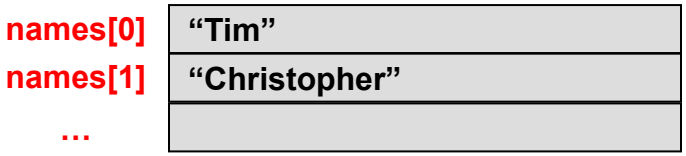
CS103 Unit 2d – Shallow and Deep Copy, Allocating 2D arrays

SHALLOW VS. DEEP COPY

Dealing with Text Strings

- What's the best way to store many text strings that we will not know until run time and that could be short or long?
- Statically:
  - Bad! Either wastes space or some user will enter a string just a little too long

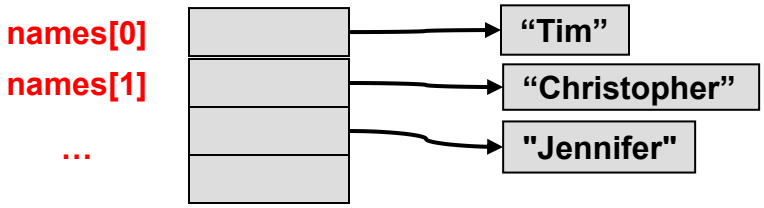
```
#include <iostream>
using namespace std;

int main()
{
  // store 10 user names of up to
  // 40 chars
  char names[10][40];



}
```



Jagged 2D-Arrays

- What we want is just enough storage for each text string
- This is known as a *jagged* 2D-array since each 1D array is a different length
- To achieve this, we will need an array of pointers
  - Each pointer will point to an array of different length

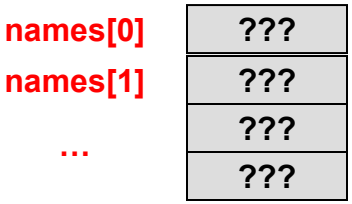
```
#include <iostream>
using namespace std;

int main()
{
  // store 10 user names
  char *names[10];

  for(int i=0; i < 10; i++){
    // read in and store each name
    //  But HOW ??
  }
}
```



More Dealing with Text Strings

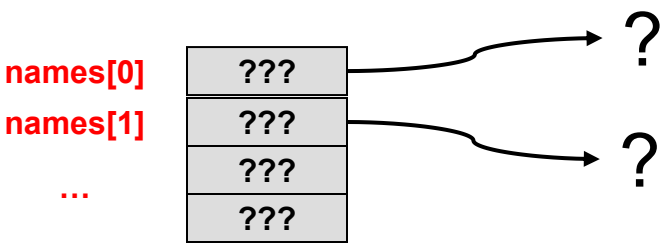
- Will this code work to store 10 names?

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


int main()
{
  // store 10 user names
  //   names type is still char**
  char* names[10];

  for(int i=0; i < 10; i++){
    cin >> names[i];
  }

  // Do stuff with names




  return 0;
}
```



More Dealing with Text Strings

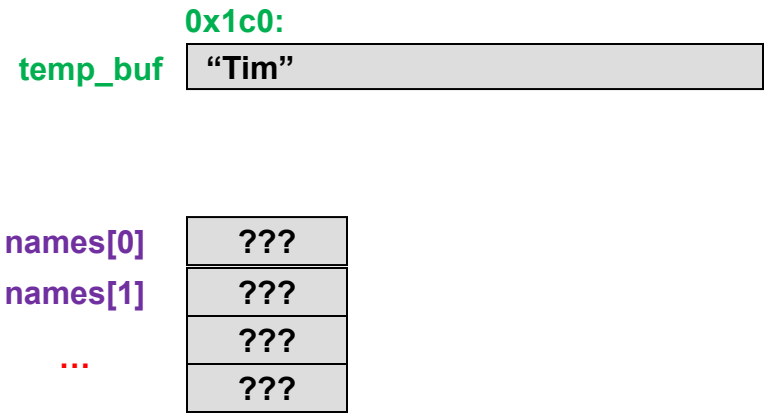
- Will this code work to store 10 names?
- No!! You must allocate storage (i.e. an actual array) before you have pointers pointing to things (e.g. you can't just move into the White House, OR… just because I make up a URL like: http://docs.google.com/uR45y781 doesn't mean there's a document there)
  - We need a real array to receive input!




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


int main()
{
  // store 10 user names
  //   names type is still char**
  char* names[10];

  for(int i=0; i < 10; i++){
    cin >> names[i];
  }

  // Do stuff with names




  return 0;
}
```

More Dealing with Text Strings

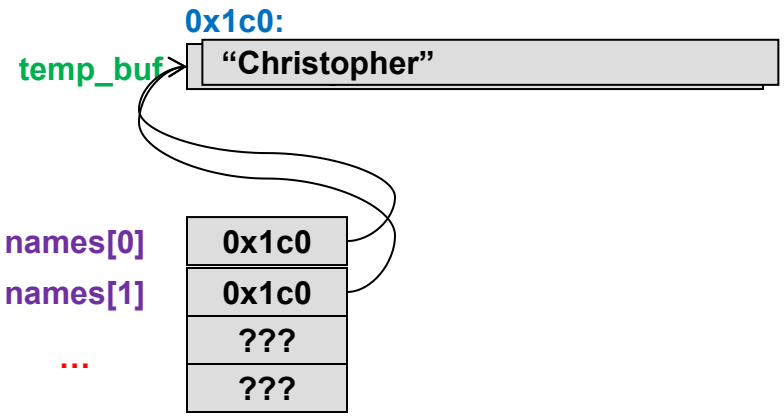
- Let's allocate 1 real character array to serve as our scratchpad for receiving input names.
- We'll make it as BIG as ANY name we might want to handle (say 40 characters)




