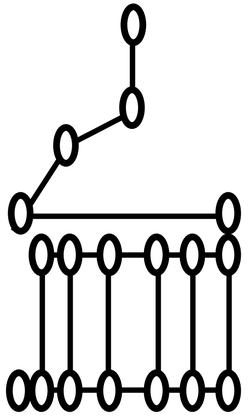
Lecture: Dynamically allocated memory

ENGR 2730: Computers in Engineering



Motivation for dynamically allocated memory

- Dynamic memory means memory allocated at run time.
- Dynamic memory doesn't have a name, pointers are used to access this memory
- Dynamically allocated memory must be explicitly de-allocated
- The size of an array is often not known at compile time
- Dynamic memory requests memory once we know how much we need
- Pointers are used "return" newly created "arrays" from functions (e.g., making a copy of an array)

instead of:

```
int data[10] = {0};
int *ptr = data;
```

(hard to know size at compile-time)

want something like:

```
int *ptr;
int n = 10;
ptr = (enough memory for n integers)
```

The new/delete operators

The new/delete operators

C++

dynamically allocates memory → new

releases the dynamically allocated memory → delete

The **new** command performs the following tasks:

1. Finds unused memory to store object.
2. Calls the object constructor.
3. Returns the address of the allocated memory on success or **nullptr** on failure.

The **delete** command performs the following tasks:

1. Calls the object destructor.
2. Releases allocated memory so it can be reused by the program.

The new/delete operators

C++

dynamically allocates memory → new

releases the dynamically allocated memory → delete

The **new** command knows the size of the object that it is creating and allocates the correct number of bytes to store object.

Example:

```
int * xptr = new int; // allocate memory
*xptr = 5;

cout << "*xptr = " << *xptr << endl;

delete xptr;        // release memory
xptr = nullptr;     // pointer exists after delete
                    // delete does not delete pointer
```

The new/delete operators

Dynamic allocation of one object

Example:

```
int * xptr = new int; // allocate memory
*xptr = 5;

cout << "*xptr = " << *xptr << endl;

delete xptr;   // release memory
xptr = nullptr;
```

Dynamic allocation of an array

Example:

```
int * xptr = new int[10]; // allocate memory
xptr[0] = 4;
xptr[9] = 5;

cout << "xptr[0] = " << xptr[0] << endl;
cout << "xptr[9] = " << xptr[9] << endl;

delete [] xptr;   // release memory
xptr = nullptr;
```

Note: Built-in primitive types (e.g., int, char, float, etc.) are considered to have constructors and destructors. However, there will not be any code generated to make explicit constructor or destructor calls for built-in types.
C++ does not initialize primitive types to zero! Default values for primitive types are undefined!

The new/delete operators

Dynamic allocation of one object

Example:

Allocates memory for one Complex object.
Then calls constructor with parameters 5 and 4.

```
Complex * xptr = new Complex(5, 4);

cout << "*xptr = " << *xptr << endl;

delete xptr;
```

Calls destructor once.
Then releases memory for one Complex object.

Dynamic allocation of an array

Example:

Allocates memory for 10 Complex objects.
Then calls constructor 10 times.

```
Complex * xptr = new Complex[10];
xptr[0].setReal(1); xptr[0].setImag(2);
xptr[9].setReal(3); xptr[9].setImag(4);

cout << "xptr[0] = " << xptr[0] << endl;
cout << "xptr[9] = " << xptr[9] << endl;

delete [] xptr;
```

Calls destructor 10 times.
Then releases memory for 10 Complex objects.

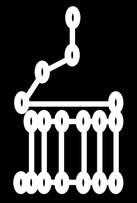
Example: Read data from file of unknown size

- Goal: Read data from file of unknown size
- Challenge: Number of elements in file are unknown a priori
- **Solution 1: Assume a maximum list size.**
  - Drawback: This is designing for worst case and is wasteful w.r.t. memory.
- **Solution 2: Allocate memory at run-time, i.e., dynamic memory allocation**
  - Advantages:
    Efficient w.r.t. memory requirements.
    Adapts to the best solution each time program runs.
    Allows memory to be released when no longer needed.

