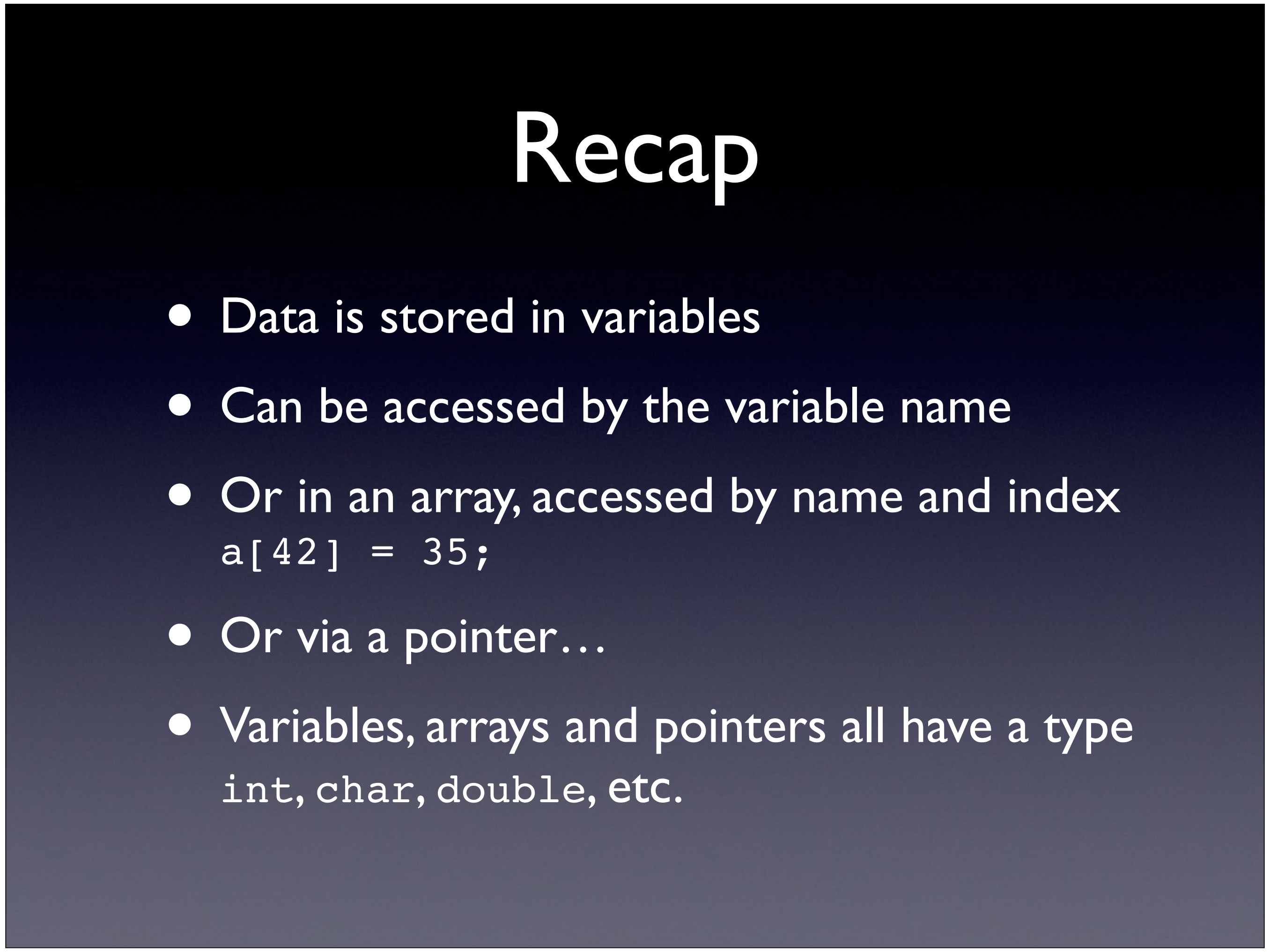
Structures

Steven R. Bagley



Recap

- Data is stored in variables
- Can be accessed by the variable name
- Or in an array, accessed by name and index
  `a[42] = 35;`
- Or via a pointer…
- Variables, arrays and pointers all have a type
  `int`, `char`, `double`, etc.

