CSCI 2132: Software Development

Structs, Unions, and Enums

Norbert Zeh

Faculty of Computer Science
Dalhousie University

Winter 2019

Structs

Combine constant number of v pes into a record

Elements are called **members**

```
struct student {
  unsigned int number;
  char name[32];
  char username[9];
};
```

Members referred to by name

Contrast with arrays:

- Arbitrary number of values
- All the same type

Important uses:

- Representing records in data collections
- Building block for data structures

Dot Operator

Struct members are accessed using dot notation, similar to Java:

```
struct student {
  unsigned int num
  char nam
  char u
};

int main() {
  struct student s;
  s.number = 321459;
  strcpy(s.name, “A Student”);
  strcpy(s.username, “astudent”);
  printf(“Student number: %8d\n”, s.number);
  printf(“Name:           %s\n”, s.name);
  printf(“Username:       %s\n”, s.username);
  return 0;
}
```

Need to write struct here.
(A tad annoying.)

Arrow Operator

Arrow operator (`->`) combines pointer dereferencing and member access:

```
struct student {
  unsigned int number;
  char name[32];
  char username[9];
};

void print_name(struct student *s) {
  printf(“%s\n”, (*s).name);
}

int main() {
  struct student s;
  ...
  print_name(&s);
}
```

Arrow Operator

Arrow operator (`->`) combines pointer dereferencing and member access:

```
struct student {
  unsigned int number;
  char name[32];
  char username[9];
};

void print_name(struct student *s) {
  printf("%s\n", s->name);
}

int main() {
  struct student s;
  ...
  print_name(&s);
}
```

A Common Idiom to Avoid Typing

```
struct student {
  unsigned int number;
  char name[32];
  char username[9];
};

void print_name(struct student *s) {
  printf(“%s\n”, s->name);
}

int main() {
  struct student s;
  ...
  print_name(&s);
}
```

A Common Idiom to Avoid Typing

```
typedef struct {
  unsigned int number;
  char name[32];
  char username[9];
} student;

void print_name(student *s) {
  printf(“%s\n”, s->name);
}

int main() {
  student s;
  ...
  print_name(&s);
}
```

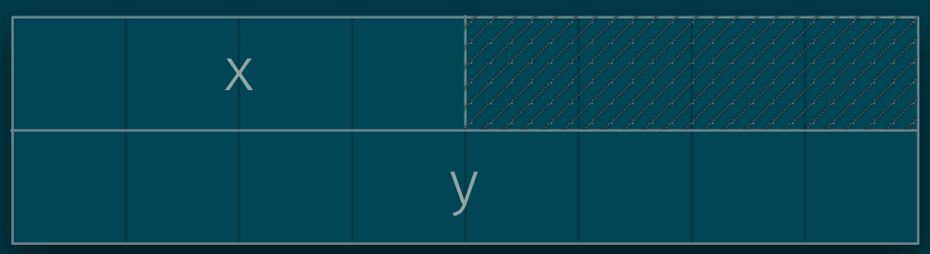
Unions

Struct stores all its members at the same time.

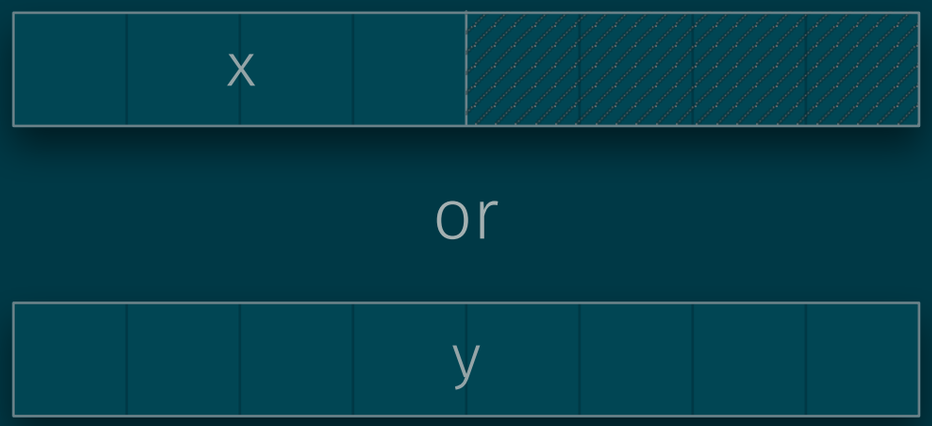
Union stores only one of its members at a time.

Size = maximum size of its members.

```
struct foo {
  int x;
  double y;
};
```




```
union bar {
  int x;
  double y;
};
```



Uses of Unions

Saving space:

```
struct student {
  int banner;
  char name[32];
  char faculty[32];
  char major[32];
  struct ac_record *record;
}
```

```
struct employee {
  int banner;
  char name[32];
  char faculty[32];
  int salary;
  struct emp_hist *history;
```

```
struct banner_record {
  int banner;
  char name[32];
  char faculty[32];
  char major[32];
  int salary;
  struct ac_record *record;
  struct emp_hist *history;
}
```

Uses of Unions

Saving space:

```
struct student_details {
  char major[32];
  struct ac_record *record;
};

struct employee_details {
  int salary;
  struct emp_hist *history;
};
```

Uses of Unions

Saving space:

```
struct student_details {
  char major[32];
  struct ac_record *record;
};

struct employee_details {
  int salary;
  struct emp_hist *history;
};

union details {
  struct student_details;
  struct employee_details;
};
```

Uses of Unions

Saving space:

```
struct student_details {
  char major[32];
  struct ac_record *record;
};

struct employee_details {
  int salary;
  struct emp_hist *history;
};

union details {
  struct student_details;
  struct employee_details;
};
```

```
struct stu
  int banner;
  char name[32
  char facult [32];
  int type;
  union details dtls;
}
```

Need a way to figure out the record type.

Uses of Unions

Re-interpreting bit content:

```
union converter {
  double d;
  unsigned long int bits;
};

int main() {
  union converter c;
  c.d = 5.312e2;
  printf("%lx\n", c.bits);
  return 0;
}
```

Enums

Are very similar to their use in Java:

```
enum color {
  RED,
  GREEN,
  BLUE
};

int main() {
  enum color col = BLUE;
  printf(“%d\n”, col);
  return 0;
}
```

```
enum flags {
  READABLE = 0x04,
  WRITABLE = 0x02,
  EXECUTABLE = 0x01
};

int main() {
  enum flags fs =
    READABLE | WRITABLE;
  printf(“%d\n”, fs);
  return 0;
}
```

They are just integers in disguise in C.