Lecture 12

Records and its variants

Records

- Data needs to be described with various attributes of various types
- Grouping of relevant information through different variables forms a record
- All variables have all the properties common (fixed, hence fixed set of fields)
- Operation on record data type is “selection of a field by name”

struct (in C)

User defined type

```
typedef struct {
    char name[15];
    char ID[11]
    int age;
    char discipline[15];
    float cgpa;
} student_rec;
student_rec s;
```

Access to individual field

```
s.name
s.ID
s.age
s.discipline
s.cgpa
```

Layout of the record data type

- Each field of a record has its own type, hence individual corresponding layout
- Whatsoever be the complex structure of a record, the layout can be known at compile time e.g.

```
typedef struct {
        char name[10];
        char ID[7];
        int age;
        float cgpa;
} student_rec;
student_rec s;
```

Example: Field sizes and size of the record variable

```c
#include <stdio.h>

int main()
{
    typedef struct {
            char name[12];
            char ID[7];
            int age;
            float cgpa;
        } student_rec;
        student_rec s;
        printf("%d\n", sizeof(s.name));
        printf("%d\n", sizeof(s.ID));
        printf("%d\n", sizeof(s.age));
        printf("%d\n", sizeof(s.cgpa));
        printf("%d\n", sizeof(s));
        return 0;
}
```

Output

```
12
7
4
4
28
```

Understand starting address of the record variable and its fields

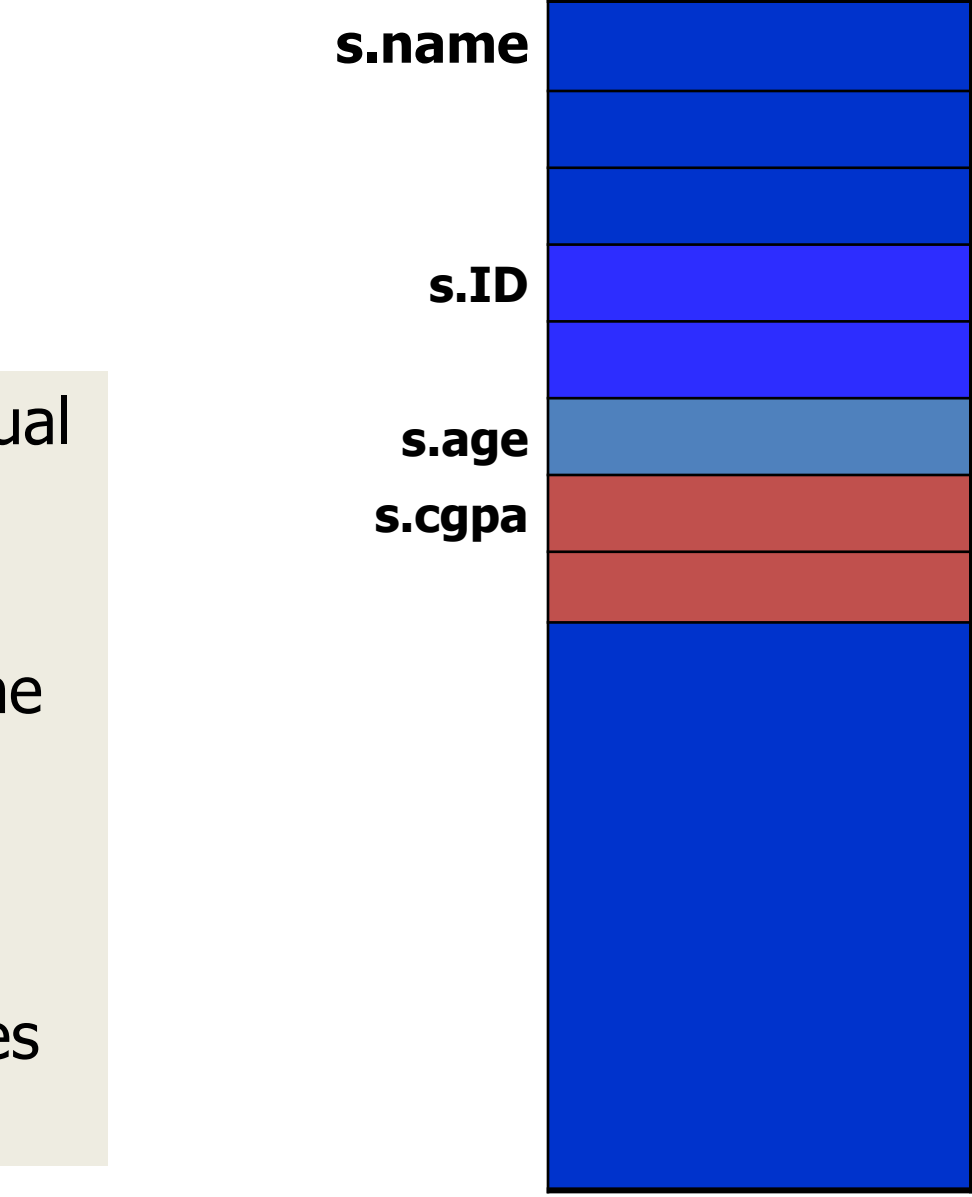
```c
#include <stdio.h>

int main()
{
    typedef struct {
            char name[12];
            char ID[7];
            int age;
            float cgpa;
        } student_rec;
        student_rec s;
        printf("%d %u\n", sizeof(s.name), &s.name);
        printf("%d %u\n", sizeof(s.ID), &s.ID);
        printf("%d %u\n", sizeof(s.age), &s.age);
        printf("%d %u\n", sizeof(s.cgpa), &s.cgpa);
        printf("%d %u\n", sizeof(s), &s);
        return 0;
}
```

