Dynamic memory, dynamic arrays, memory leaks, classes with pointer members, constructors and destructors

Comp Sci 1575 Data Structures

OOPs...

From back when Object wriented Programming and data hiding were new:

"In C++ it's harder to shoot yourself in the foot, but when you do, you blow off your whole leg." Bjarne Stroustrup (the original author of C++).

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Dynamic variables

- **Operator:** new
- **General syntax:** pointer = new type
- **Example:**
  int *p1;
  p1 = new int;
  *p1 = 42;
- **Example 2:**
  double *p2 = new double;
  *p2 = 42

Name of variable	Storage address	Value
	0x7ffcb158c140	
	0x7ffcb158c144	42
	0x7ffcb158c148	
p1	0x7ffcb158c14c	0x7ffcb158c144
	0x7ffcb158c150	

Address and value pointed to by p1 have no named alias

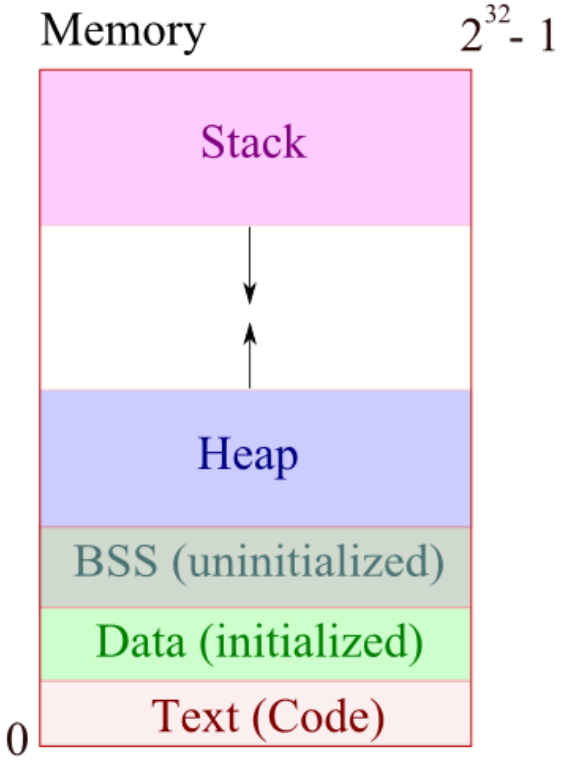
Outline

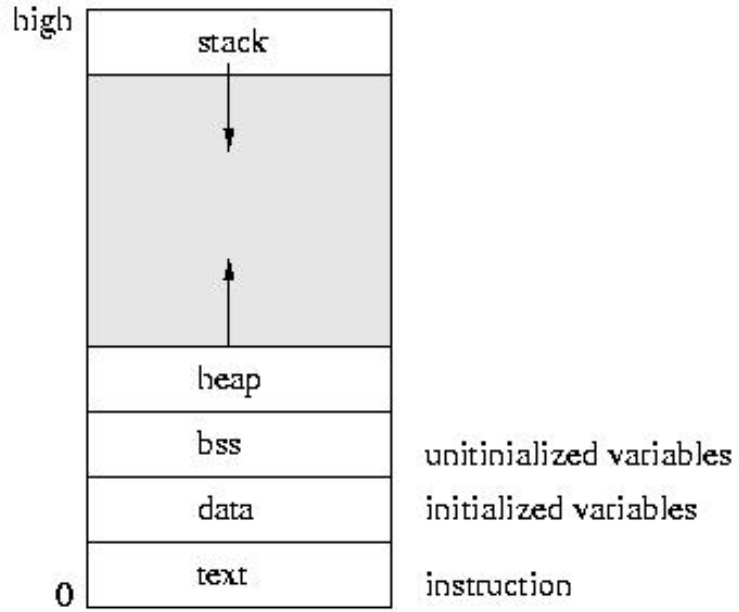
Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Dynamic memory

- **Stack:** Declared variables will reside in stack memory
- **Heap:** Can be used to allocate memory dynamically when program runs
- When finished running, stack memory is automatically de-allocated, but the data referenced by new pointers on the heap is not
- Operator "delete" should be used to de-allocate the data pointed to
- **delete p;** de-allocateds item being pointed to on the heap
- What is de-allocation? It is not actually deleting, but marking the memory as available for use