Example: Read data from file of unknown size

- Solution 2: Pseudo code

1. Compute size of file
2. Dynamically allocate array to hold data
3. Read data from file into array
4. Close file
5. Print data to screen
6. Free memory



Example: Read data from file of unknown size

```
int main () {
    ifstream fin("data.txt", ifstream::binary); // Assume file contains binary

    if (fin) {
        // get length of file:
        fin.seekg(0, fin.end);    // Go to end of stream (i.e., file)
        int length = fin.tellg();  // Get current position of stream
        fin.seekg(0, fin.beg);    // Go to start of stream (i.e., file)

        char * buffer = new char [length]; // Dynamically allocate buffer array

        fin.read(buffer,length);   // Read entire file into buffer

        fin.close();               // Finish with file so close it

        cout.write(buffer,length); // Print buffer array to screen

        delete [] buffer;          // Free buffer array memory
    }

    return 0;
}
```

data.txt

```
2.3
1.7
4.2
5.1
6.6
```

Output

```
2.3
1.7
4.2
5.1
6.6
```

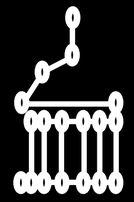
New/delete operator for objects

C++

```
int main()
{
    Robot *robot;

    robot = new Robot;
    robot->MoveForward(5);
    robot->Speak("Hi");
    .
    .
    .
    robot->Dance();
    delete robot;
}
```

After calling **delete**, don't use the pointer unless you reinitialize it!



New/delete operator for arrays

C++

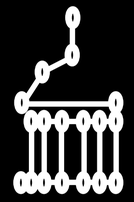
```
int main()
{
    int *myDynamicArray = nullptr;
    int n = 10;

    myDynamicArray = new int[n];

    for (int i=0; i < n; i++)
    {
      myDynamicArray[i] = i;
      cout << "myDynamicArray[" << i << "] = " << myDynamicArray[i] << endl;
    }

    delete [] myDynamicArray;
}
```

note the use of [ ]



Dynamically Allocated Array Example

```
int main(){
    /* allocate space for 10 integer values */
    int *data = new int[10];
    if (data != nullptr){
        /* initialize all values to zero */
        for (int i = 0; i < 10; i++){
            data[i] = 0;
        }

        /* modify select values using different syntax examples */
        *data = 5;       /* set first int (element 0) to 5 */
        data[1] = 3;     /* set second int (element 1) to 3 */
        *(data + 4) = 1; /* set fifth int (element 4) to 1 */

        /* display all values */
        printArray(data, 10);

        /* free memory */
        delete [] data;
    } else {
        cout << "Error: unable to allocate memory." << endl ;
    }
}
```

