Structures and Unions in C

Objectives

Be able to use compound data structures in programs

Be able to pass compound data structures as function arguments, either by value or by reference

Be able to do simple bit-vector manipulations

Structures

Compound data:

```c
#include <stdio.h>
#include<string.h>
struct new1
{
     char name[10];
     int rollno;
     int id;

}s;
int main()
{
    strcpy(s.name,"abc");
    s.rollno=1;
    s.id=1;
    printf("%s %d %d",s.name,s.rollno,s.id);
    return 0;
}
```

Structure Representation & Size

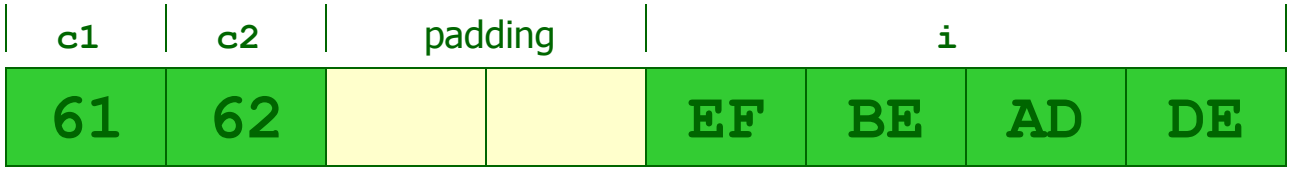
`sizeof(struct …)` =

sum of `sizeof(`field`)`

\+ **alignment padding**

Processor- and compiler-specific

```
struct CharCharInt {
    char  c1;
    char  c2;
    int   i;
} foo;

foo.c1 = 'a';
foo.c2 = 'b';
foo.i  = 0xDEADBEEF;
```



Typedef

Mechanism for creating new type names

- **New names are an alias for some other type**
- ***May* improve the portability and/or clarity of the program**

```
typedef long int64_t;
typedef struct ADate {
    int month;
    int day;
    int year;
} Date;

int64_t i = 100000000000;
Date d = { 2, 4, 2021 };
```

Overload existing type names for portability

Simplify complex type names

Structures

Compound data:

```c
#include <stdio.h>
#include<string.h>
struct new1
{
    char name[10];
    int rollno;
    int id;

}s;
int main()
{
   printf("Enter name, rollno,id");
   scanf("%s",s.name);
   scanf("%d",&s.rollno);
   scanf("%d",&s.id);
   printf("%s %d %d",s.name,s.rollno,s.id);
   return 0;
}
```

Unions

A union value doesn't "know" which case it contains

```
union AnElt {
   int   i;
   char  c;
} elt1, elt2;

elt1.i = 4;
elt2.c = 'a';
elt2.i = 0xDEADBEEF;

if (elt1 currently has a char) ...
```


How should your program keep track whether `elt1`, `elt2` hold an `int` or a `char`?