Memory organization





Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Delete example


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

int main(){
   double *pvalue  = NULL; // Why?
   pvalue  = new double; // Request memory
   *pvalue = 29494.99; // ??

   cout << *pvalue << endl; // ??

   delete pvalue; // free up the memory.

   return 0;
}
```

- What happens if your program repeats without delete?
- What is it called when you forget delete?
  Microsoft Windows...

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Dynamic array operator: new [ ]

Dynamic array creation: pointer = new type[numElements]

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

int main(){
  int userDefinedSize;
  int *pUserSizedArray;

  cout << ``How large of an array?" << endl;
  cin >> userDefinedSize;

  pUserSizedArray = new int[userDefinedSize];
  pUserSizedArray[1] = 23;
  cout << *(pUserSizedArray+1) << endl; // 23
  return 0;
}
```

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Deleting array values


```
int *pArray;
pArray = new int[500];

// assign and manipulate here...

delete pArray;
```

What will happen if you execute this many times?

Dynamic array operator: delete [ ]

To avoid the memory leak caused by the previous slide's mistake, use:

```
int *pArray = NULL;
pArray = new int[500];

// assign and manipulate here...

delete[ ] pArray;
```

Arrays created with new require: "delete[ ] pArray;" or it will only delete the first element, leaving the rest as garbage

Kahoot

Outline

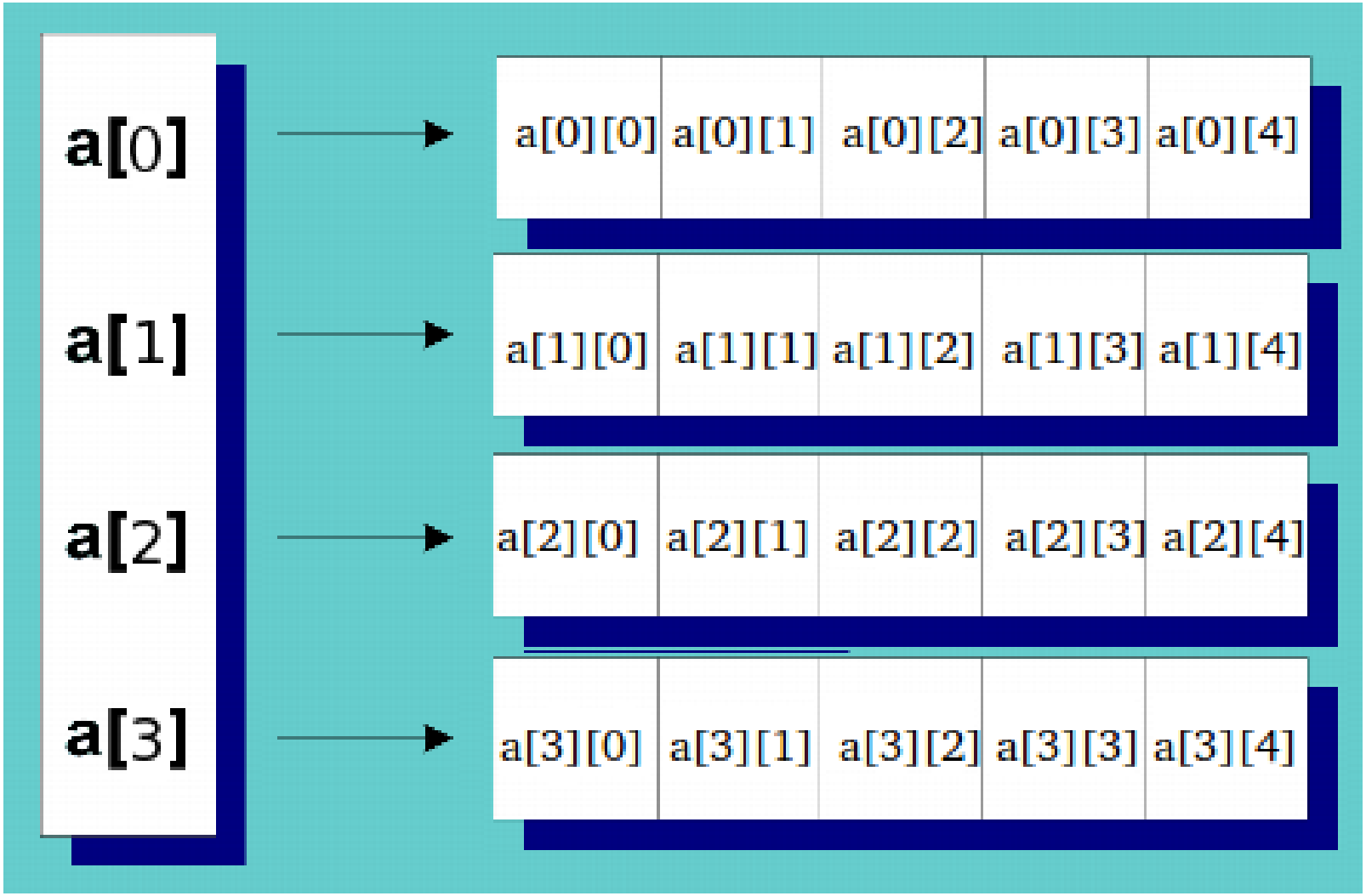
Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Dynamic multidimensional arrays


```
cin >> numRows;
cin >> numCols;

// Allocate memory for rows (left column next)
double **a  = new double *[numRows];

// Allocate memory for columns
for(int i = 0; i < numRows; i++) {
  a[i] = new double[numCols];
}

a[1][2] = 32; // array[row][col]

// deallocate
for(int i = 0; i < numRows; i++) {
  delete[] a[i];
}
delete[] a;
a = nullptr;
```