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

int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];

  // One "scratchpad" array to read in a name
  char temp_buf[40];

  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = temp_buf;
  }
  // Do stuff with names

  return 0;
}
```

More Dealing with Text Strings

- Now we can cin one name at a time
- But will this code work to store 10 ***unique*** names?
- This leads to many copies of the pointer but only one array (aka SHALLOW copies)
- **Shallow copy** = copies of a pointer but not copies of the data

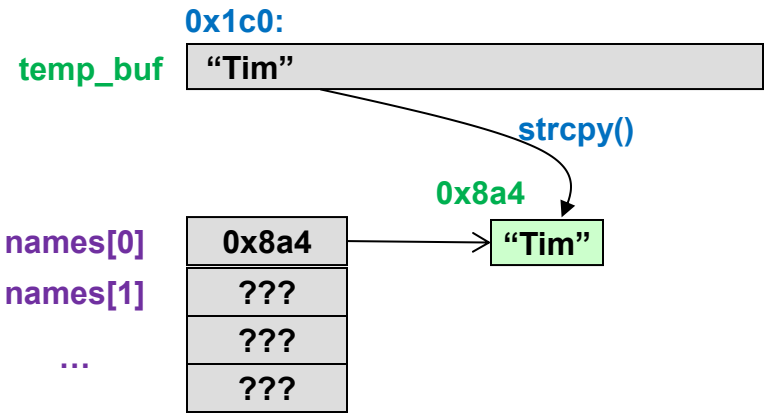


Assigning an array name just assigns a pointer and does NOT make a copy of the array.

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

int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];

  // One "scratchpad" array to read in a name
  char temp_buf[40];

  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = temp_buf;
  }
  // Do stuff with names

  return 0;
}
```

More Dealing with Text Strings

- What's the best way to store text strings for data that we will not know until run time and that could be short or long?
- Dynamically:
  - Better memory usage
  - Requires a bit more coding

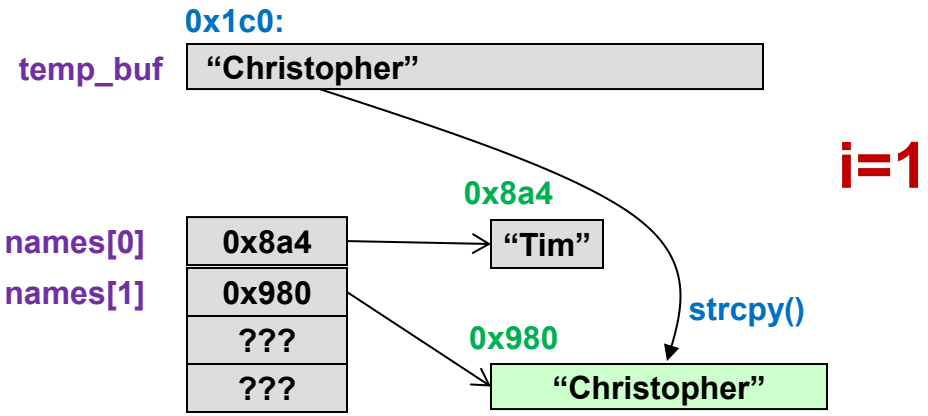


i=0

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

int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];
  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;

    int len = strlen(temp_buf);  // find length
    // allocate just enough
    names[i] = new char[len + 1];

    // copy from scratchpad to new array
    strcpy(names[i], temp_buf);
  }
  // Do stuff with names

  for(int i=0; i < 10; i++){
    delete [] names[i];
  }
  return 0;
}
```

More Dealing with Text Strings

- Here, we perform a "**DEEP COPY**" where we allocate a whole new array and make a copy of the data, not just the pointer
- A **DEEP COPY** takes more work and usually requires:
  - 1.) Allocating a new array
  - 2.) Copying data from the old to the new



i=1

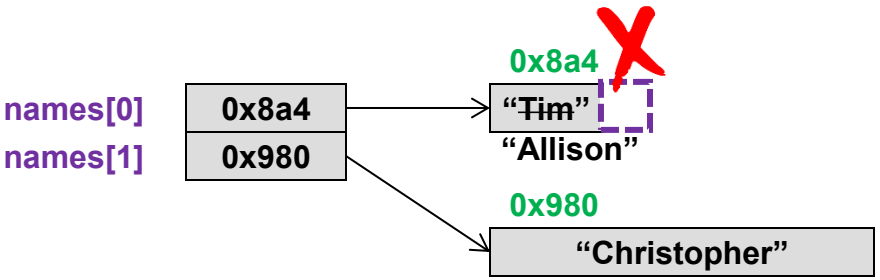
```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];
  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;

    int len = strlen(temp_buf);    // find length
    names[i] = new char[len + 1]; // allocate
    strcpy(names[i], temp_buf);    // copy
  }
  // Do stuff with names

