Pointer Usages & Dynamic Memory

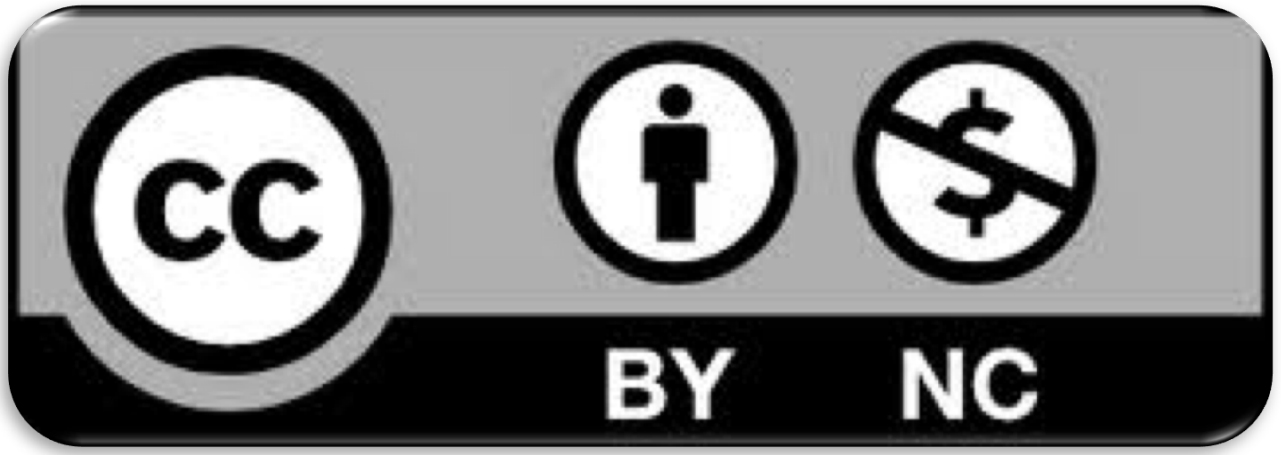
Alessandro SAVINO
Politecnico di Torino (Italy)

alessandro.savino@polito.it

www.testgroup.polito.it

License Information

This work is licensed under the Creative Commons BY-NC License



To view a copy of the license, visit:
http://creativecommons.org/licenses/by-nc/3.0/legalcode

Disclaimer

- **We disclaim any warranties or representations as to the accuracy or completeness of this material.**
- **Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.**
- **Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.**

Goal

– **This lecture presents a global overviews of problems and issues related to dynamic memory allocation and pointers usage**

Prerequisites

- **Basic knowledge of C programming language**

Homework

- **None**

Outline

- **Pointers Usage:**
  - **Pointers and functions**
  - **Dynamic Memory**
  - **Declare and Scan Arrays**
  - **Dynamic Memory C++ style**

Rel. 08/04/2016

Outline

- **Pointers Usage:**
  - **Pointers and functions**
  - **Dynamic Memory**
  - **Declare and Scan Arrays**
  - **Dynamic Memory C++ style**

Pointers and functions

- **Passing pointers to functions**
  - ***Nonconstant* pointer to *nonconstant* data**

```
void f(int *ptr);

int main ()
{
   int y=1, x=2;
   int *ptr =&y;
   f(ptr);
   ptr = &x;
   return 0;
}

void f(int *ptr){
   *ptr = 20;
}
```

Pointers and functions

- **Passing pointers to functions**
  - **Nonconstant pointer to constant data**

```
void f(const int *ptr);

int main ()
{
   int y=1, x=2;
   int *ptr =&y;
   f(ptr);
   ptr = &x;
   return 0;
}

void f(const int *ptr){
   *ptr = 20; // ERROR assignment of read-only location
}
```

Pointers and functions

- **Passing pointers to functions**
  - **Constant pointer to nonconstant data**

```
void f(int *ptr);

int main ()
{
  int x=1, y=2;
  int * const ptr = &y;
  f(ptr);
  ptr = &x; //ERROR assignment of read-only location
  return 0;
}

void f(int *ptr){
  *ptr = 20;
}
```

