Records

CS 115

Dr. Joseph Eremondi, adapted from Dr. Shakil Khan, Dr. Philip Fong, and Dr. Howard Hamilton

Last updated: January 8, 2025

Structs and unions

Motivation

Motivation

Title	string
Author	string
Publisher	string
Year	unsigned int
Call Number	string
Price	double

Motivation

- E.g. Catalog information in a library

Title	string
Author	string
Publisher	string
Year	unsigned int
Call Number	string
Price	double

Motivation

- E.g. Catalog information in a library
- Data in collection is heterogenous

Title	string
Author	string
Publisher	string
Year	unsigned int
Call Number	string
Price	double

Motivation

- E.g. Catalog information in a library
- Data in collection is heterogenous

Title	string
Author	string
Publisher	string
Year	unsigned int
Call Number	string
Price	double

Motivation

- E.g. Catalog information in a library
- Data in collection is heterogenous

Title	string
Author	string
Publisher	string
Year	unsigned int
Call Number	string
Price	double

- Solution using arrays:

Motivation

- E.g. Catalog information in a library
- Data in collection is heterogenous

Title	string
Author	string
Publisher	string
Year	unsigned int
Call Number	string
Price	double

- Solution using arrays:

Motivation

- E.g. Catalog information in a library
- Data in collection is heterogenous

Title	string
Author	string
Publisher	string
Year	unsigned int
Call Number	string
Price	double

- Solution using arrays:

```
string titles[N];
string authors[N];
string publishers[N];
unsigned int publishingYears[N];
string callNumbers[N];
double Price[N];
```

- Poor choice of interface!

Motivation

- E.g. Catalog information in a library
- Data in collection is heterogenous

Title	string
Author	string
Publisher	string
Year	unsigned int
Call Number	string
Price	double

- Solution using arrays:

```
string titles[N];
string authors[N];
string publishers[N];
unsigned int publishingYears[N];
string callNumbers[N];
double Price[N];
```

- Poor choice of interface!
- (many arguments to pass for functions)

Use a record instead!

Use a record instead!

Use a record instead!

- Data can be heterogenous

Use a record instead!

- Data can be heterogenous
- Define:

Use a record instead!

- Data can be heterogenous
- Define:

Use a record instead!

- Data can be heterogenous
- Define:

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
};
```

Use a record instead!

- Only 1 argument needs to be passed

- Data can be heterogenous
- Define:

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
};
```

Use a record instead!

- Data can be heterogenous
- Define:

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
};
```

- Only 1 argument needs to be passed
- Declare:

Use a record instead!

- Data can be heterogenous
- Define:

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
};
```

- Only 1 argument needs to be passed
- Declare:

Use a record instead!

- Data can be heterogenous
- Define:

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
};
```

- Only 1 argument needs to be passed
- Declare:

```
struct CatalogEntry c;
// or, equivalently this:
CatalogEntry c;
```

- Initialize:

Use a record instead!

- Data can be heterogenous
- Define:

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
};
```

- Only 1 argument needs to be passed
- Declare:

```
struct CatalogEntry c;
// or, equivalently this:
CatalogEntry c;
```

- Initialize:

Use a record instead!

- Data can be heterogenous
- Define:

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
};
```

- Only 1 argument needs to be passed
- Declare:

```
struct CatalogEntry c;
// or, equivalently this:
CatalogEntry c;
```

- Initialize:

```
c.title = "Peter Pan";
c.author = "J. M. Barrie";
c.publisher = "Scribner";
c.publishingYear = 1980;
c.callNumber = "B2754 1980";
```

Initializing a Record

- As with arrays

Initializing a Record

- As with arrays

Initializing a Record

- As with arrays

```
CatalogEntry c = {"Peter Pan",
                  "J. M. Barrie",
                  "Scribner",
                  1980,
                  "B2754 1980"};
```

Copying a Record

Copying a Record

```
// initialization list
CatalogEntry c = { ... };

// initialization by copying
CatalogEntry c1 = c;

// default initialization
CatalogEntry c2;
// assignment operator
c2 = c;
```

Functions operating on records

Functions operating on records