Array creation and access

double **a = new double *[numRows]; // left column



a[i]=new double[numCols]; // each right row

Multidimensional array templated


```
template <typename T>
T ** AllocateDynamicArray(int nRows, int nCols){
  T **dynamicArray;
  dynamicArray = new T *[nRows];
  for(int i = 0 ; i < nRows ; i++){
    dynamicArray[i] = new T [nCols];
  }
  return dynamicArray;
}
template <typename T>
void FreeDynamicArray(T **dArray, nRows){
  for(int i = 0; i < numRows; i++){delete[] dArray[i];}
  delete[] dArray;
}
int main(){
  int **my2dArr = AllocateDynamicArray<int>(4,4);
  my2dArr[2][2] = 8;
  FreeDynamicArray<int>(my2dArr, 4);
  return 0;
}
```

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Memory leaks

- Memory leaks occur when new memory is allocated dynamically and never deallocated.
- In C++, new memory is usually allocated by the new and new [ ] operators and deallocated by the delete or the delete [ ] operators.
- One of the most common mistakes leading to memory leaks is applying the wrong delete operator.
- Deallocating multi-dimensional arrays can also lead to problems

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Invalid and dangling pointers


```
int *p; // uninitialized pointer

p = new int;
*p = 5;
delete p;
int *q = p; // ?? What if delete p; was last?

int *pArr;
int myarray[10];
pArr = myarray + 20; // ??

int *dynArr;
dynArr = new int[10];
delete dynArr; // correct?
cout << *dynArr << endl; // ??
cout << *(dynArr + 2) << endl; // ??
```

Set dangling pointers to nullptr

Dangling pointer should be assigned to nullprt (old: NULL or 0)

```
int *q = nullptr; // C++ 11; recommended
int *p = 0;
int *r = NULL;

if(q)      // succeeds if p is not null
if(q)     // succeeds if p is null
```

Kahoot

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Dynamic user-defined types


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

class Box{
  private:
    int w, h, d;
  public:
    void setDim(int x, int y, int z) {w=x; h=y; d=z;}
    int volume(){return w*h*d;}
};

int main(){
  Box *myBox = new Box;
  myBox->setDim(4, 5, 2);
  cout << myBox->volume(); // 40
  delete myBox;
  Box *myBoxArray = new Box[4];
  myBoxArray[2].setDim(4, 1, 2);
  delete[ ] myBoxArray; // Delete array
  return 0;
}
```

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Review: objects and this

- Inside every non-static member function, the variable: **T *const this** holds the address of the class object from which the member function was invoked
- **this** represents a pointer to the object whose member function is being executed
- **this** is a hidden parameter accessible in a class's function to refer to the object of which the function is a member:

How many variables are in the two functions below?