Pointers and functions

- **Passing pointers to functions**
  - **Constant pointer to constant data**

```
void f(const int *ptr);

int main ()
{
  int x=1, y=2;
  int * const ptr = &y;
  f(ptr);
  ptr = &x; //ERROR assignment of read-only location
  y=20;
  return 0;
}

void f(const int *ptr){
  *ptr = 20;//ERROR assignment of read-only location
}
```

Pointers and functions

- **Function pointers**
  - **A pointer to a function contains the function address in memory**
  - **The function name can be used as the function pointer**
  - **A function pointer can be passed to other functions.**

Pointers and functions

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

int f1(int val);
int f2(int val);
int f3(int val);
int call_f(int val, int (*funct)(int));

int main ()
{
  int x=5;
  cout << x << endl;
  cout << call_f(x,f1) << endl;
  cout << call_f(x,f2) << endl;
  cout << call_f(x,f3) << endl;
  return 0;
}
```

Pointers and functions

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

int f1(int val);
int f2(int val);
int f3(int val);
int call_f(int val, int (*funct)(int));

int main ()
{
  int x=5;
  cout << x << endl;
  cout << call_f(x,f1) << endl;
  cout << call_f(x,f2) << endl;
  cout << call_f(x,f3) << endl;
  return 0;
}
```

The function receives:

- an integer as first parameter
- a pointer to a function as a second parameter

Pointers and functions

```
int f1(int val){
   return ++val;
}

int f2(int val){
   return --val;
}

int f3(int val){
   return val=0;
}

int call_f(int val, int (*funct)(int)){
    return funct(val);
}
```

Outline

- **Pointers Usage:**
  - **Pointers and functions**
  - **Dynamic Memory**
  - **Declare and Scan Arrays**
  - **Dynamic Memory C++ style**

Dynamic Memory

- ***Dynamic memory allocation*** **allows programmers to write code that requires an amount of memory that is NOT FIXED "a priori"**
- **The memory can be dynamically allocated or be freed during the program execution.**

Dynamic Memory

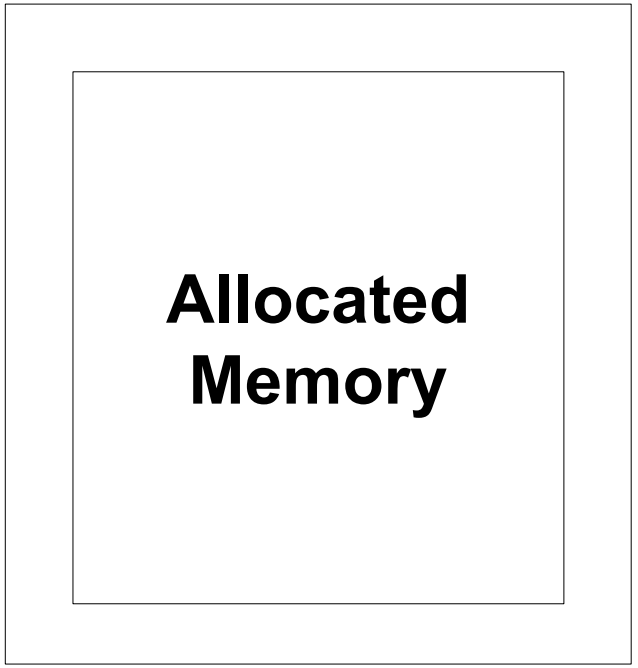
- **There are two scenarios in which *dynamic memory allocation* can be exploited:**
  1. **The program is able to determine, at each execution, how much memory it needs**