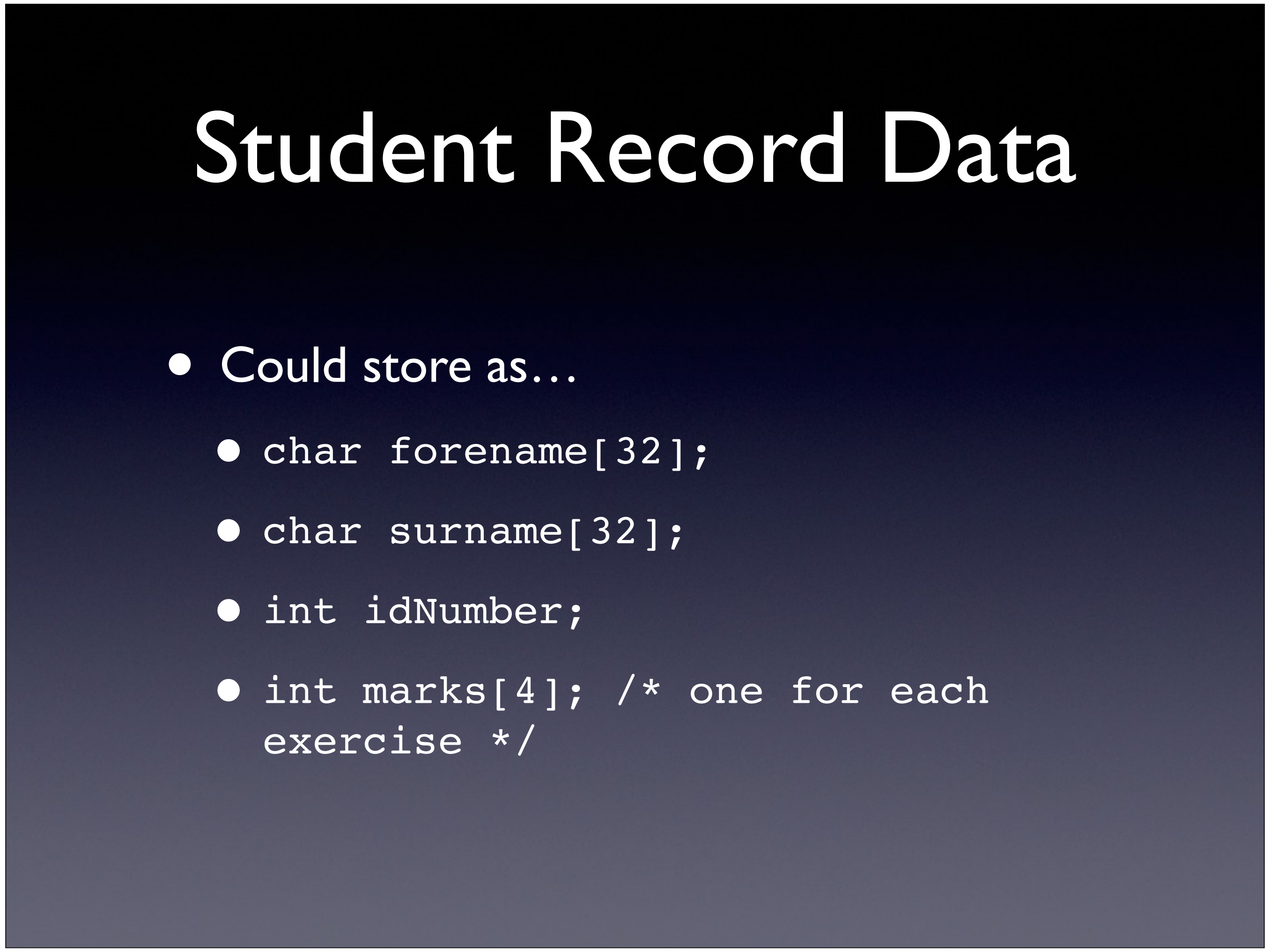
Going to look at another way of storing data