```
void printCatalogEntry(CatalogEntry c){
  cout << "Title: " << c.title << endl;
  cout << "Author: " << c.author << endl;
  cout << "Publisher: " << c.publisher << endl;
  cout << "Publishing Year: " << c.publishingYear << endl;
  cout << "Call Number: " << c.callNumber << endl;
}
```

- As usual, by default arguments are passed by value (call by value)

Functions operating on records

- For efficiency, call by reference is also supported

Functions operating on records

- For efficiency, call by reference is also supported

Functions operating on records

- For efficiency, call by reference is also supported

```
void printCatalogEntry(const CatalogEntry &c){
  cout << "Title: " << c.title << endl;
  cout << "Author: " << c.author << endl;
  cout << "Publisher: " << c.publisher << endl;
  cout << "Publishing Year: " << c.publishingYear << endl;
  cout << "Call Number: " << c.callNumber << endl;
}
```

Equality checking

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

```
bool CatalogEntryEquals(const CatalogEntry& c1, const CatalogEntry& c2){
```

- return

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

```
bool CatalogEntryEquals(const CatalogEntry& c1, const CatalogEntry& c2){
```

- return
- c1.title == c2.title &&

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

```
bool CatalogEntryEquals(const CatalogEntry& c1, const CatalogEntry& c2){
```

- return
- c1.title == c2.title &&
- c1.author == c2.author &&

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

```
bool CatalogEntryEquals(const CatalogEntry& c1, const CatalogEntry& c2){
```

- return
- c1.title == c2.title &&
- c1.author == c2.author &&
- c1.publisher == c2.publisher &&

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

```
bool CatalogEntryEquals(const CatalogEntry& c1, const CatalogEntry& c2){
```

- return
- c1.title == c2.title &&
- c1.author == c2.author &&
- c1.publisher == c2.publisher &&
- c1.publishingYear == c2.publishingYear &&

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

```
bool CatalogEntryEquals(const CatalogEntry& c1, const CatalogEntry& c2){
```

- return
- c1.title == c2.title &&
- c1.author == c2.author &&
- c1.publisher == c2.publisher &&
- c1.publishingYear == c2.publishingYear &&

Equality checking

```
if (c1 == c2)  // invalid
```

- As in the case for arrays, must do this each field at a time

```
bool CatalogEntryEquals(const CatalogEntry& c1, const CatalogEntry& c2){
```

- return
- c1.title == c2.title &&
- c1.author == c2.author &&
- c1.publisher == c2.publisher &&
- c1.publishingYear == c2.publishingYear &&

```
c1.callNumber == c2.callNumber;
}
```

Complex record data structures

Complex record data structures

```
CatalogEntry A[3];
```

- CatalogEntry A[] =

Complex record data structures

```
CatalogEntry A[3];
```

- CatalogEntry A[] =

Complex record data structures

```
CatalogEntry A[3];
```

- CatalogEntry A[] =

```
{
  {"Peter Pan",
      "J. M. Barrie",//
      "Scribner",//
      1980,//
      "B2754 1980"},



    {"C++ Primer",
        "Stanley B. Lippman",//
        "Addison-Wesley",//
        1998,//
```

Practise!

- See the very first announcement in UR Courses

Practise!

- See the very first announcement in UR Courses
- Try the exercises there

Practise!

- See the very first announcement in UR Courses
- Try the exercises there
    - declare a C++ struct to represent a point in the Cartesian coordinate system

Practise!

- See the very first announcement in UR Courses
- Try the exercises there
    - declare a C++ struct to represent a point in the Cartesian coordinate system
    - declare a C++ struct to represent a hexagon

Practise!

- See the very first announcement in UR Courses
- Try the exercises there
  - declare a C++ struct to represent a point in the Cartesian coordinate system
  - declare a C++ struct to represent a hexagon
  - declare a C++ struct to represent a circle

Complex record data structures

Complex record data structures

```
const int MAX_NAMES = 100;

struct FullName {
  string name_component[MAX_NAMES];
  int name_count;
};
```

Complex record data structures

Complex record data structures

```
const int SCREEN_HEIGHT = 768, SCREEN_WIDTH = 1024;
struct Screen{
  char screen_array[SCREEN_HEIGHT][SCREEN_WIDTH];
};

...