1st Scenario

Main memory

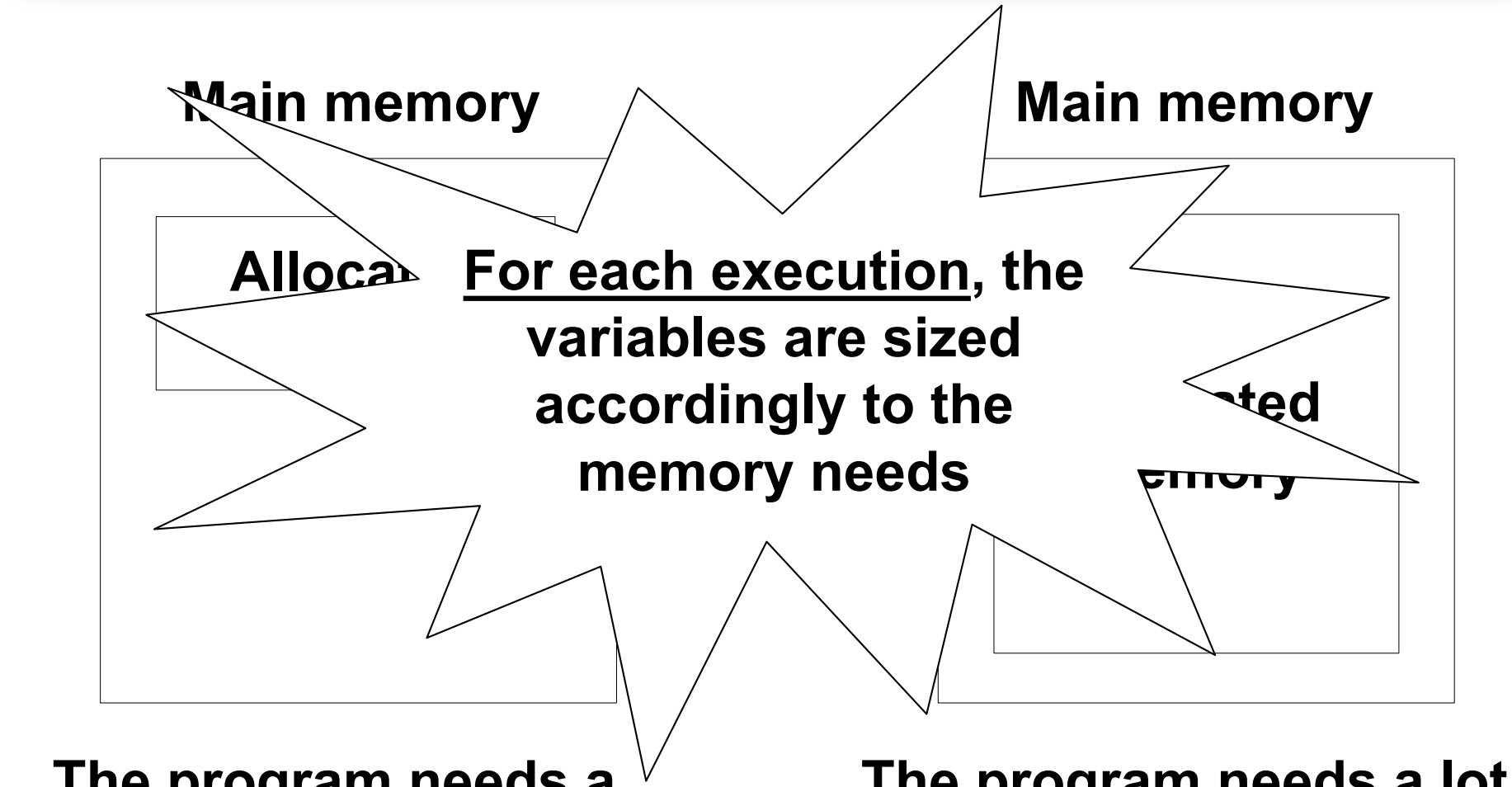
The program needs a small amount of memory