use **new** to allocate memory for 10 integers

initialize all elements to zero

modify select elements

display all values

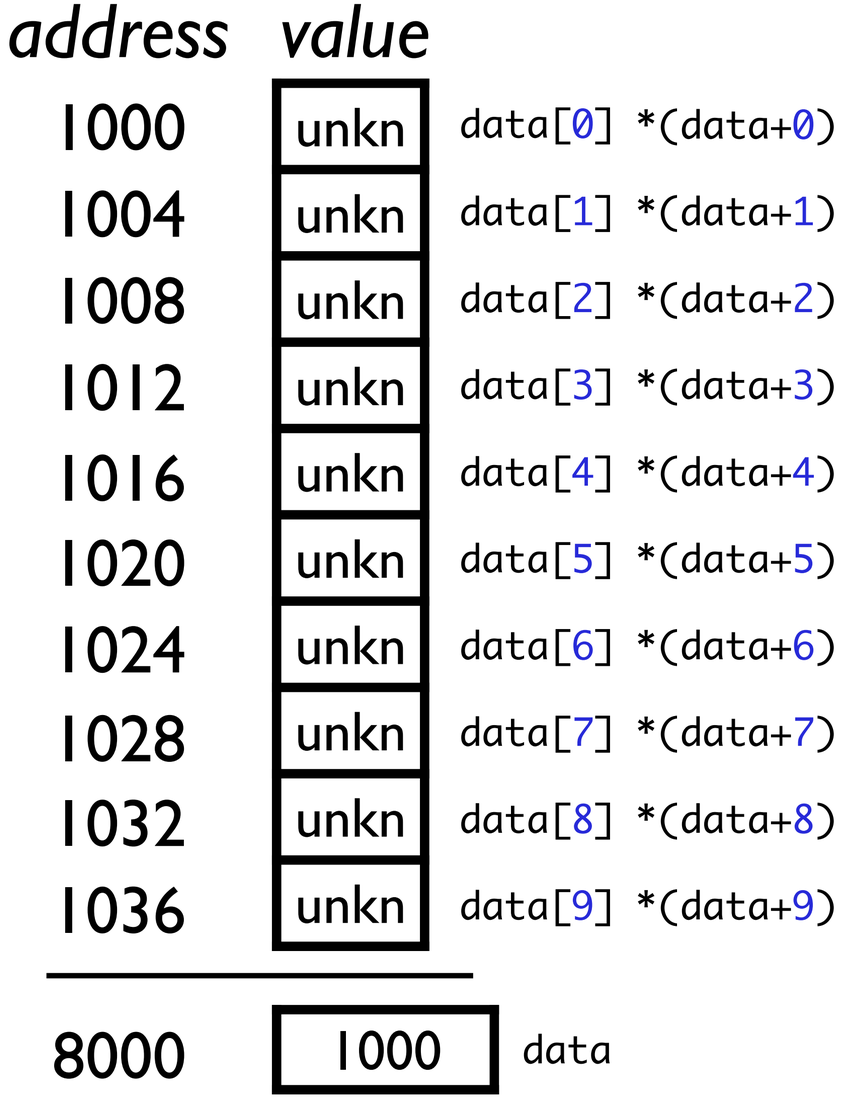
free memory back to system

Example view of memory

```
/* allocate space for 10 integer values */
int *data = new int[10];

if (data != nullptr)
{
    /* initialize all values to zero */
    for (int i = 0; i < 10; i++)
    {
        data[i] = 0;
    }

    *data = 5;
    data[1] = 3;
    *(data + 4) = 1;
```

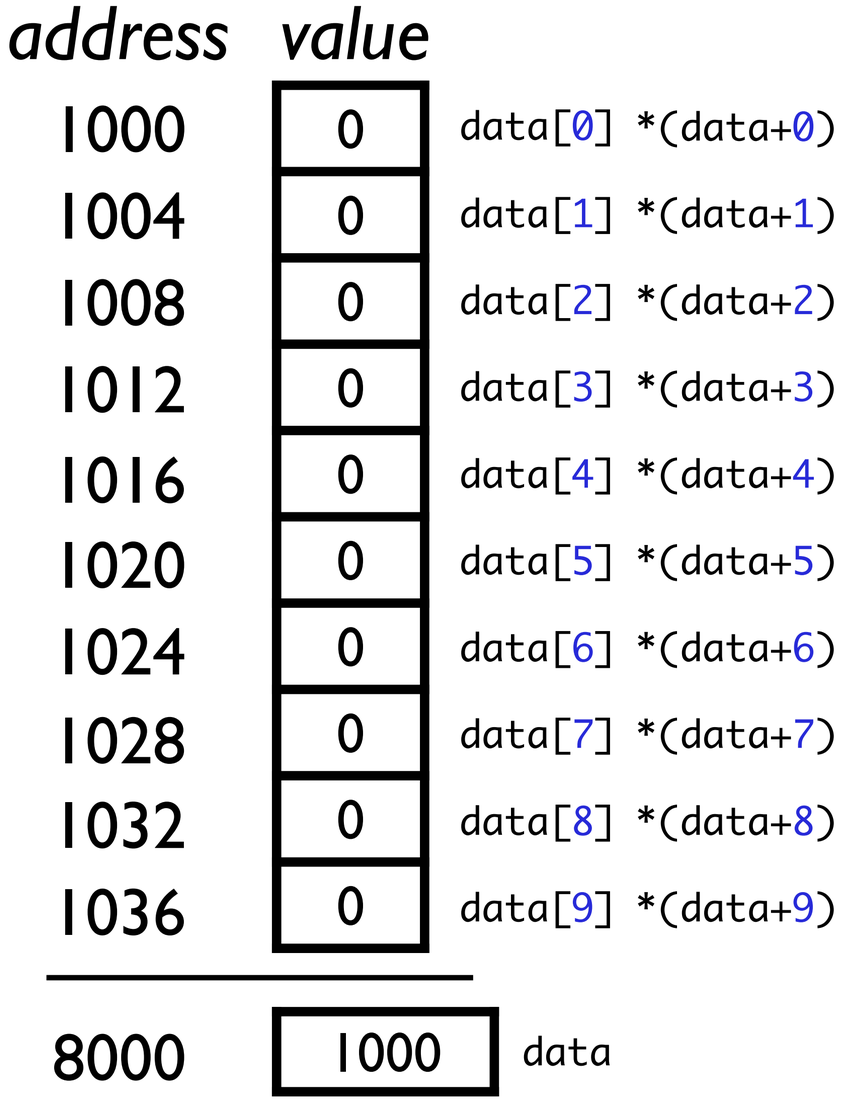


Example view of memory

```
/* allocate space for 10 integer values */
int *data = new int[10];

if (data != nullptr)
{
    /* initialize all values to zero */
    for (int i = 0; i < 10; i++)
    {
        data[i] = 0;
    }

    *data = 5;
    data[1] = 3;
    *(data + 4) = 1;
```