  for(int i=0; i < 10; i++){
    delete [] names[i];
  }
  return 0;
}
```

DEEP COPY

Make Sure There is Room

- If we want to change a name, what do we have to do?
- Can we just cin to the array?

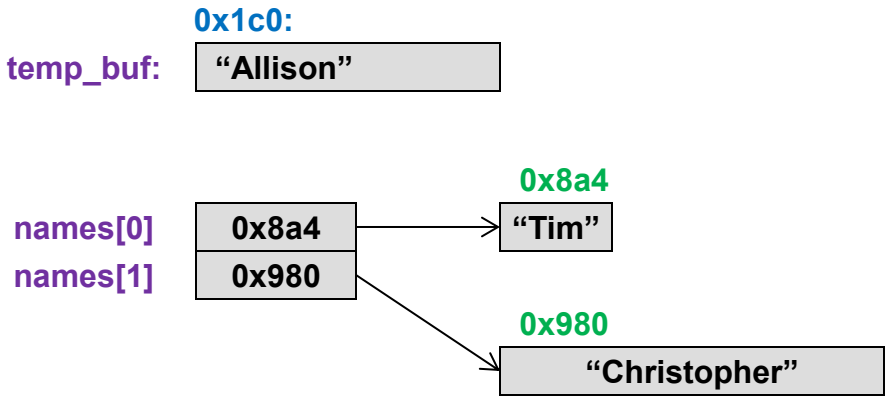
- **No!** Because what if the new name is **longer** than the array allocated for the old name…we'd write off the end of the array and corrupt memory




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


int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];

  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = new char[strlen(temp_buf)+1];
    strcpy(names[i], temp_buf);

  }

  // What if I want to change names[0]

  cin >> names[0];  // user enters "Allison"


  for(int i=0; i < 10; i++){
    delete [] names[i];
  }
  return 0;
}
```

Shallow Copy vs. Deep Copy

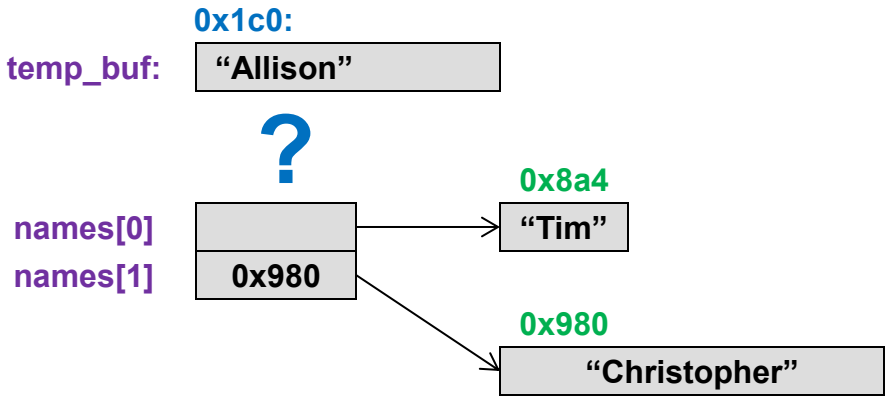
- Ok, let's use our scratchpad again.
- Can we just use the assignment operator, '='?




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


int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];

  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = new char[strlen(temp_buf)+1];
    strcpy(names[i], temp_buf);

  }

  // What if I want to change names[0] & [1]

  cin >> temp_buf;  // user enters "Allison"
  names[0] = temp_buf;
  cin >> temp_buf;  // user enters "Jennifer"
  names[1] = temp_buf;

  for(int i=0; i < 10; i++){
    delete [] names[i];
  }
  return 0;
}
```

Assignment = Shallow Copy

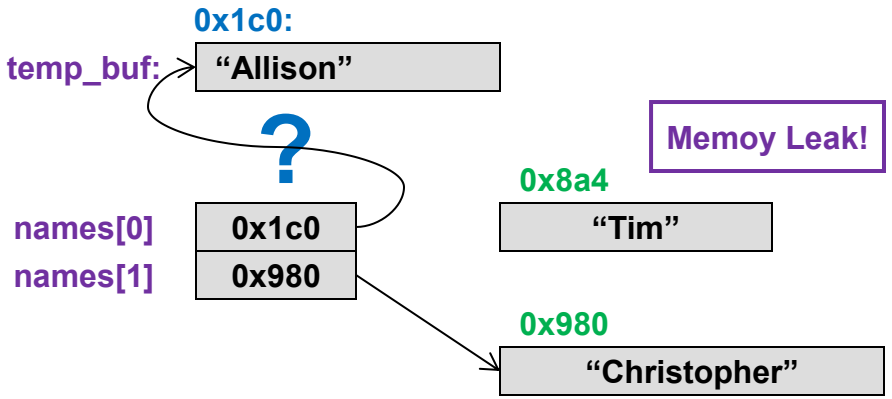
- Ok, let's use our scratchpad again.
- Can we just use the assignment operator, '='?
- What happens when we do `names[0] = temp_buf;`




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


int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];

  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = new char[strlen(temp_buf)+1];
    strcpy(names[i], temp_buf);

  }

  // What if I want to change names[0] & [1]

  cin >> temp_buf;  // user enters "Allison"
  names[0] = temp_buf;
  cin >> temp_buf;  // user enters "Jennifer"
  names[1] = temp_buf;

  for(int i=0; i < 10; i++){
    delete [] names[i];
  }
  return 0;
}
```