Screen my_screen;
for (int i = 0; i < SCREEN_HEIGHT; i++){
  my_screen.screen_array[i][0] = '*';
 }
```

Complex record data structures

Complex record data structures

```
struct str1 {
  int a[2];
  int b;
};

void func1(str1 A[ ]){
  A[0].a[0] = 10;
  A[0].a[1] = 20;
  A[0].b = 30;
}

int main( ) {
  str1 A[ ] = {{{1,0},2}, {{3,0},4},{{0,0},9}};
  func1(A);

  std::cout << A[0].b<<"\n";
  std::cout << A[0].a[1]<<"\n";
}
```

Enumerations

- User-defined data type that consists of integral constants

Enumerations

- User-defined data type that consists of integral constants

Enumerations

- User-defined data type that consists of integral constants

```
enum day {
  Friday = 99,//
  Saturday,//
  Sunday = 90,//
  ...,
  Thursday //
};

day d;
d = Thursday;

if (d == Saturday  d == Sunday)
  cout << "Enjoy the weekend!" ;

cout << d+1 ;
```

Variant records

- Multiple component fields can be defined

Variant records

- Multiple component fields can be defined
- At most one field can be in use at one time (fields share the same memory)

Variant records

- Multiple component fields can be defined
- At most one field can be in use at one time (fields share the same memory)

Variant records

- Multiple component fields can be defined
- At most one field can be in use at one time (fields share the same memory)

```
union Coordinates {
  int a, //
  double b, //
  char c //
};

Coordinates x;

x.a = 5;
cout << x.a;          // works, prints 5

x.b = 416.905;      // destroys the value of x.a
x.c = `p';               // destroys the value of x.a and x.b
cout << x.a;          // invalid!
```

Example

Example

```
enum CatalogEntryType {
  BookEntry, //
  DVDEntry //
};

struct BookSpecificInfo {
  unsigned int pages;
};



struct DVDSpecificInfo {
  unsigned int discs;
  unsigned int minutes;
};

union CatalogEntryVariantPart {
  BookSpecificInfo book;
```

Example (cont’d)

Example (cont'd)

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
  CatalogEntryType tag;
  CatalogEntryVariantPart variant;
};
```

Example (cont’d)

Example (cont'd)

```
void printCatalogEntry(const CatalogEntry& c) {
  cout << "Title: " << c.title << endl;
  ...
    cout << "Call Number: " << c.callNumber << endl;
  switch (c.tag) {
  case BookEntry:
    cout << "Pages: " << c.variant.book.pages << endl;
    break;
  case DVDEntry:
    cout << "Discs: " << c.variant.dvd.discs << endl;
    cout << "Minutes: " << c.variant.dvd.minutes << endl;
    break;
  }
}
```

Anonymous declaration of records and variant-records

- Earlier:

Anonymous declaration of records and variant-records

- Earlier:

Anonymous declaration of records and variant-records

- Earlier:

```
union CatalogEntryVariantPart {
  BookSpecificInfo book;
  DVDSpecificInfo dvd;
};
```

- Could have actually declared them in-line:

Anonymous declaration of records and variant-records

- Earlier:

```
union CatalogEntryVariantPart {
  BookSpecificInfo book;
  DVDSpecificInfo dvd;
};
```

- Could have actually declared them in-line:

Anonymous declaration of records and variant-records

- Earlier:

```
union CatalogEntryVariantPart {
  BookSpecificInfo book;
  DVDSpecificInfo dvd;
};
```

- Could have actually declared them in-line:

```
union CatalogEntryVariantPart {
  struct BookSpecificInfo { unsigned int pages; } book;
  struct DVDSpecificInfo { unsigned int discs, minutes; } dvd;
};
```

Anonymous declaration of records and variant-records

- Can also anonymize:

Anonymous declaration of records and variant-records

- Can also anonymize:

Anonymous declaration of records and variant-records

- Can also anonymize:

```
union CatalogEntryVariantPart {
  struct { unsigned int pages; } book;
  struct { unsigned int discs, minutes; } dvd;
};
```

Anonymous declaration of records and variant-records

- In fact, we could have done the same with the union

Anonymous declaration of records and variant-records

- In fact, we could have done the same with the union

Anonymous declaration of records and variant-records

- In fact, we could have done the same with the union

```
struct CatalogEntry {
  string title;
  string author;
  string publisher;
  unsigned int publishingYear;
  string callNumber;
  CatalogEntryType tag;
  union {
    struct { unsigned int pages; } book;
    struct { unsigned int discs, minutes; } dvd;
  } variant;
};
```