Example view of memory

```
/* allocate space for 10 integer values */
int *data = new int[10];

if (data != nullptr)
{
    /* initialize all values to zero */
    for (int i = 0; i < 10; i++)
    {
        data[i] = 0;
    }

    *data = 5;
    data[1] = 3;
    *(data + 4) = 1;
```

address	*value*	
1000	5	data[0] *(data+0)
1004	0	data[1] *(data+1)
1008	0	data[2] *(data+2)
1012	0	data[3] *(data+3)
1016	0	data[4] *(data+4)
1020	0	data[5] *(data+5)
1024	0	data[6] *(data+6)
1028	0	data[7] *(data+7)
1032	0	data[8] *(data+8)
1036	0	data[9] *(data+9)

8000	1000	data

Example view of memory

```
/* allocate space for 10 integer values */
int *data = new int[10];

if (data != nullptr)
{
    /* initialize all values to zero */
    for (int i = 0; i < 10; i++)
    {
        data[i] = 0;
    }

    *data = 5;
    data[1] = 3;
    *(data + 4) = 1;
```

address	*value*	
1000	5	data[0] *(data+0)
1004	3	data[1] *(data+1)
1008	0	data[2] *(data+2)
1012	0	data[3] *(data+3)
1016	0	data[4] *(data+4)
1020	0	data[5] *(data+5)
1024	0	data[6] *(data+6)
1028	0	data[7] *(data+7)
1032	0	data[8] *(data+8)
1036	0	data[9] *(data+9)
8000	1000	data

Example view of memory

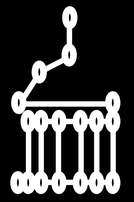
```
/* allocate space for 10 integer values */
int *data = new int[10];

if (data != nullptr)
{
    /* initialize all values to zero */
    for (int i = 0; i < 10; i++)
    {
        data[i] = 0;
    }

    *data = 5;
    data[1] = 3;
    *(data + 4) = 1;
```

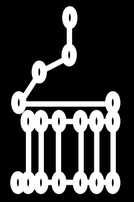
address	*value*	
1000	5	data[0] *(data+0)
1004	3	data[1] *(data+1)
1008	0	data[2] *(data+2)
1012	0	data[3] *(data+3)
1016	1	data[4] *(data+4)
1020	0	data[5] *(data+5)
1024	0	data[6] *(data+6)
1028	0	data[7] *(data+7)
1032	0	data[8] *(data+8)
1036	0	data[9] *(data+9)

8000	1000	data



In-class Program

- Create a new CLion project called DynamicMemoryExample
- Create a new text file in the project called “data.txt”
- Add an arbitrary number of double numbers to the data.txt file.
- Write a program to read in the numbers and count how many numbers are in the file.
- Dynamically allocate a data array of the correct size to hold the numbers in the file.
- Reset the file stream to the beginning of the file
- Read the numbers into the array
- Print the numbers from the array to the screen.



Solution

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main(){

    ifstream fin("data.txt");
    if (fin.fail()){
        cout << "Error" << endl;
        return -1;
    }

    // Count the number of values in file
    int count = 0;
    double value;

    while (!fin.eof()){
        fin >> value;
        if(!fin.fail()){
            count++;
        }
    }

    // Dynamically allocate memory for array
    double * data = new double[count];
    if (data == nullptr){
        cout << "Error" << endl;
        return -2;
    }

    // Reset file stream to start of file
    fin.clear(); // Clear the EOF flag
    fin.seekg(0,fin.beg);

    // Read data from file
    for(int i=0; i<count; i++){
        fin >> data[i];
    }

    // Print data
    for(int i=0; i<count; i++){
        cout << data[i] << endl;
    }

    // Close file and free memory
    fin.close();
    delete [] data;

    return 0;
}
```