Shallow Copy

- A SHALLOW copy is made
  - **Shallow copy** = copy of ***pointers*** to data rather than copy of *actual data*
- Pointers are references… assigning a pointer doesn't make a copy of what its pointing at but only makes a copy of the pointer (a.k.a. "shallow copy")



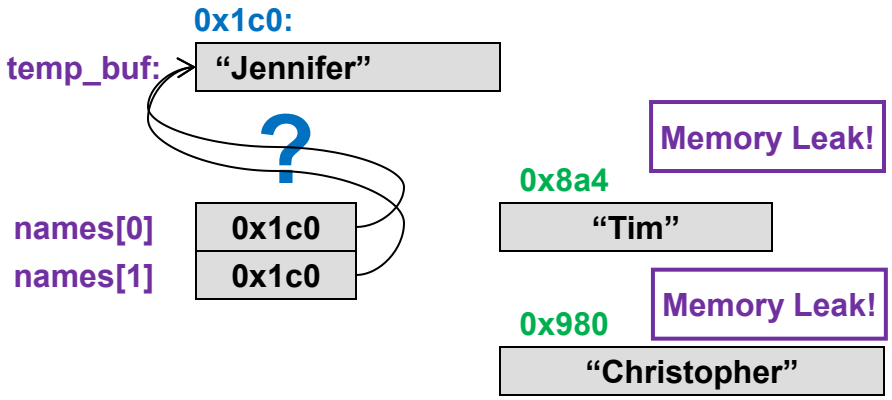
temp_buf evaluates to address of array.
So names[0] = temp_buf simply copies address of array into names[0]…It does not make a copy of the array

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


int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];

  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = new char[strlen(temp_buf)+1];
    strcpy(names[i], temp_buf);

  }

  // What if I want to change names[0] to [1]

  cin >> temp_buf;  // user enters "Allison"
  names[0] = temp_buf;
  cin >> temp_buf;  // user enters "Jennifer"
  names[1] = temp_buf;

  for(int i=0; i < 10; i++){
    delete [] names[i];
  }
  return 0;
}
```

SHALLOW COPY

Take Care To Avoid Leaks

- Clean up any old data your pointer references before changing it

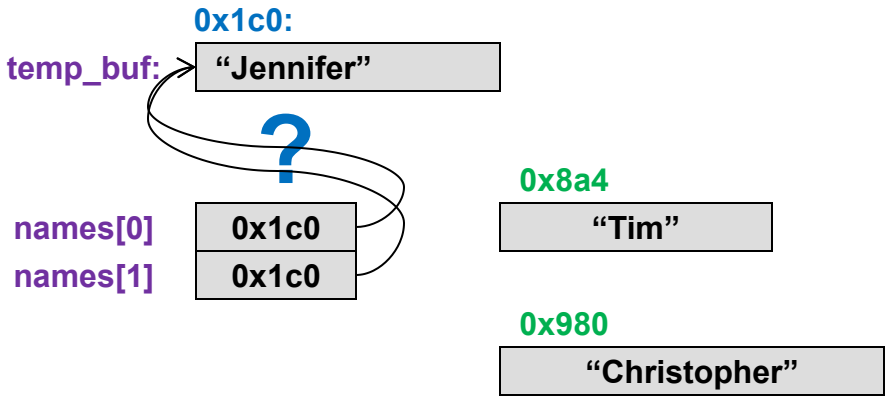


Same problem with assignment of temp_buf to names[1]. Now we have two things pointing at one array and we have lost track of memory allocated for Timothy and Christopher…memory leak!

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


int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];

  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = new char[strlen(temp_buf)+1];
    strcpy(names[i], temp_buf);

  }

  // What if I want to change names[0] & [1]

  cin >> temp_buf;  // user enters "Allison"
  names[0] = temp_buf;
  cin >> temp_buf;  // user enters "Jennifer"
  names[1] = temp_buf;

  for(int i=0; i < 10; i++){
    delete [] names[i];
  }
  return 0;
}
```

Only delete Dynamically Allocated Memory

- When we get to the point of deleting our dynamic memory, we will no longer be pointing at the dynamically allocated arrays!
- Deleting memory that was not allocated with new may crash the program.

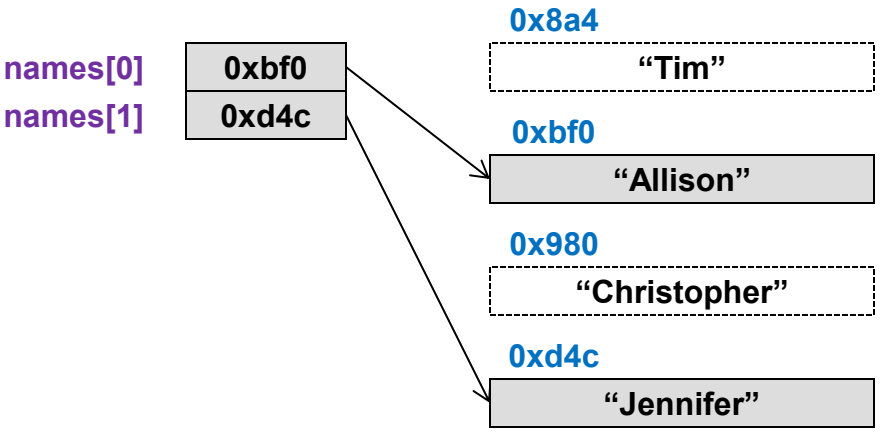


When we try to "delete" or free the memory pointed to by names[i], it will now try to return memory it didn't even allocate (i.e. temp_buf) and cause the program to crash!

