structs are a collection of fields

In a previous PA...

```
int student1ID;
int student1Age;
double student1GPA;
char student1Name[64];

int student2ID;
int student2Age;
double student2GPA;
char student2Name[64];
```

Now, with structs...

```
typedef struct student {
    int id;
    int age;
    double gpa;
    char name[64];
} Student;

Student students[2];
```

Author: Alan Chu (ualch9@gmail.com)
https://github.com/ualch9/cpts121-lab-presentations

structs are for programmers

```c
typedef struct student {
    int id;
    int age;
    double gpa;
    char name[64];
} Student;
```

```
{ int, int, double, char[64] }
```

As a programmer, `int` by itself means nothing.
Having fields makes it easier to deal with data.

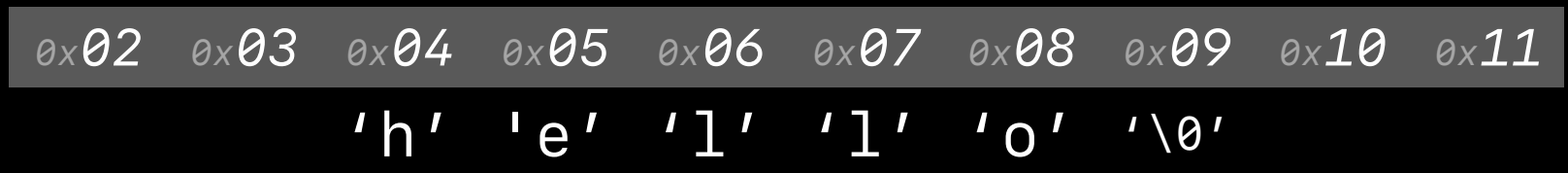
Author: Alan Chu (ualch9@gmail.com)

structs in memory

Like **strings**, which is a contiguous set of **char**,

structs are a contiguous set of fields

```
char* message = "hello";
```



When you use **message** in your .c code,
message is a pointer pointing at 0x04

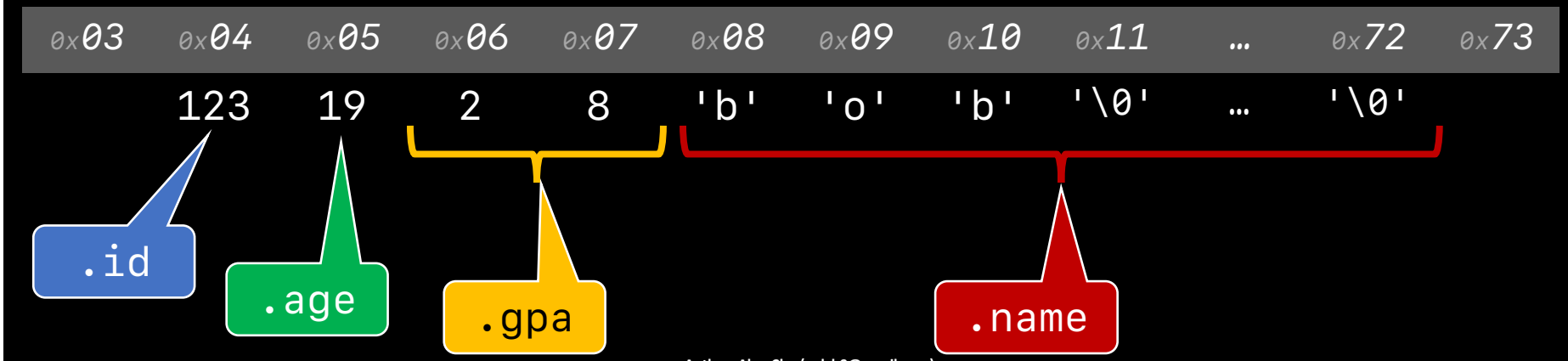
Author: Alan Chu (ualch9@gmail.com)

structs in memory

Like `strings`, which is a contiguous set of `char`,

`structs` are a contiguous set of fields

```
Student bob;
bob.id = 123;
bob.age = 19;
bob.gpa = 2.8;
strcpy(bob.name, "bob");
```



memory representation is a massive simplification.

Author: Alan Chu (ualch9@gmail.com)
https://github.com/ualch9/cpts121-lab-presentations

naming is important

```c
typedef struct student {
    int id;
    int age;
    double gpa;
    char name[64];
} Student;
```

```
{ int, int, double, char[64] }
```

```c
typedef struct foo {
    int a;
    int b;
    double c;
    char d[64];
} Foo;
```

```
{ int, int, double, char[64] }
```

To a  , these are the same thing

You could use *abcd*, but why do that to yourself?

Author: Alan Chu (ualch9@gmail.com)