Organizing Data

- Often the data we need to store is not just a single value
- And may also need to be of different types
- Classic example of this would be storing a student record of G51PRG marks

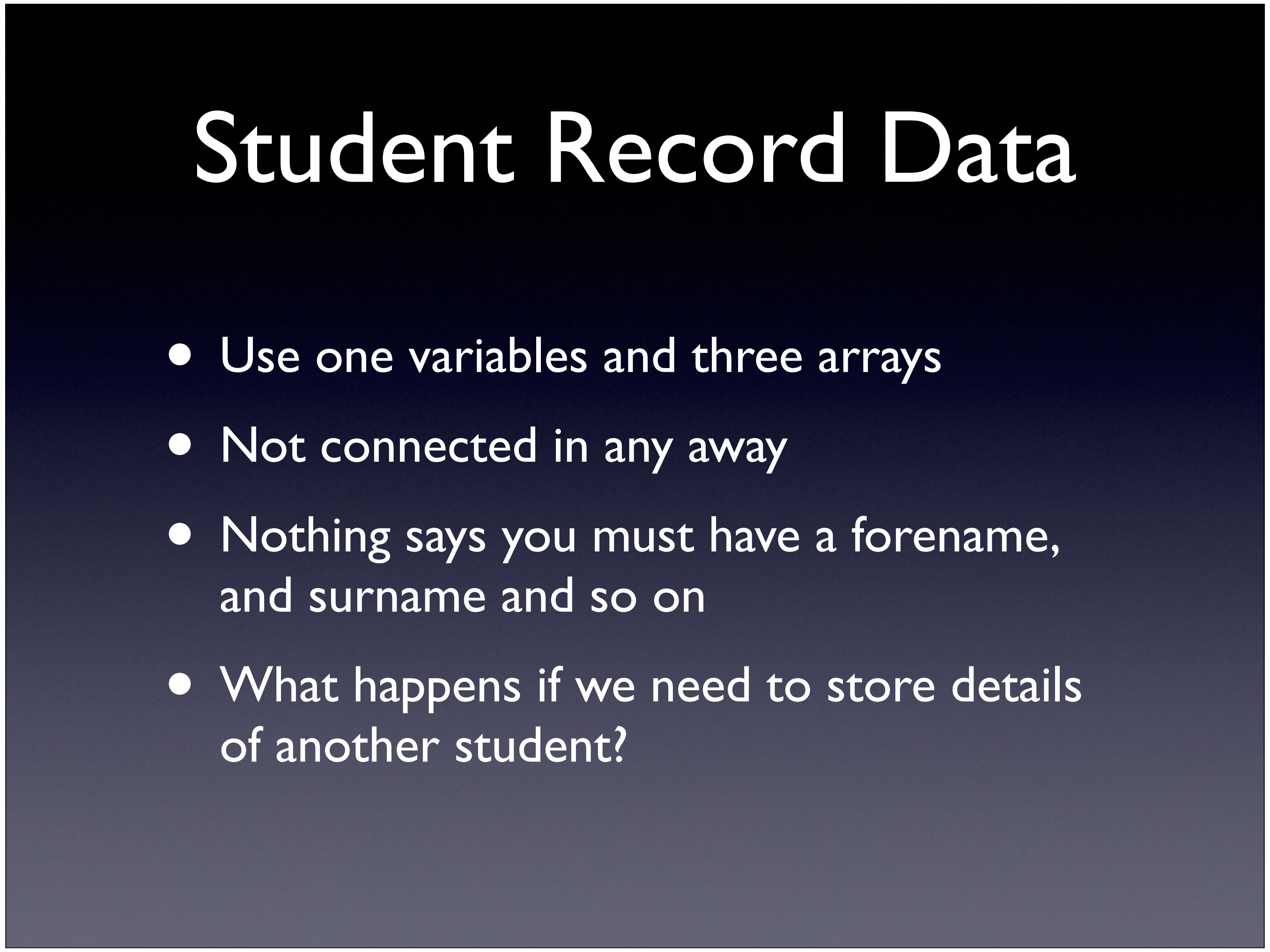
Student Record Data

- Would want to store
  - Forename
  - Surname
  - Student ID number
  - Marks for each exercise