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


int main()
{
  // store 10 user names
  //   names type is still char **
  char* names[10];

  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = new char[strlen(temp_buf)+1];
    strcpy(names[i], temp_buf);

  }

  // What if I want to change names[0] & [1]

  cin >> temp_buf;  // user enters "Allison"
  names[0] = temp_buf;
  cin >> temp_buf;  // user enters "Jennifer"
  names[1] = temp_buf;

  for(int i=0; i < 10; i++){
    delete [] names[i];
  }
  return 0;
}
```

Deep Copies

- If we want to change the name, what do we have to do?
- Must allocate new storage and copy original data into new memory (a.k.a. deep copy)
  - **Deep copy** = allocate new memory AND then copy the original data (1 by 1) to the new memory




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


int main()
{
  // store 10 user names
  //   names type is still char **
  char *names[10];

  char temp_buf[40];
  for(int i=0; i < 10; i++){
    cin >> temp_buf;
    names[i] = new char[strlen(temp_buf)+1];
    strcpy(names[i], temp_buf);

  }

  // What if I want to change names[0] & [1]

  cin >> temp_buf;  // user enters "Allison"
  delete [] names[0];
  names[0] = new char[strlen(temp_buf)+1];
  strcpy(names[0], temp_buf);
  cin >> temp_buf;  // user enters "Jennifer"
  delete [] names[1];
  names[1] = new char[strlen(temp_buf)+1];
  strcpy(names[1], temp_buf);
  ...
```

DEEP COPY

2D ARRAY ALLOCATION

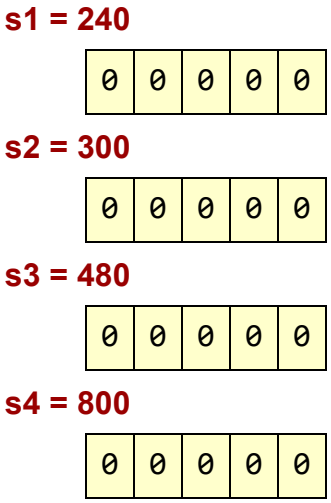
Arrays of pointers

- We often want to have several arrays to store data
  - Store each student's grades
- Those arrays may be related (i.e. scores of students in a class)

```
int s1[5] = {0,0,0,0,0};
int s2[5] = {0,0,0,0,0};
int s3[5] = {0,0,0,0,0};
int s4[5] = {0,0,0,0,0};

int main(int argc, char *argv[])
{
  int avg = 0;

  for(i=0;i < 5;i++) { avg += stu1scores[i]; }
  for(i=0;i < 5;i++) { avg += stu2scores[i]; }
  for(i=0;i < 5;i++) { avg += stu3scores[i]; }
  for(i=0;i < 5;i++) { avg += stu4scores[i]; }
  avg /= 4*5;

}
```

- Yes, a 2D array would probably have been better in this case (int scores[4][5]), but suppose we already had a bunch of these 1D arrays. How can I access them easily?

Painful

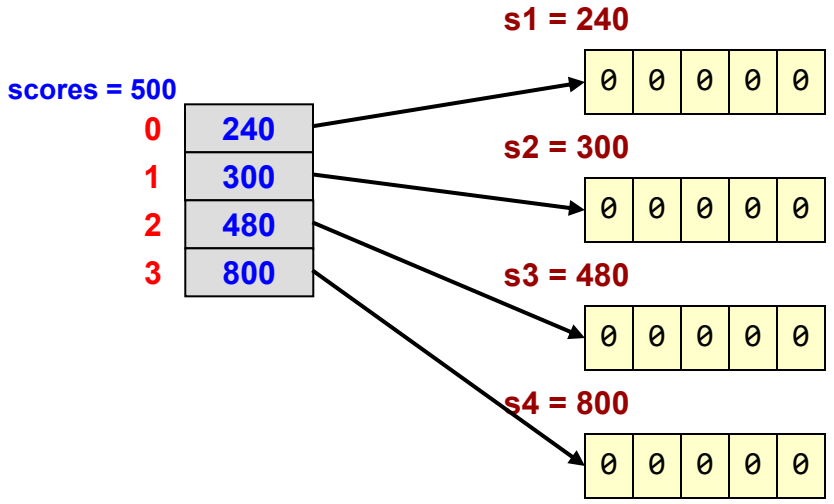


Arrays of pointers

- We can make an array of pointers where each entry points at an array of integers (of a student's scores)
- This effectively forms a 2D array.
- Fact: An array of pointers is how a 2D array is often represented.

```
int s1[5] = {0,0,0,0,0};
int s2[5] = {0,0,0,0,0};
int s3[5] = {0,0,0,0,0};
int s4[5] = {0,0,0,0,0};
int main(int argc, char *argv[])
{
  int avg = 0;
  int *scores[4] = {s1, s2, s3, s4};
  for(int i=0; i < 4; i++){
    for(int k=0; k < 5; k++){
      avg += scores[i][k];
    }
  }
  avg /= 4*5;
}
```

