struct

Define

```
#include <stdbool.h>

struct student {
    int age;
    const char *name;
    double gpa;
    bool ugrad;
};

int
main(void)
{
    /* … */
}
```

Declare

```
int
main(void)
{
    struct student a;
}
```

Initialize (positional, all fields)

```
int
main(void)
{
    struct student a = {23, “Alice”, 3.9, false}
}
```

Initialize (positional, first field(s))

```
int
main(void)
{
    struct student a = {23, “Alice”};
}
```

Remaining fields zero’d

(depending on compiler flags, may elicit a warning)

Initialize (by field name)

```
int
main(void)
{
    struct student a = {
            .gpa = 3.9,
            .name = “Alice”};
}
```

Field order is irrelevant

Remaining fields zero’d

Initialize (zero’d, implicit)

```
int
main(void)
{
	struct student a = {0};   /* or {.age = 0}; */
}
```

Remaining fields zero’d
(depending on compiler flags, may elicit a warning)

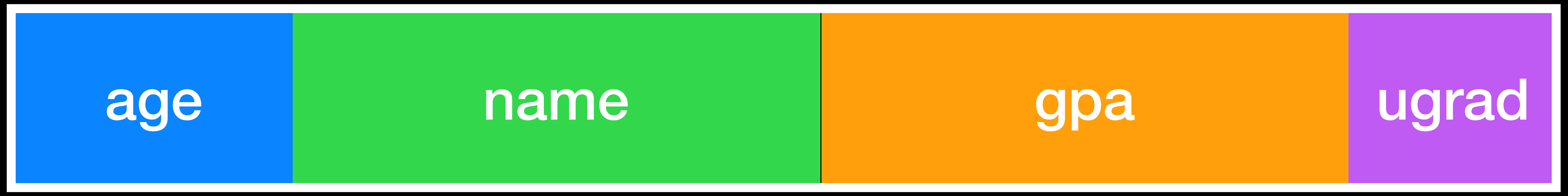
Initialize (zero’d, explicit)

```
#include <string.h>

int
main(void)
{
    struct student a;

    memset(&a, 0x00, sizeof(a));
}
```

Memory representation

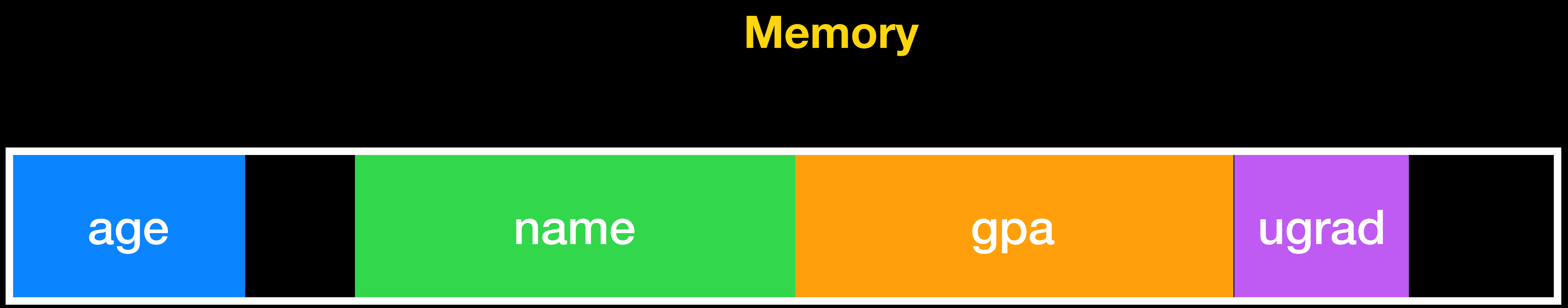
```
struct student {
    int age;
    const char *name;
    double gpa;
    bool ugrad;
};

int
main(void)
{
    struct student a;
}
```

Memory



Memory representation

```
struct student {
    int age;
    const char *name;
    double gpa;
    bool ugrad;
};

int
main(void)
{
    struct student a;
}
```



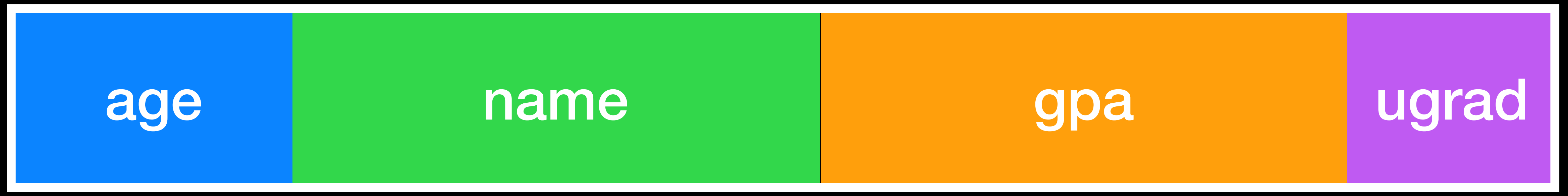
Compiler can add padding for field alignment purposes

Memory representation

```
struct student {
    int age;
    const char *name;
    double gpa;
    bool ugrad;
} __attribute__((packed));

int
main(void)
{
    struct student a;
}
```

Do not add padding.
(Only really useful if struct will be serialized to disk/network)

Memory



https://gcc.gnu.org/onlinedocs/gcc/Common-Type-Attributes.html

Accessing fields

```
#include <stdio.h>

int
main(void)
{
    struct student a;

    a.gpa = 3.9;
    printf(“gpa= %.2f\n”, a.gpa);
}
```

To the compiler, gpa is just a fixed offset within the struct.

structs are lvalues

```
#include <stdio.h>

int
main(void)
{
    struct student a = { .gpa = 3.9}, b;

    b = a;
    if (a.gpa == b.gpa)
        puts(“structs (shallow) copied”);
}
```

All fields of the struct are memcpy’d to lvalue on assignment

Passing structs to functions

```
#include <stdio.h>

void
make_older(struct student s)
{
    s.age += 1;
}

int
main(void)
{
    struct student a = { .age = 20 };

    make_older(a);
    printf(“%d\n”, a.age); /* still prints 20 */
}
```

Copied by value (like all arguments in C)

Pointers and structs

```
#include <stdio.h>

void
make_older(struct student *s)
{
    s->age += 1;
}

int
main(void)
{
    struct student a = { .age = 20 };

    make_older(&a);
    printf(“%d\n”, a.age); /* now prints 21 */
}
```

`(*s).age and s->age are equivalent, but -> is clearer syntax`

Using typedef with struct

```
typedef struct student {
    int age;
    const char *name;
    double gpa;
    bool ugrad;
} student;
```

or

```
struct student {
    int age;
    const char *name;
    double gpa;
    bool ugrad;
};

typedef struct student student;
```

```
int
main(void)
{
    student s;
    struct student t;
}
```

The types `student` and `struct  student` are the same