Student Record Data

- Could store as…
  - `char forename[32];`
  - `char surname[32];`
  - `int idNumber;`
  - `int marks[4]; /* one for each exercise */`



Student Record Data

- Use one variables and three arrays
- Not connected in any away
- Nothing says you must have a forename, and surname and so on
- What happens if we need to store details of another student?

Only connected in the mind of the programmer.

Multiple students

- Could store in an array…
  - `char forename[150][32];`
  - `char surname[150][32];`
  - `int idNumber[150];`
  - `int marks[150][4]; /* one for each exercise */`

Multiple students

- Solved the naming problem (refer to it by number)
- But data is still not connected…
- Worse now than just using five variables
- If we get the index wrong when updating a student, we corrupt another student

I am not a number, I’m a free man...

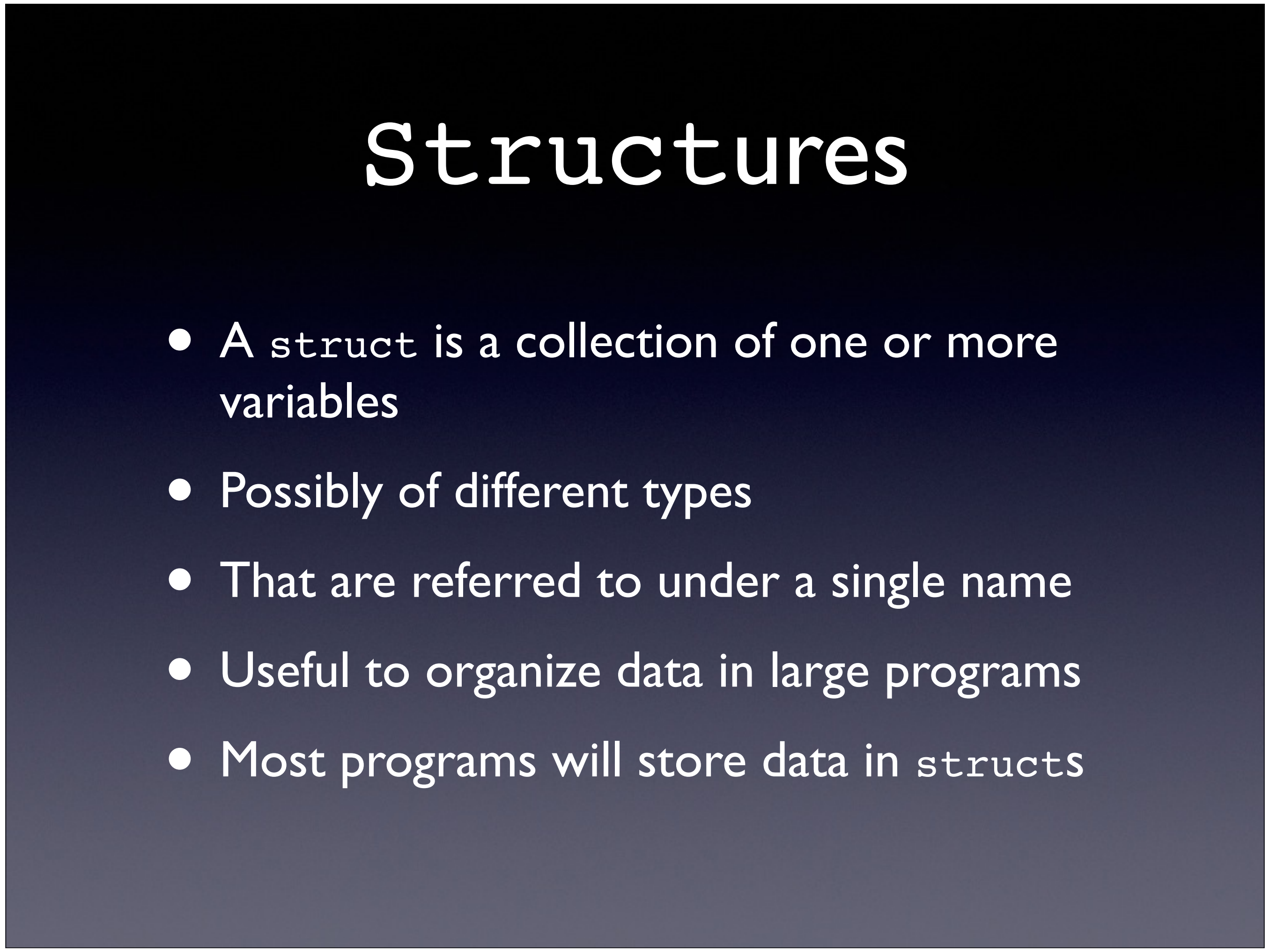
Multiple students

- Nightmare to pass to a function…

```
int
ProcessStudents(int numStudents,
                char forename[][32],
                char surname[][32],
                int idNumbers[],
                int marks[][5]);
```

Collating Data

- Need to be able to define a *student* as having various properties
- Then refer to the *student* as an entity in its own right
- C provides a mechanism to do this, the `struct`