Better



Arrays of pointers

- What if the number of students and the number of scores for each student are NOT KNOWN until the program runs.
- Are the options shown to the right legal in C++?
- I need dynamic allocation!

```
int main(int argc, char *argv[])
{
  int numStu, numScores;
  cin >> numStu >> numScores;

  // Is this okay to do in C++?
  int scores[numStu][numScores];

  // Or is this okay to do in C++?
  int s1[numScores] = {0,0,0,0,0};
  int s2[numScores] = {0,0,0,0,0};
  // And I don't even know how many of
  // these to allocate
  ...
}
```

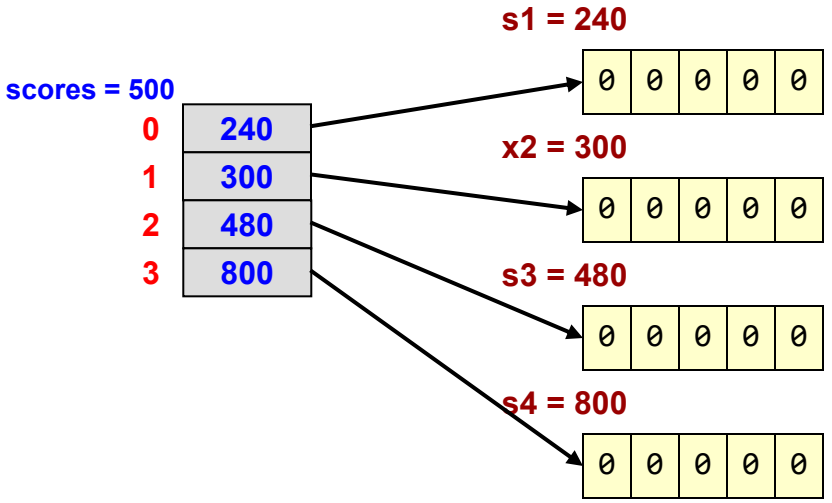
Allocating a 2D Array Structure

- Fact: Sadly, C/C++ does **NOT** easily support dynamic allocation of 2D-, 3D-, etc. arrays of variable size.

- BUT, it does support dynamic allocation of:
  - 1D arrays AND
  - 1D arrays of pointers
- By putting those two together we can get a variable-size 2D Array

```
int main(int argc, char *argv[])
{
  int numStu, numScores;
  cin >> numStu >> numScores;
  // 2D array dynamic allocation is not supported
  int** scores = new int[numStu][numScores]; //BAD

  // Allocate a 1D array of (row) pointers
  int** scores = new int*[numStu];
  for(int i=0; i < numStu; i++){
     scores[i] = new int[numScores];
  }
  ...
}
```

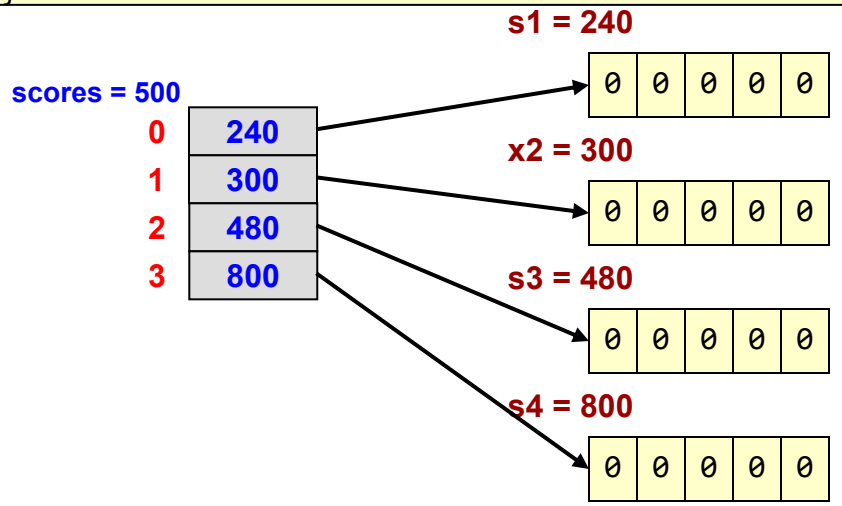
Dynamic Allocation of variable-sized 2D arrays



Deallocating 2D Array Structure

- Be sure to deallocate all the memory
- The structure used to deallocate memory should be a MIRROR of the code used to allocate with new

```
int main(int argc, char *argv[])
{
  int numStu, numScores;
  cin >> numStu >> numScores;

  // Allocate a 2D structure (array of pointers)
  // then each row separately
  int** scores = new int*[numStu];
  for(int i=0; i < numStu; i++){
     scores[i] = new int[numScores];
  }
  // Deallocate 2D array structure
  for(int i=0; i < numStu; i++){
     delete [] scores[i];
  }
  delete [] scores;
}
```



