Oregon State University

COLLEGE OF ENGINEERING | School of Electrical Engineering and Computer Science

CS 161

Introduction to CS I

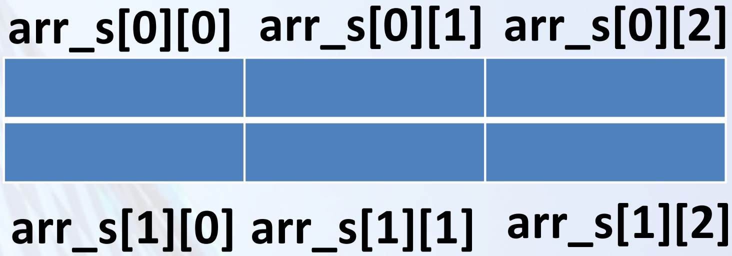
Lecture 20

- Multidimensional arrays
- Structs: create your own data types!

Review: Create 2D arrays

- Stack (static, one block of memory, row-major)

```
char arr_s[2][3];
```



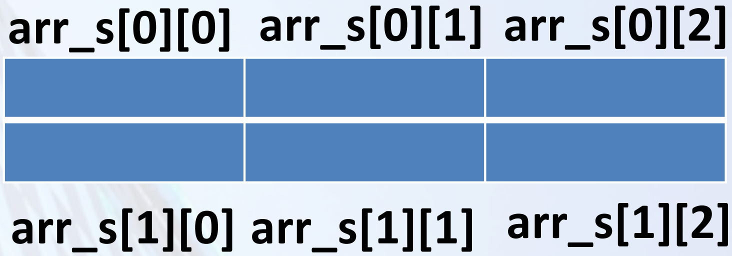
arr_s[0][0] arr_s[0][1] arr_s[0][2]

arr_s[1][0] arr_s[1][1] arr_s[1][2]

Review: Create 2D arrays

- Stack (static, one block of memory, row-major)

```
char arr_s[2][3];
```

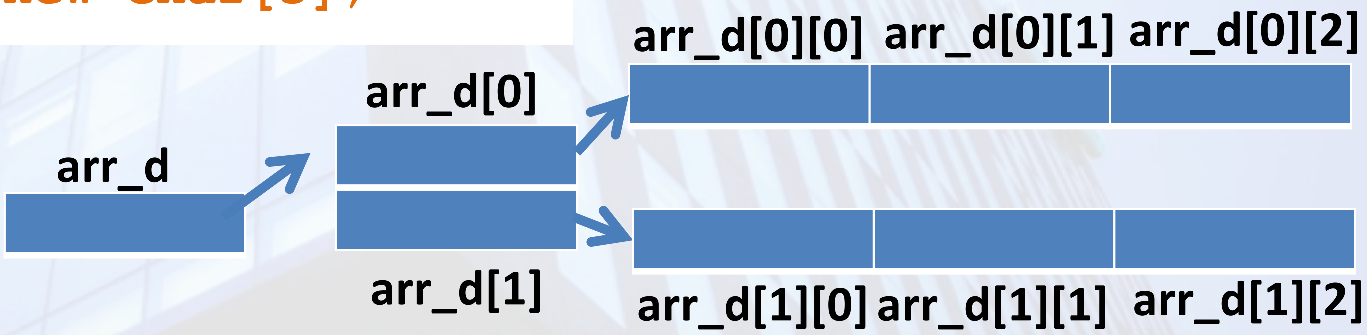


arr_s[0][0] arr_s[0][1] arr_s[0][2]

arr_s[1][0] arr_s[1][1] arr_s[1][2]

- Heap (dynamic, pointers to pointers)

```
char** arr_d = new char*[2];
for (int i=0; i<2; i++)
  arr_d[i] = new char[3];
```



arr_d

arr_d[0]

arr_d[1]

arr_d[0][0] arr_d[0][1] arr_d[0][2]

arr_d[1][0] arr_d[1][1] arr_d[1][2]

Passing static 2D arrays to functions

- Static: must include the size of both dimensions, or at least the final dimension
  - So that we know where each row starts

See lec20-pass-2D-arrays.cpp

```
int main() {
  int array[3][3];

  pass_2Darray_1(array);
  pass_2Darray_2(array);

  return 0;
}
```

```
void pass_2Darray_1(int a[3][3]) {
  cout << a[0][0] << endl;
}
/* OR */
void pass_2Darray_2(int a[][3]) {
  cout << a[0][0] << endl;
}
```

Passing dynamic 2D arrays to functions

- Dynamic: no sizes need to be specified, because the row pointers indicate where each row starts

See lec20-pass-2D-arrays.cpp

```
int main() {
  int** array;
  /* allocate array */
  pass_2Darray_3(array);
  pass_2Darray_4(array);
  /* free array */
  return 0;
}
```

```
void pass_2Darray_3(int* a[]) {
  cout << a[0][0] << endl;
}
/* OR */
void pass_2Darray_4(int** a) {
  cout << a[0][0] << endl;
}
```

Useful: include array dimensions

- Just as with 1D arrays, if you want to iterate over items in an array, pass the sizes as arguments

```
1. void pass_static_2Darray(int a[3][2], int rows, int cols);
2. void pass_static_2Darray(___________, int rows, int cols);

3. void pass_dyn_2Darray(________, int rows, int cols);
4. void pass_dyn_2Darray(int** a,  int rows, int cols);
```