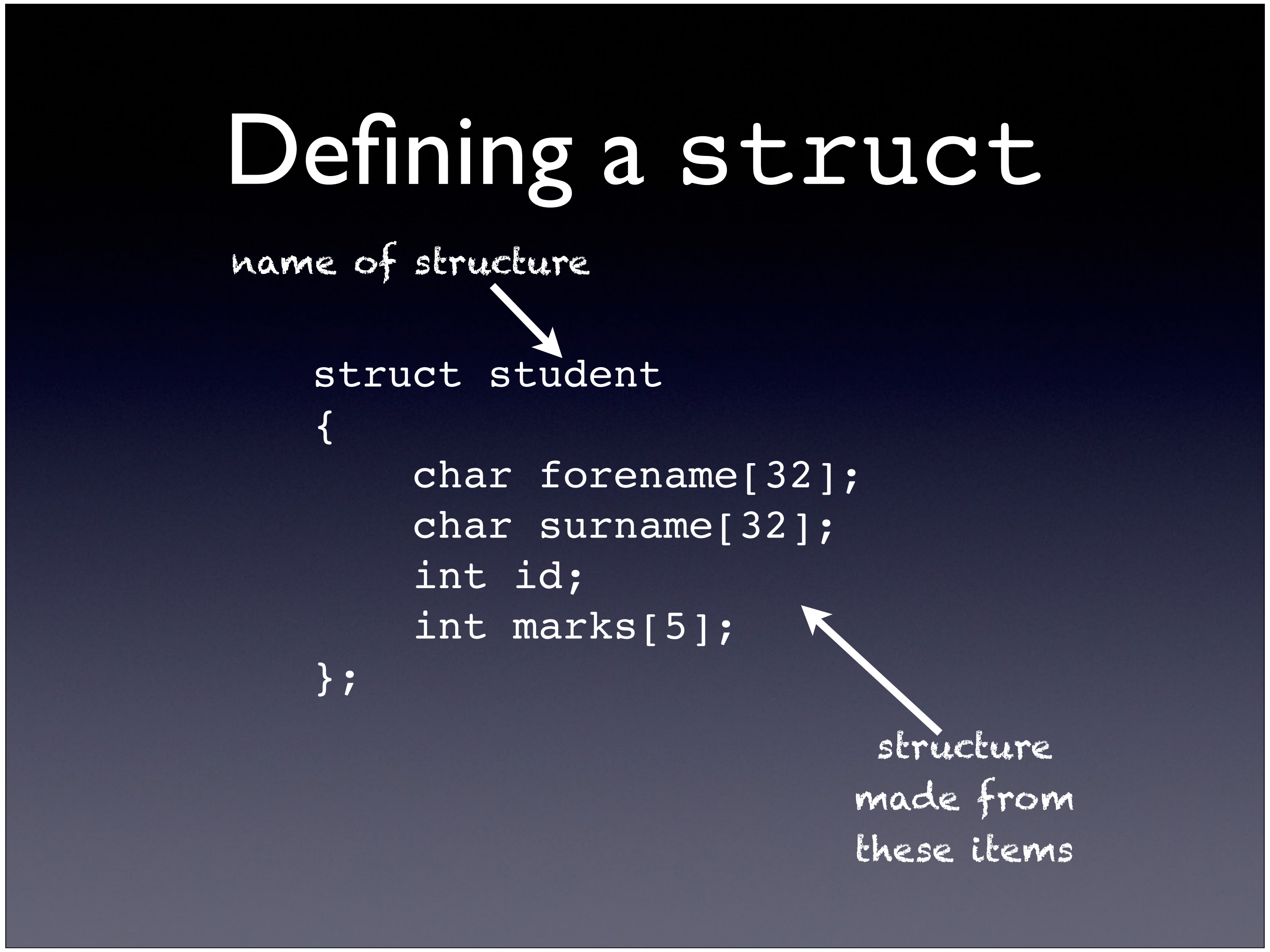
Structures

- A `struct` is a collection of one or more variables
- Possibly of different types
- That are referred to under a single name
- Useful to organize data in large programs
- Most programs will store data in `struct`s

AKA a record in some languages (notably pascal)

Defining a `struct`

- Specify a name to represent the type of struct
- Can have several in use in a program
- Then you specify the variables that make up its parts with curly braces



Defining a `struct`

name of structure

```
struct student
{
    char forename[32];
    char surname[32];
    int id;
    int marks[5];
};
```

structure
made from
these items

Allocates space for 64 characters (32+32) and 6 ints

Defining a `struct`

- `struct` can contain any data type, including primitives, arrays, pointers and even other `struct`s
- To use a `struct`, we have to create one just as with any variables

```
struct student steve;
```