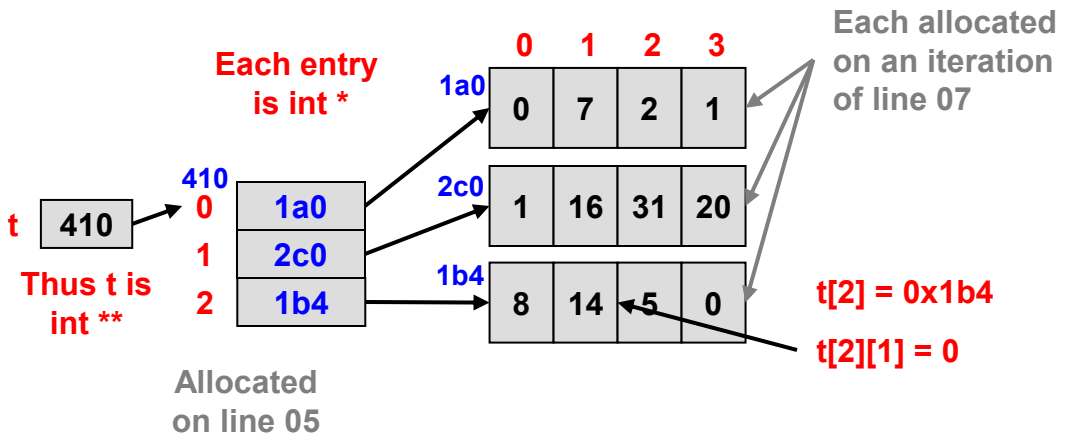
2D Allocation: Repeated Example

- Dynamically allocating 2D arrays in C/C++ doesn't really work
  - Instead conceive of 2D array as an "array of arrays" which boils down to a pointer to a pointer
- To allocate an N row x M column array dynamically we :
  - Allocate an array of N pointers
  - Allocate each row of M columns separately and have each of the N pointers point to a different row

0	7	2	1
1	16	31	20
8	14	5	0

```
01  int main()
02  {
03    int n, m;
04    cin >> n >> m;
05    int *t = new int[n][m];
06       // DOESN'T WORK!!
07    ...
08  }
```

```
01  int main()
02  {
03    int n, m;
04    cin >> n >> m;
05    int **t = new int*[n];
06    for(int i=0; i < n; i++){
07      t[i] = new int[m];
08      for(int j = 0; j < m; j++){
09        t[i][j] = 0;
10      }
11    }
12    // Access t[r][c] as desired
13
14    // deallocate arrays
15     for(int i=0; i < n; i++){
16      delete t[i];
17    }
18    delete [] t;
19  }
```



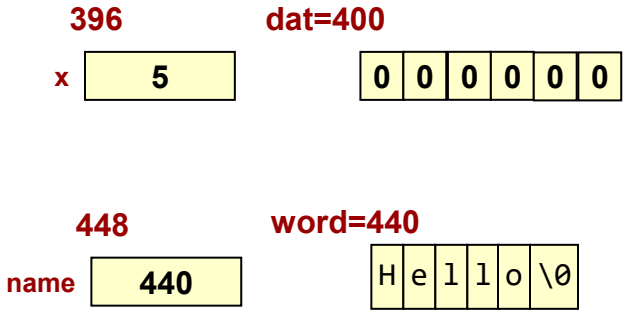
Exercise

- In-class-exercises
  - nxmboard

IF TIME

cin/cout & char*s

- `cin/cout` determine everything they do based on the **type** of data passed
- `cin/cout` have a unique relationship with `char*s`
- When `cout` is given a variable or expression of any type, it will print the value stored in that exact variable
  - Exception: When `cout` is given a `char*` it will assume it is pointing at a C-string, go to that address, and loop through each character, printing them out
- When `cin` is given a variable it will store the input data in that exact variable
  - Exception: When `cin` is given a `char*` it will assume it is pointing at a C-string, go to that address, and place the typed characters in that memory

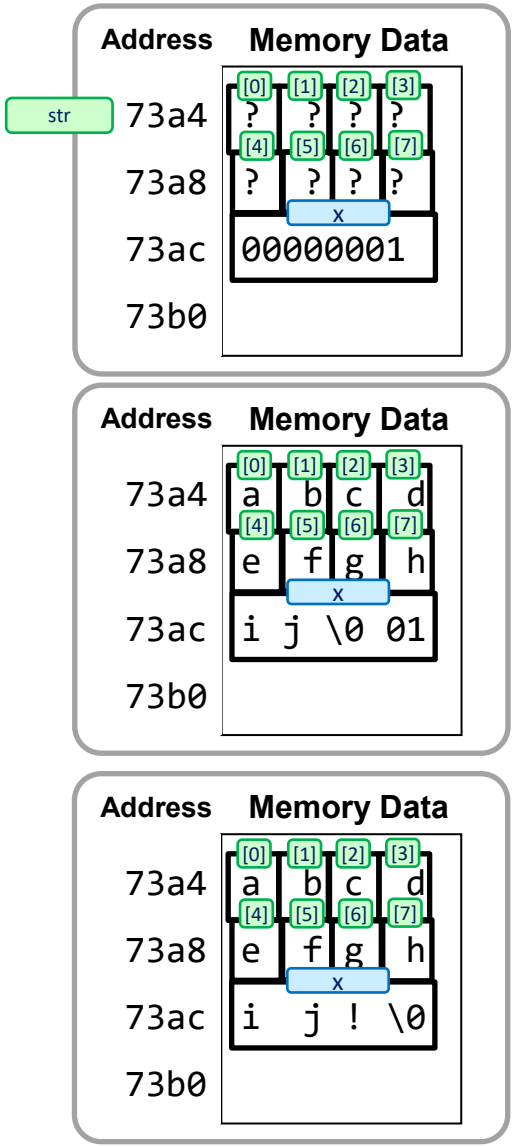



```
#include <iostream>
using namespace std;
int main()
{
  int x = 5, dat[10] = {0}; // dat is an int*
  char word[10] = "Hello";
  char *name = word;

  cout << x << endl;        // 5
  cout << dat << endl;      // 400
  cout << word << endl;     // Hello
  cout << name << endl;     // Hello
  cout << name[0] << endl; // H
  cout << (void*) name << endl;  // 440

  cin >> dat;     // Doesn't work, use a loop
  cin >> name;    // Store many chars
                  // starting at 440
  return 0;
}
```