Useful: include array dimensions

- Just as with 1D arrays, if you want to iterate over items in an array, pass the sizes as arguments

```
1. void pass_static_2Darray(int a[3][2], int rows, int cols);
2. void pass_static_2Darray(int a[][2],  int rows, int cols);

3. void pass_dyn_2Darray(int* a[], int rows, int cols);
4. void pass_dyn_2Darray(int** a,  int rows, int cols);
```

Create and return a 2D char array in a function?

```
______ create_2D_array(_______________);
```

1. Heap or stack?
2. What return type?
3. What input arguments?

Create and return a 2D char array in a function?

```
______ create_2D_array(_______________);
```

1. Heap or stack?
   - Must be **heap** – stack is freed when function returns
2. What return type?
   - `char**`
3. What input arguments?
   - `int n_rows, int n_cols`

Create and return a 2D char array in a function

Let's do it!

```
char** create_2D_array(int r, int c) {
   char** ttt = new char*[r]; /* row pointers */
   for (int i=0; i<r; i++)
      ttt[i] = new char[c];   /* row arrays */
   return ttt;
}
```

Oregon State University
College of Engineering

Your favorite collectible item

- Anime body pillows
- Military coins
- Books
- Porcelain dolls
- Money
- …

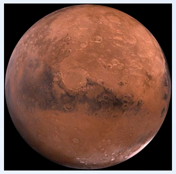
Let's collect planets

- We want to store, for each planet:
  - Name
  - Radius (in km)
  - Number of moons
  - Color

```
string p1_name;
string p2_name;
string p3_name;
…
```

```
string name[9];
float radius[9];
int n_moons[9];
…
```

Mars



Jupiter

(Not to scale)

Define your own data structure ("struct")

- Package info into one data structure for each item

See lec20-structs.cpp

Members

```
struct planet {
    string name;
    float radius;
    unsigned short n_moons;
    string color;
};
```

```
planet mars;
mars.name = "Mars";
mars.radius = 3389.5;
mars.n_moons = 2;
mars.color = "red";
```

- Much more readable and easier to manipulate

Your turn: Define your structure

- What item do you like to collect?
- What attributes do you want to store?
  - Think of at least 3
  - Choose appropriate data types for each member

```
struct planet {
  string name;
  float radius;
  unsigned short n_moons;
  string color;
};
```

Functions on structs

See lec20-structs.cpp

```
/* Return the name of the largest planet */
string largest_planet(planet p1, planet p2) {
  if (p1.radius >= p2.radius)
    return p1.name;
  else /* p2.radius > p1.radius */
    return p2.name;
}
```

- Access members with the . operator (e.g., p1.radius)
- Your turn: Think of a function you would like for your struct

Oregon State University
College of Engineering

Functions on structs

See lec20-structs.cpp

- The previous function passed structs by value (made a copy)
- As structs get larger, it is better to **pass by reference**
- If you want to modify values, you must pass an address (**pass by reference**, or **pass a pointer**)

Functions on structs: pass by reference

See lec20-structs.cpp

-

```
/* We discovered a new moon for this planet! */
void add_moon_r(planet& p) {
  p.n_moons++;
}
```

```
int main() {
  planet jupiter;
  jupiter.n_moons = 79;
  add_moon_r(jupiter);
  cout << jupiter.n_moons << endl;
  return 0;
}
```

Functions on structs: pass a pointer

See lec20-structs.cpp

```
/* We discovered a new moon for this planet! */
void add_moon_p(planet* p) {
  (*p).n_moons++;
}
```

```
int main() {
  planet jupiter;
  jupiter.n_moons = 79;
  add_moon_p(&jupiter);
  cout << jupiter.n_moons << endl;
  return 0;
}
```

The arrow operator (->)

See lec20-structs.cpp

```
/* We discovered a new moon for this planet! */
void add_moon_p(planet* p) {
  p->n_moons++;
}
```

```
int main() {
  planet jupiter;
  jupiter.n_moons = 79;
  add_moon_p(&jupiter);
  cout << jupiter.n_moons << endl;
  return 0;
}
```

Your turn: Create an array of 9 structs (planets)

```
1. _______ my_planets[____]; /* stack */


2. _______ my_planets = new _______[_____]; /* heap */
```

Your turn: Create an array of 9 structs (planets)

```
1. planet  my_planets[9]; /* stack */


2. planet* my_planets = new planet[9]; /* heap */
```

Delete the array of structs off the heap

```
1. planet my_planets[9]; /* stack */


2. planet* my_planets = new planet[9]; /* heap */
3. delete [] my_planets;
```

Oregon State University
College of Engineering

Struct initializer

- One member at a time:

```
planet mars;
mars.name = "Mars";
mars.radius = 3389.5;
mars.n_moons = 2;
mars.color = "red";
```

- All members at once (in order):

```
planet mars = { "Mars", 3389.5, 2, "red" };
```

What vocabulary did we learn today?

- Struct
- Member
- . ("dot") operator: access a member
- -> ("arrow") operator: dereference pointer and access member

What ideas and skills did we learn today?

- How to pass 2D arrays to functions
- How to define your own data structures (structs)
  - e.g., planet
- How to access and update the members of a struct
  - e.g., p.radius
- How to pass structs to a function
- How to create an array of structs on the stack or heap
- How to initialize structs

Oregon State University
College of Engineering

Week 7 nearly done!

- □ Attend lab (laptop required)
- □ Read http://www.cplusplus.com/doc/tutorial/structures/
- □ **Assignment 4** (due **Sunday, Feb. 23**)

See you Monday!