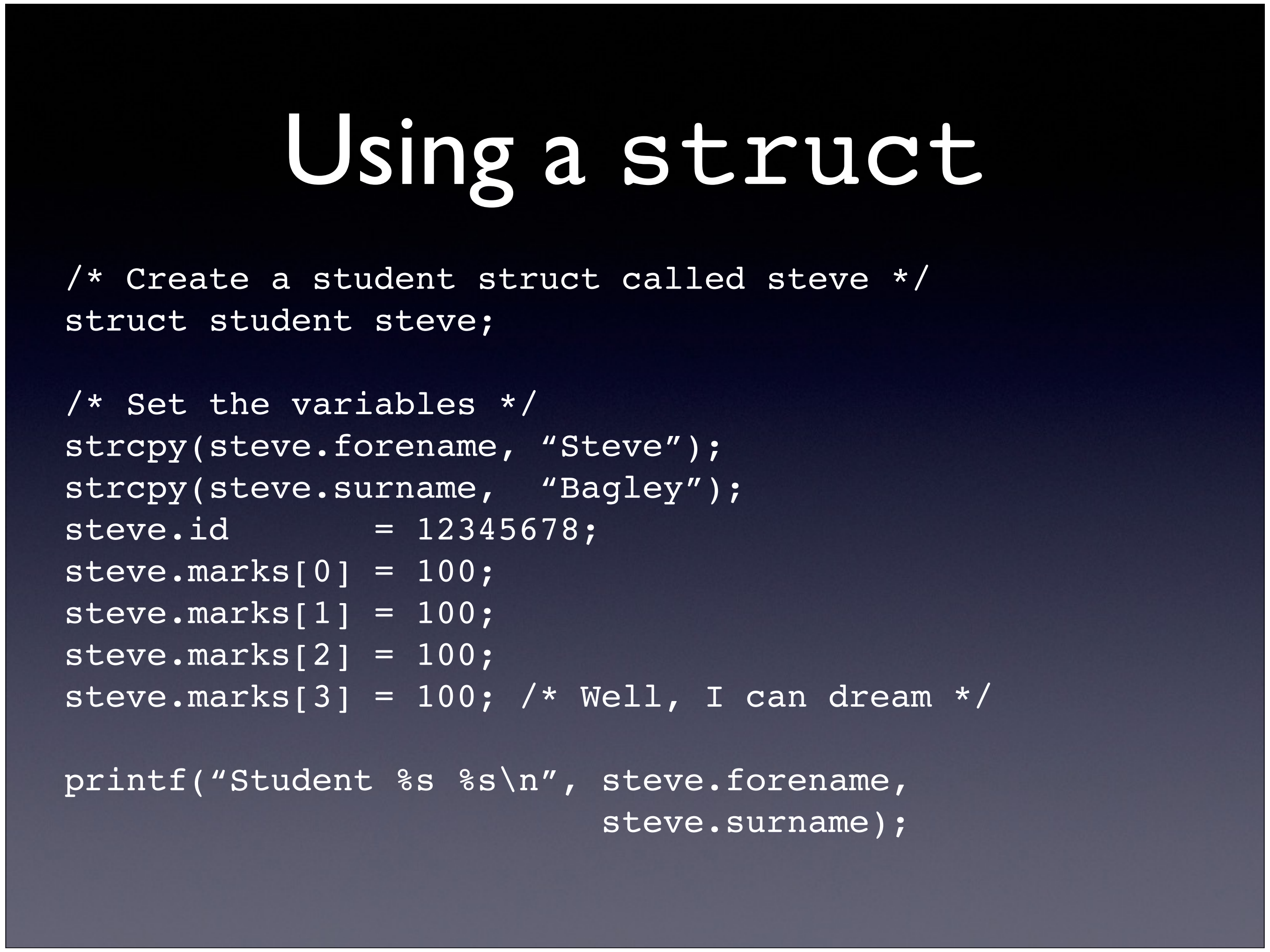
- Would create space for a `student struct` and give it the name `steve`

`structs` and Memory

- Note that the items in a struct are laid out consecutively in memory
- Padded to ensure word-alignment
- Accessed by offsets from the base address
- This means that if you go past the end of an array you will overwrite something else in the `struct`

Accessing `struct` data

- To access the data stored in a `struct`, you give the name of the struct (e.g. `steve`)
- Name of the variable you want to access (e.g. `id`) after it (separated by a `.`)
- So `steve.id` would access the id variable in the `struct` named `steve`
- Variable must exist or you'll get a compile error