C-String Library Vulnerabilities

- What could go wrong with these library functions?
  - Consider the code below.

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

int main()
{
  char str1[8];
  int x = 1;
  strcpy(str1, "abcdefghij");

  strcat(str1, "!");

  cout << str3 << endl;

  return 0;
}
```



Safe C-String Library

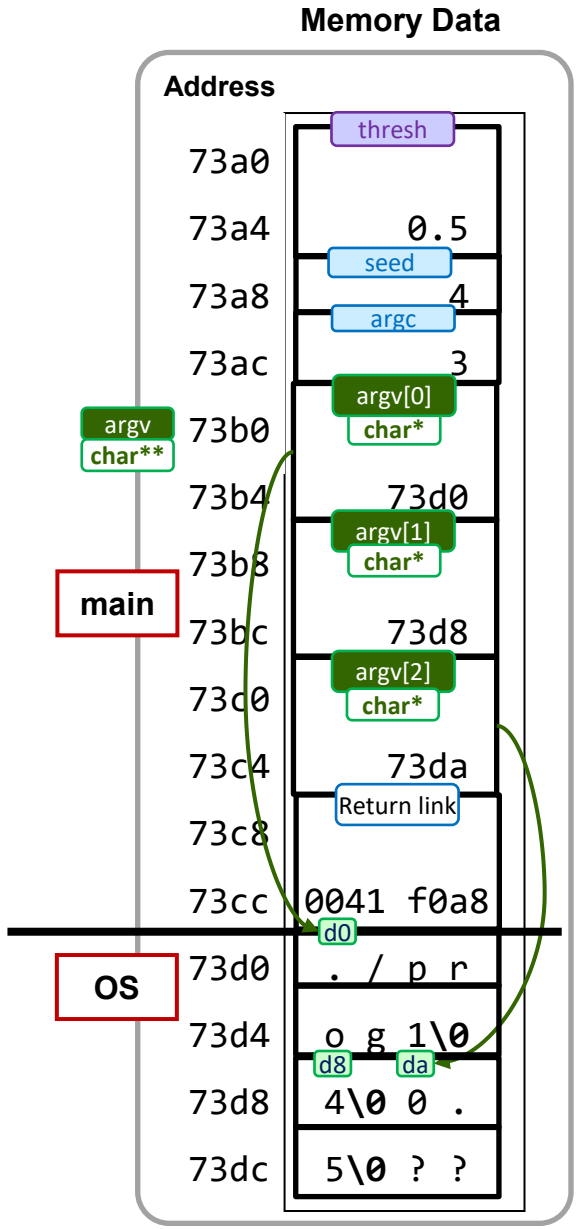
- The <cstring> library was updated in subsequent versions of C++ to provide safer alternatives to avoid array buffer overflows with many functions now having a counterpart with n in the name representing a maximum length to read or write.
  - `int strlen(const char *dest)`
  - `int strncmp(const char *str1, const char *str2, size_t num);`
    - Return 0 if equal, >0 if first non-equal char in str1 is alphanumerically larger, <0 otherwise
    - Compares a maximum of n characters (which should match the length of the shortest input)
  - `char *strncpy(char *dest, const char *src, size_t num);`
    - Maximum of num characters copied
  - `char *strncat(char *dest, const char *src, size_t num);`
    - Maximum of num characters concatenated plus a NULL
- See the documentation (https://cplusplus.com/reference/cstring/)

A Stack View

- Here is a memory/stack view of the command line arguments, and the argc, argv arguments passed to main()

```
$ ./prog1 4 0.5
```

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

int main(int argc, char *argv[])
{
  if(argc < 3) {
    cout << "Not enough inputs" << endl;
    return 1;
  }
  int seed = strtol(argv[1]);
  double threshold = strtod(argv[2]);
  // use seed and threshold
  // ...
}
```



BACKUP

Why Pointers To Arrays

- 4 friends got sequential hotel rooms.
- Alice hates that her room number is 413 and would like to be "next" to her friend Gina.
- She asks Tim to swap rooms. Tim doesn't care but DOESN'T want to move all his stuff.
- Kyle has an idea to "satisfy" both. Can you guess his approach?