Basic answer: Another variable holds that info

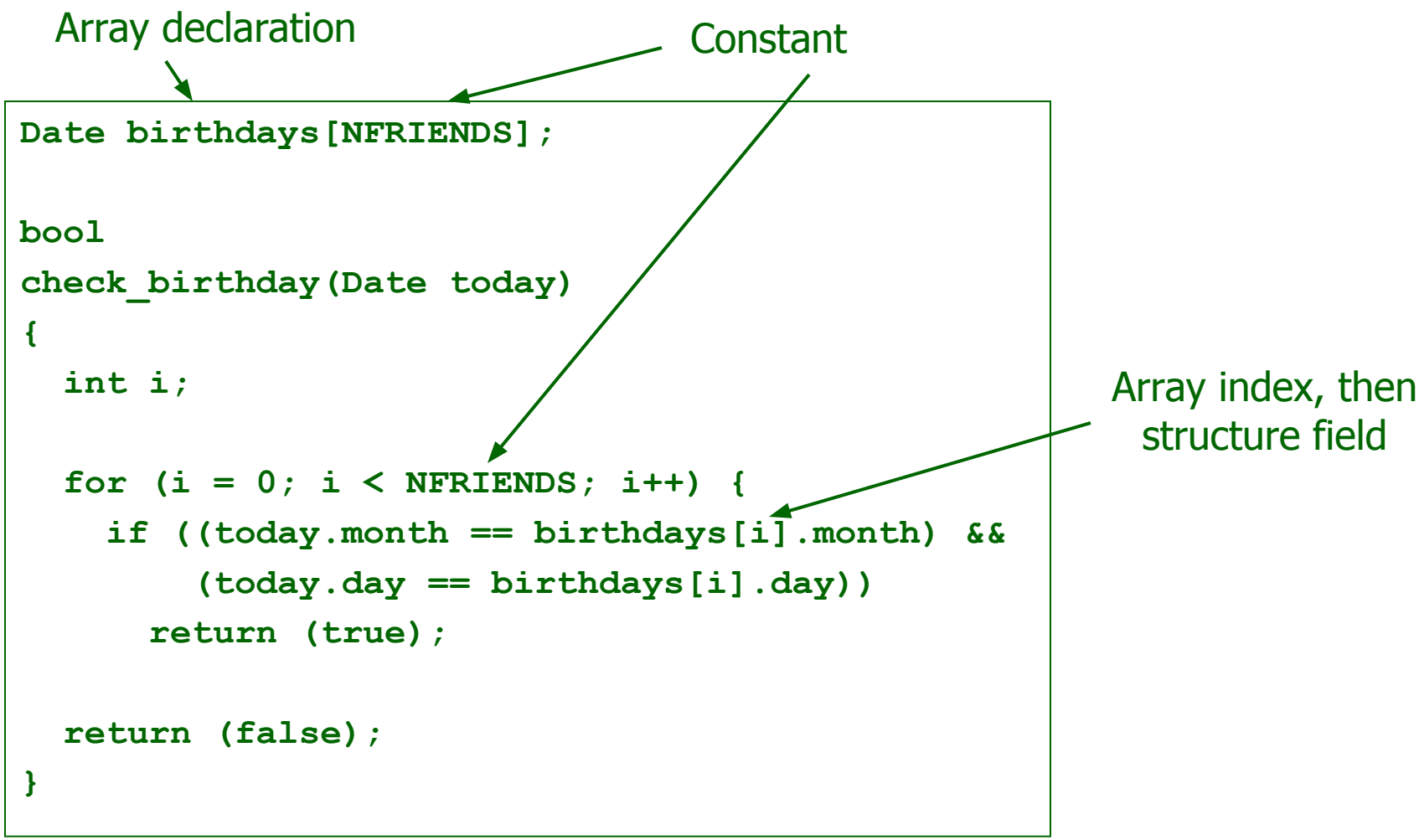
Structures vs Unions

```
void main()
{

    struct s s1={38,"riya"};
    union u u1={40,"raj"};


    printf("structure data  %d  name %s: \n",
                s1.data, s1.name);
    printf("union data  %d  name %s: \n",
                u1.data, u1.name);
    printf("\nsizeof structure : %d\n", sizeof(s1));
    printf("\nsizeof union : %d\n", sizeof(u1));

    // difference five
    printf("\n Accessing all members at a time:");
    s1.data = 39;
    strcpy(s1.name, "computer");

     printf("structure data  %d  name %s: \n",
                s1.data, s1.name);

      u1.data = 39;
    strcpy(u1.name, "computer");


    printf("union data  %d  name %s:\n",
                u1.data, u1.name);


}
```

Structures

```
void main()
{

    struct s s1={38,"riya"};
    union u u1={40,"raj"};


    printf("structure data  %d  name %s: \n",
                s1.data, s1.name);
    printf("union data  %d  name %s: \n",
                u1.data, u1.name);
    printf("\nsizeof structure : %d\n", sizeof(s1));
    printf("\nsizeof union : %d\n", sizeof(u1));

    // difference five
    printf("\n Accessing all members at a time:");
    s1.data = 39;
    strcpy(s1.name, "computer");

     printf("structure data  %d  name %s: \n",
                s1.data, s1.name);

      u1.data = 39;
    strcpy(u1.name, "computer");


    printf("union data  %d  name %s:\n",
                u1.data, u1.name);


}
```

Arrays of Structures




```
Date birthdays[NFRIENDS];

bool
check_birthday(Date today)
{
  int i;

  for (i = 0; i < NFRIENDS; i++) {
    if ((today.month == birthdays[i].month) &&
        (today.day == birthdays[i].day))
      return (true);

  return (false);
}
```

Pointers to Structures

```
Date
create_date1(int month,
             int day,
             int year)
{
  Date d;

  d.month = month;
  d.day   = day;
  d.year  = year;

  return (d);
}
```

```
void
create_date2(Date *d,
             int month,
             int day,
             int year)
{
  d->month = month;
  d->day   = day;
  d->year  = year;
}
```

Pass-by-reference

Copies date

```
Date today;

today = create_date1(2, 4, 2021);
create_date2(&today, 2, 4, 2021);
```