Output

```
12 2984457264
7 2984457276
4 2984457284
4 2984457288
28 2984457264
```

Example layout

- Each record gets storage equal to the sum of sizes of fields (aligned to words)
- The field name appears in the same order as they appear in the definition
- Data is aligned to 4 addressable bytes (some bytes remain unused)

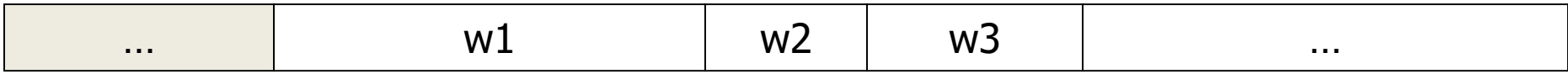


Layout of record data type

- All the fields of a value of record type are laid out together
- If the record structure is constructed as
  - Record {

    type1 field1; type2 field 2 ; type3 field3;

    }
- If w1, w2, w3 are the sizes of type1,2 and 3 respectively then the data type is laid out as



Type signature (expression) of the record data type

- Type of a record data is defined as the Cartesian products of the fields
- Type of a variable of following record data type is defined as int x int

```
struct complex {
        int real;
        int imaginary;
}
```

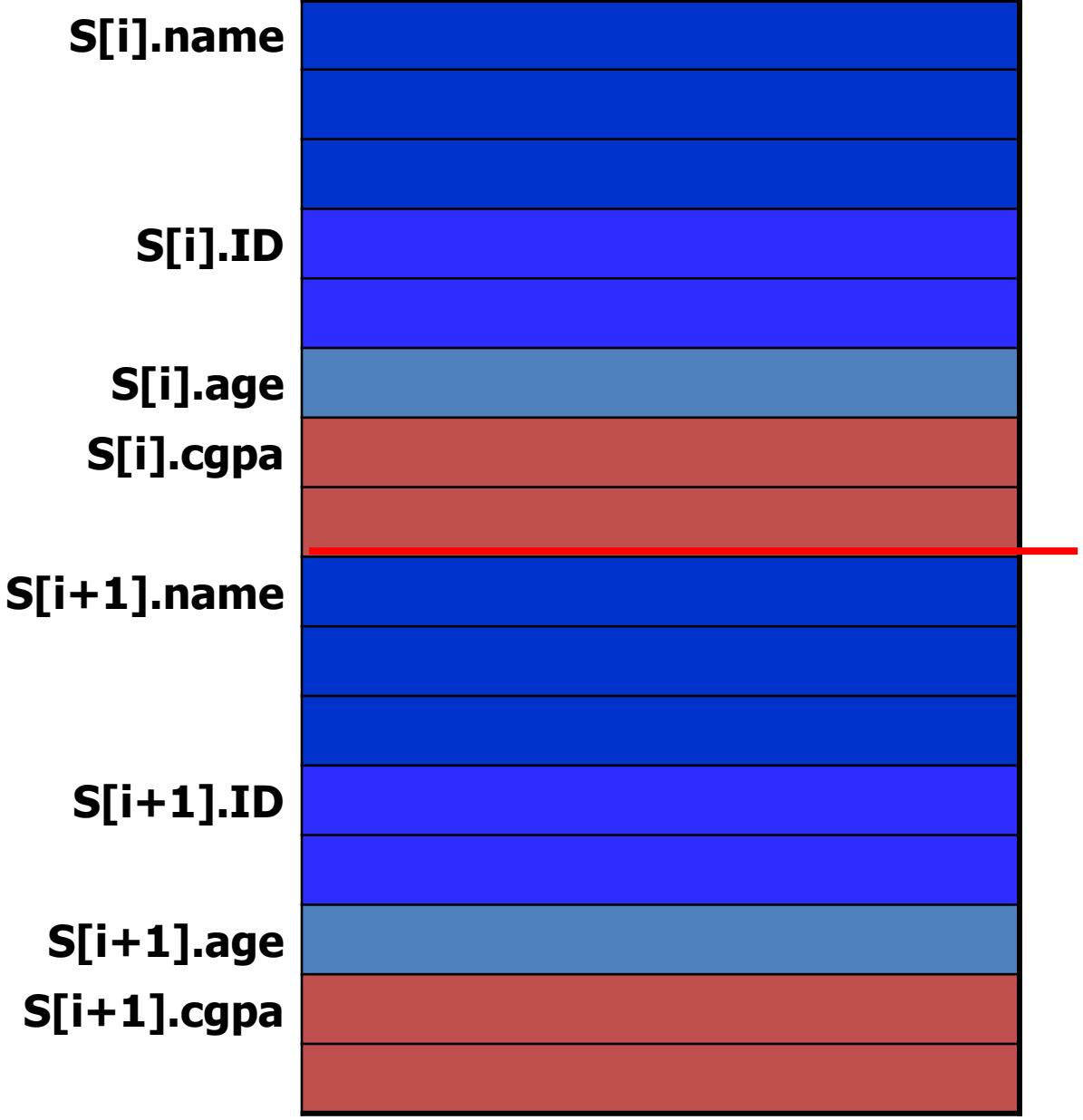
- Type of the student_rec record data type is defined as

array(12, char) x array(7, char) x int x float

Layout of array of records

- Each record is laid as a sequence of contiguous locations with the respective fields laid as discussed
- All records in the array are laid as a sequence of contiguous blocks of memory
- Example student_Rec S[10]

Offset can be computed at compile time



Home work: compute the relative offset of S[i+1].age

Comparison of Arrays and Records

- array – homogeneous collection of elements
- record- heterogeneous collection of elements

- **array element – A[i] can change at run time**
- **record- field (s.name) is fixed at compile time**

- **array element –selected by indices anytime**
- **record- selected by names that are known at compile time**

Union type variable and sizes of its field data

```c
#include <stdio.h>

int main()
{
    typedef union {
            char name[12];
            char ID[7];
            int age;
            float cgpa;
        } student_rec;
        student_rec s;
        printf("%d %u\n", sizeof(s.name), &s.name);
        printf("%d %u\n", sizeof(s.ID), &s.ID);
        printf("%d %u\n", sizeof(s.age), &s.age);
        printf("%d %u\n", sizeof(s.cgpa), &s.cgpa);
        printf("%d %u\n", sizeof(s), &s);
        return 0;
}
```

Output

```
12 612403476
7 612403476
4 612403476
4 612403476
12 612403476
```

Issues with data access