```
class Rectangle{
    int width, height;
  public:
    int getArea(){return width*height;} // #param?
    void printWidth(){
      cout << this->width << endl;
      cout << (*this).width << endl;
      cout << width << endl;
    }
}
```

More useful application of this


```
class Rectangle
{
  private:
    int width, height;
  public:
    int getArea(){return width*height;}
    int compare(Rectangle rect){ // #param?
      return this->getArea() > rect.getArea();
    }
}

//Assignment operator= overload
Rectangle & Rectangle::operator=(const Rectangle &rhs)
{
  width = rhs.width;
  height = rhs.height;

  // Allows chaining of operator= when called.
  return *this;
}
```

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Constructors and destructors

The compiler provides each Class has a default constructor, so we can declare via:

```
MyClass classObject;
```

until defining our own parameterized constructor, then we need a new default constructor (or defaulted values)

Further, C++ automatically generates some member functions methods for every class.

1. **copy constructor** used for definition with initialization:
   MyClass B = A;
   MyClass B(A);
   Also called when passing or returning by value, rather than by reference with &
2. **operator=** used for assignment between existing objects:
   A = B = C; (can be chained with multiple assignment)
3. **destructor** called automatically when a class goes out of scope, or is explicitly deallocated with delete

Constructors and destructors: main


```
int main() {
  // parameterized constructor
  Rectangle rect(3, 4, 253);

  // default constructor
  Rectangle recta;

  // copy constructor
  Rectangle rectb(rect);
  Rectangle rectc = rect;

  // operator= assignment (not constructor)
  recta = rect;

  return 0;
}
```

Constructors and destructors: declarations


```
class Rectangle{
    int width, height, *pfill;

  public:
    // parameterized constructor
    Rectangle(int, int, int);

    // new default constructor
    Rectangle();

    // copy constructor
    Rectangle(const Rectangle &);

     // assignment (overload operator, not constructor)
    const Rectangle & operator=(const Rectangle &);

    // destructor uses ~ in front of class name
    ~Rectangle();

    int printFill() {return *pFill;}
};
```

Constructors and destructors: definitions


```
Rectangle::Rectangle(int a, int b, int fillVal){
  width = a;  height = b;
  pFill = new int(fillVal);
}

Rectangle::Rectangle(){
  width = 5;  height = 5;
  pFill = new int(255);
}

Rectangle::Rectangle(const Rectangle &source){
  width = source.width;
  height = source.height;
  // pFill = source.pFill; // shallow copy pointer itself
  pFill = new int(*(source.pFill)); // deep copy contents
}

const Rectangle & Rectangle::operator=(const Rectangle &rhs){
  if (this != &rhs){
    width = rhs.width;
    height = rhs.height;
    // pFill = rhs.pFill; // shallow copy pointer itself
    *pFill = *(rhs.pFill); // deep copy contents
    return *this;
  } // what if pFill was an array?  delete[] old first for size mismatch?
}

Rectangle::~Rectangle(){
  delete pFill;
}
```

Watch out for shallow and deep copy!

Kahoot

Outline

Dynamic memory
Dynamic variables
Dynamic memory
delete
Dynamic arrays
delete[ ]
Multidimensional arrays
Problems
Memory leaks
Invalid and dangling pointers
Dynamic user-defined types
this
Structs and classes with dynamic memory
Constructors and destructors
Requirements of dynamic members

Class with dynamic members

If a your user-defined class with dynamically allocated members is to function in all typical ways, you likely need to to re-write its:

1. Default constructor
2. Parameterized constructor
3. Copy constructor
4. Assignment operator=
5. Default destructor

Guidelines for classes with dynamic memory

- Initialize pointers in the constructor! If not allocating space right away, best to initialize to **nullptr** until ready for use.
- Use **new** inside class member functions to allocate space
- Use **delete** to clean up dynamically allocated space whenever finished using it. Do so in the destructor, which is the last function that runs for an object
- Isolate memory management tasks from the functionality/algorithmic tasks wherever possible: Write a set of member functions just for dealing with memory management issues – like creation of space, deallocation, resizing, etc. Your algorithmic functions can call the memory-handling functions, when needed