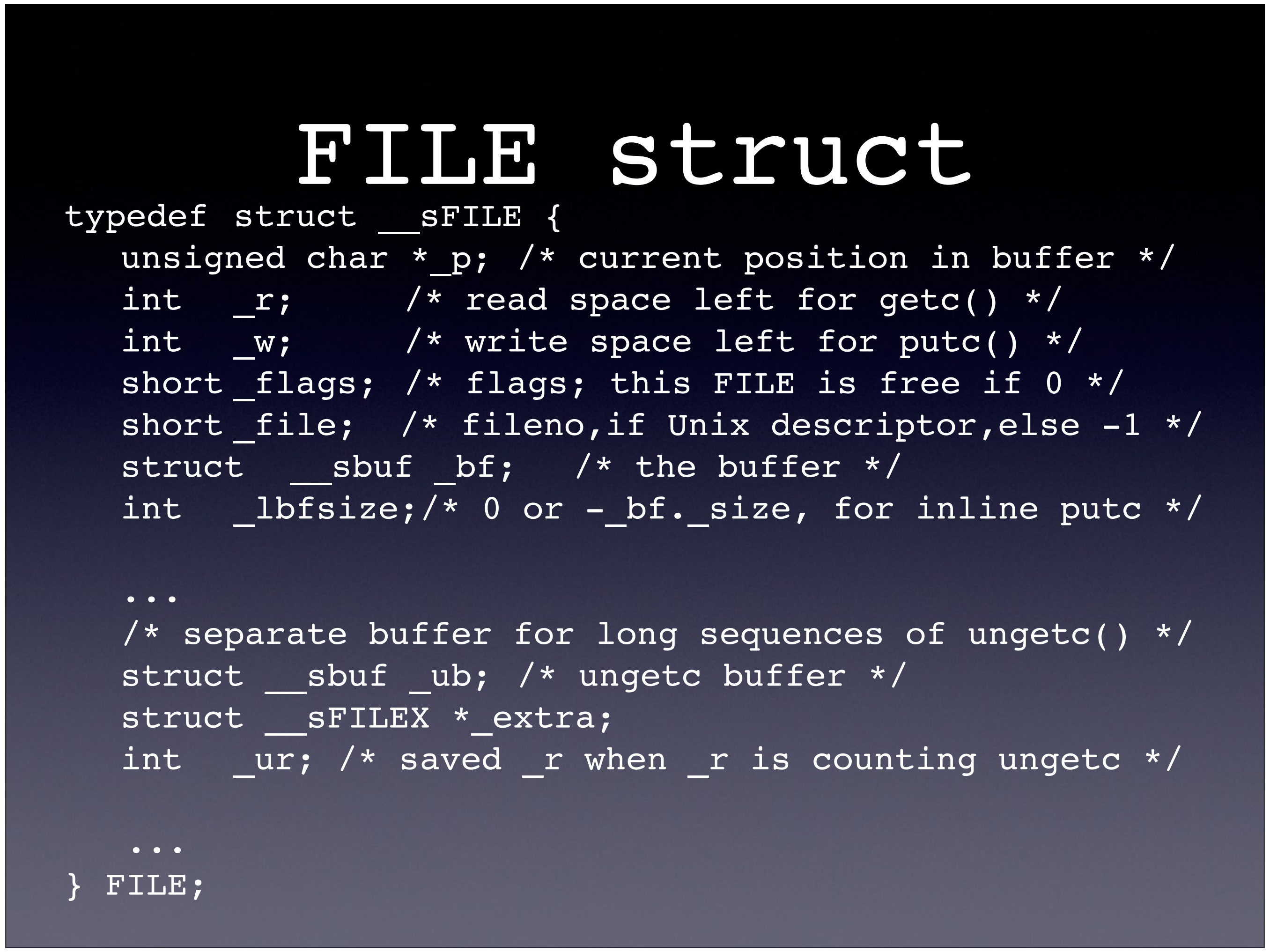
Using a `struct`

```
/* Create a student struct called steve */
struct student steve;

/* Set the variables */
strcpy(steve.forename, “Steve”);
strcpy(steve.surname,  “Bagley”);
steve.id       = 12345678;
steve.marks[0] = 100;
steve.marks[1] = 100;
steve.marks[2] = 100;
steve.marks[3] = 100; /* Well, I can dream */

printf(“Student %s %s\n”, steve.forename,
                          steve.surname);
```

Note we have to copy the strings, we can’t just assign



FILE struct

```
typedef struct __sFILE {
    unsigned char *_p; /* current position in buffer */
    int   _r;      /* read space left for getc() */
    int   _w;      /* write space left for putc() */
    short _flags;  /* flags; this FILE is free if 0 */
    short _file;  /* fileno,if Unix descriptor,else -1 */
    struct  __sbuf _bf;   /* the buffer */
    int   _lbfsize;/* 0 or -_bf._size, for inline putc */

    ...
    /* separate buffer for long sequences of ungetc() */
    struct __sbuf _ub; /* ungetc buffer */
    struct __sFILEX *_extra;
    int   _ur; /* saved _r when _r is counting ungetc */

    ...
} FILE;
```

Note the typedef -- this means we can just use FILE as if it were a type