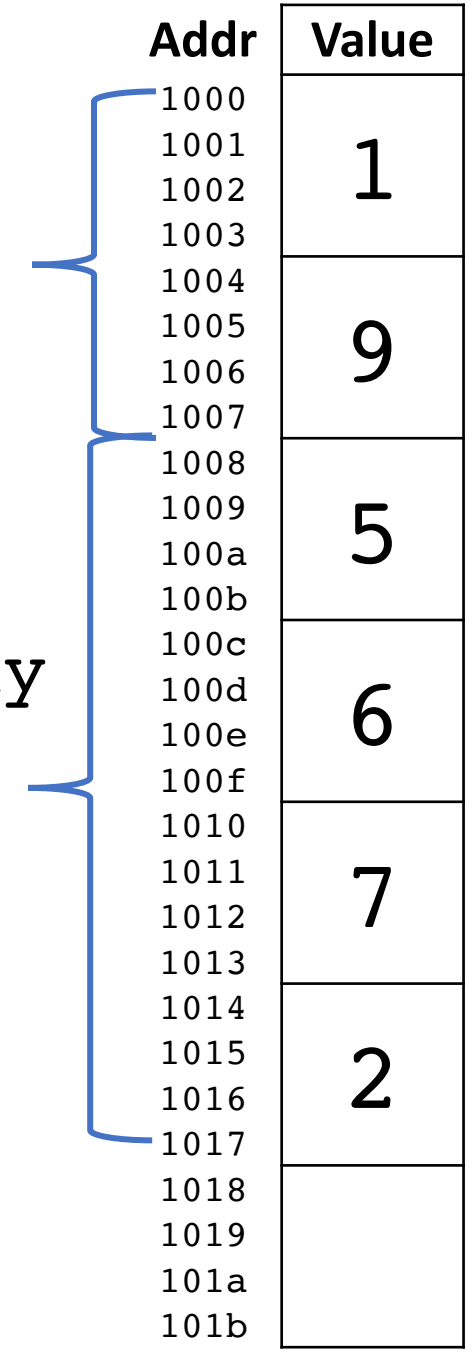
point_array



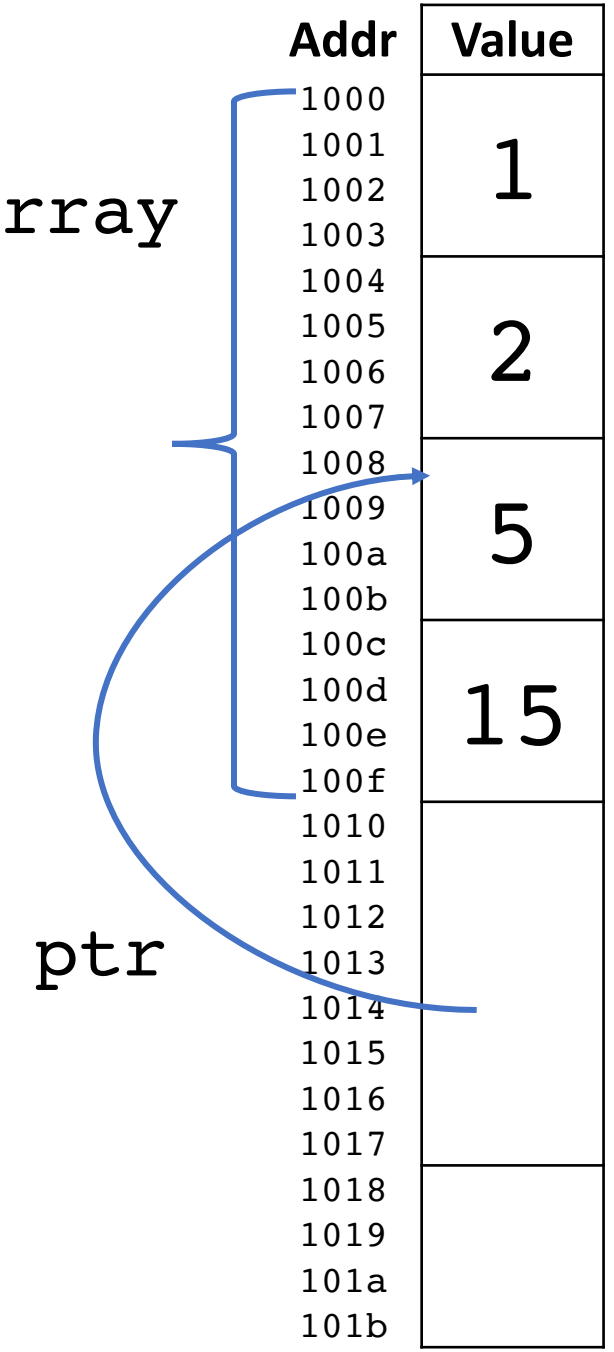
```
Point p1 = {1,2};
Point point_array[2] = {{5,6}, p1};
```



```
Point p1 = {1,2};
Point point_array[2] = {{5,6}, p1};

point_array[1].x = 7;
p1.y = 9;
```



```
Point point_array[2] = {{1,2}, {5,6}};

Point* ptr = &point[1];
ptr->y = 15;
```



structs and functions

- Can pass and return structs to/from functions.
- Can pass a pointer to a struct to a function
  - Remember errors will occur if you return a pointer to a struct created in a function scope

- Demo – write a function that takes an array of students and returns the student that has the highest grade

Define a struct that stores information about cars and initialize it with some data

- Car make - max length 50
- Type (string: SUV, coupe, truck) - max length 20
- Model Year
- Max speed

Update your vehicle to increase the max speed by 10 km/h

Define a function that returns a car struct from information passed to it

- The function should accept
  - Make
  - Type
  - Year
  - Max speed
- The function should return a new car struct