Records

CS 115

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

Last updated: January 21, 2025

Structs

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:

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:

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!

- 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

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:

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:

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

Initializing a Record

- As with arrays

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

Copying a Record

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

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)

Passing References

- For efficiency, call by reference is also supported

Passing References

- For efficiency, call by reference is also supported

Passing References

- For efficiency, call by reference is also supported

Passing References

- 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

- Not supported by default

Equality checking

- Not supported by default

Equality checking

- Not supported by default

Equality checking

- Not supported by default

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

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

Equality checking

- Not supported by default

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

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

Equality checking

- Not supported by default

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

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

Equality checking

- Not supported by default

```
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 &c
  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

- Arrays of records

Complex record data structures

- Arrays of records

Complex record data structures

- Arrays of records

Complex record data structures

- Arrays of records

```
CatalogEntry A[3];
CatalogEntry A[] = {{"Peter Pan",
                     "J. M. Barrie",
                     "Scribner",
                     1980,
                     "B2754 1980"},
                    {"C++ Primer",
                     "Stanley B. Lippman",
                     "Addison-Wesley",
                     1998,
                     "QA 76.73 C15 L57 1998"},
                    {"Anatomy of LISP",
                     "John Allen",
                     "McGraw-Hill",
                     1978,
                     "QA 76.73 L23A44"}};
```

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

Arrays inside of records

- Can put arrays as fields of records

Arrays inside of records

- Can put arrays as fields of records

Arrays inside of records

- Can put arrays as fields of records

Arrays inside of records

- Can put arrays as fields of records

```
const int MAX_NAMES = 100;

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

Multi-Dimensional Arrays in Records

Multi-Dimensional Arrays in Records

Multi-Dimensional Arrays in Records

```
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] = '*';
 }
```

Mix and Match

Mix and Match

Mix and Match

Mix and Match

Mix and Match

```
#include <iostream>
using namespace std;
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[3] = {{{1,0},2}, {{3,0},4},{{0,0},9}};
  func1(A);

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

Mix and Match

```cpp
#include <iostream>
using namespace std;
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[3] = {{{1,0},2}, {{3,0},4},{{0,0},9}};
  func1(A);

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

- What will the ouput be?

Enums

Enumerations

Enumerations

Enumerations

Enumerations

Enumerations

```cpp
#include <iostream>
using namespace std;

enum day {
  Sunday = 0,
  Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday
};
```

Enumerations

```
#include <iostream>
using namespace std;

enum day {
  Sunday = 0,
  Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday
};
```

Enumerations

```cpp
#include <iostream>
using namespace std;

enum day {
  Sunday = 0,
  Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday
};
```

Enumerations

```cpp
#include <iostream>
using namespace std;

enum day {
  Sunday = 0,
  Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday
};
```

```cpp
int main() {
  day d;
  d = Thursday;
  d = 1001;

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

  cout << d + 1;
}
```

- User-defined data type that consists of integral constants

Enumerations

```cpp
#include <iostream>
using namespace std;

enum day {
  Sunday = 0,
  Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday
};
```

```cpp
int main() {
  day d;
  d = Thursday;
  d = 1001;

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

  cout << d + 1;
}
```

- User-defined data type that consists of integral constants
- What will the output be?

Unions

Variant records

- Called `union` in C++

Variant records

- Called `union` in C++
- Multiple component fields can be defined

Variant records

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

Example

Example

Example

Example

Example

```
#include <iostream>
using namespace std;

union Coordinates {
  char a;
  double b;
  char c;
};

int main() {
  Coordinates x;
  x.a = 5;
  // works, prints 5
  cout << x.a << endl;

  x.b = 0.0;  // destroys the value of x.a
  x.c = 'p'; // destroys the value of x.a and x.b
  cout << x.a << endl; // invalid!
  cout << x.b << endl; // invalid!
  cout << x.c;         // works, prints p
```

```
^^E
p
5.53354e-322
p
```

Example

```
#include <iostream>
using namespace std;

union Coordinates {
  char a;
  double b;
  char c;
};

int main() {
  Coordinates x;
  x.a = 5;
  // works, prints 5
  cout << x.a << endl;

  x.b = 0.0;  // destroys the value of x.a
  x.c = 'p'; // destroys the value of x.a and x.b
  cout << x.a << endl; // invalid!
  cout << x.b << endl; // invalid!
  cout << x.c;         // works, prints p
```

- The invalid accesses print garbage

```
^^E
p
5.53354e-322
p
```

Library Example

Library Example

Library Example

Library Example

Library Example

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

struct BookSpecificInfo {
  unsigned int pages;
};
```

Library Example

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

struct BookSpecificInfo {
  unsigned int pages;
};
```

Library Example

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

struct BookSpecificInfo {
  unsigned int pages;
};
```

Library Example

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

struct BookSpecificInfo {
  unsigned int pages;
};
```

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

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

Example (cont’d)

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)

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;
  }
}
```

Prof’s Aside

- C++ unions are unsafe

Prof’s Aside

- C++ unions are unsafe
    - Without the tag, there’s no way to know which type a union contains

Prof’s Aside

- C++ unions are unsafe
    - Without the tag, there’s no way to know which type a union contains
    - C++ doesn’t require the tag to be there

Prof’s Aside

- C++ unions are unsafe
  - Without the tag, there’s no way to know which type a union contains
  - C++ doesn’t require the tag to be there
    - You have to make sure it’s there

Prof’s Aside

- C++ unions are unsafe
  - Without the tag, there’s no way to know which type a union contains
  - C++ doesn’t require the tag to be there
    - You have to make sure it’s there
    - You have to make sure the tag actually matches the data

Prof’s Aside

- C++ unions are unsafe
  - Without the tag, there’s no way to know which type a union contains
  - C++ doesn’t require the tag to be there
    - You have to make sure it’s there
    - You have to make sure the tag actually matches the data
- Other languages have safe combinations of tags and unions

Prof's Aside

- C++ unions are unsafe
    - Without the tag, there's no way to know which type a union contains
    - C++ doesn't require the tag to be there
        - You have to make sure it's there
        - You have to make sure the tag actually matches the data
- Other languages have safe combinations of tags and unions
    - `enum` in Rust and Swift

Prof’s Aside

- C++ unions are unsafe
  - Without the tag, there’s no way to know which type a union contains
  - C++ doesn’t require the tag to be there
    - You have to make sure it’s there
    - You have to make sure the tag actually matches the data
- Other languages have safe combinations of tags and unions
  - `enum` in Rust and Swift
  - Sealed Classes in Java/Kotlin

Prof’s Aside

- C++ unions are unsafe
  - Without the tag, there’s no way to know which type a union contains
  - C++ doesn’t require the tag to be there
    - You have to make sure it’s there
    - You have to make sure the tag actually matches the data
- Other languages have safe combinations of tags and unions
  - `enum` in Rust and Swift
  - Sealed Classes in Java/Kotlin
  - Algebraic datatypes in functional languages (CS 350)

Anonymous declaration of records and variant-records

- Earlier:

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:

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:

Anonymous declaration of records and variant-records

- Can also anonymize:

```c
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

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;
};
```