Main memory



The program needs a lot of memory

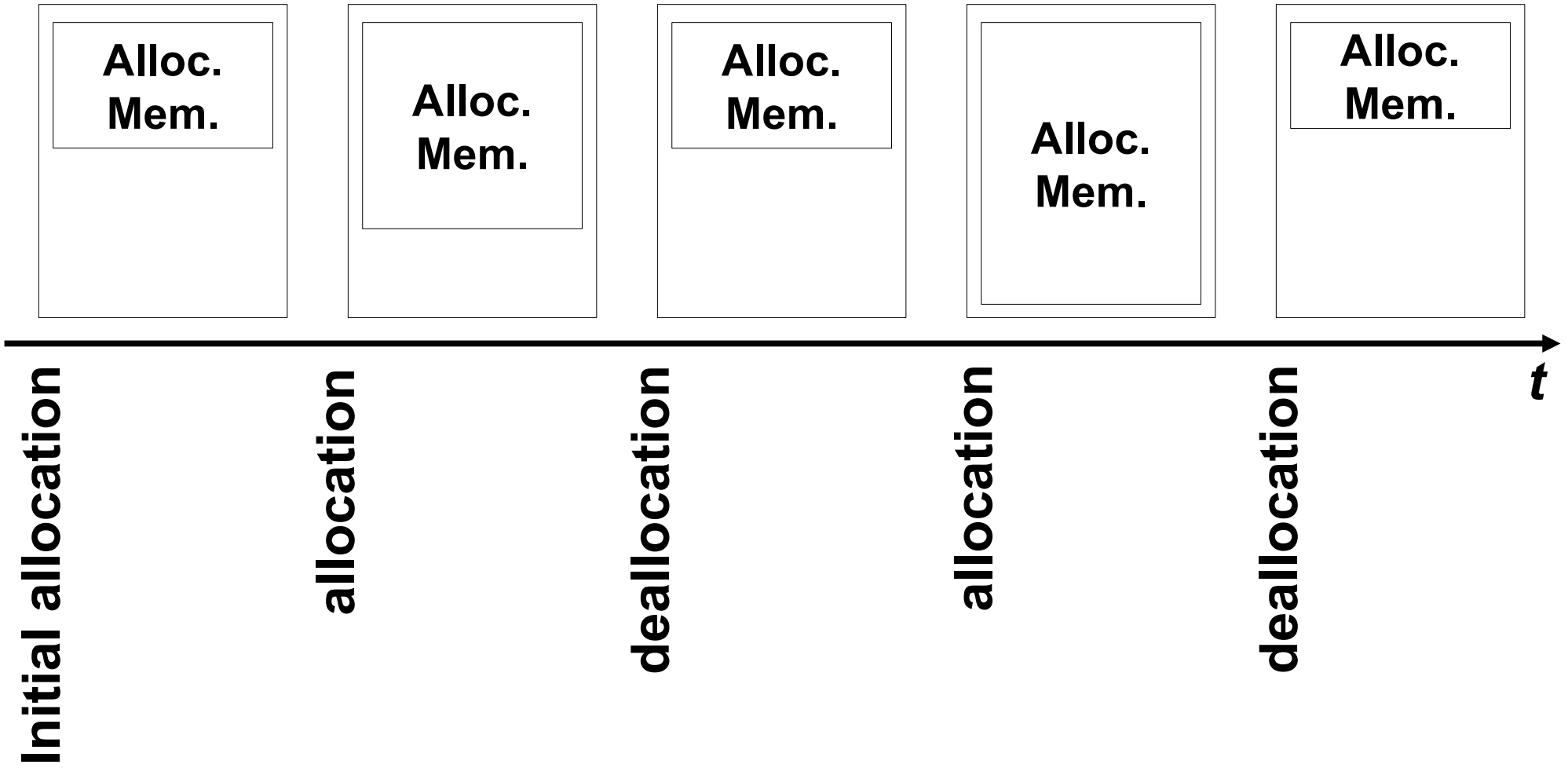
1st Scenario



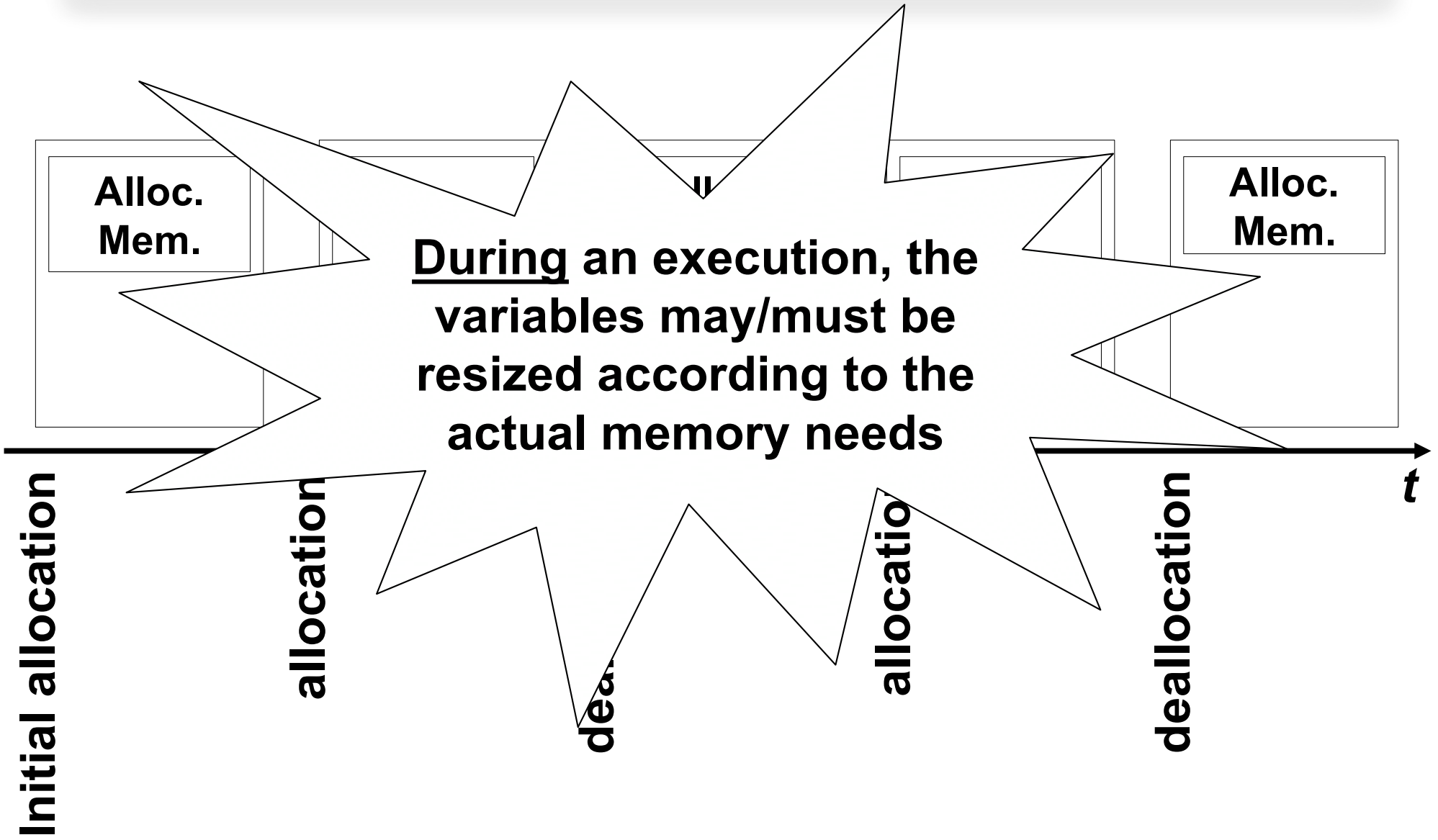
Dynamic Memory

- **There are two scenarios in which *dynamic memory allocation* can be exploited:**
  1. **The program is able to determine, at each execution, how much memory it needs**
  2. **During execution, the program needs a variable amount of memory**

2nd Scenario



2nd Scenario



How can we <u>dynamically</u> allocate and deallocate memory?

Dynamic Memory Allocation

- **The main function to allocate memory in C is:**

```
void *malloc ( <memory_size> );
```

It asks the Operating System to:

- **allocate a memory portion of bytes having size equal to `<memory_size>`**
- **return the pointer to the beginning of the allocated space.**

Dynamic Memory Allocation

ction to allocate memor

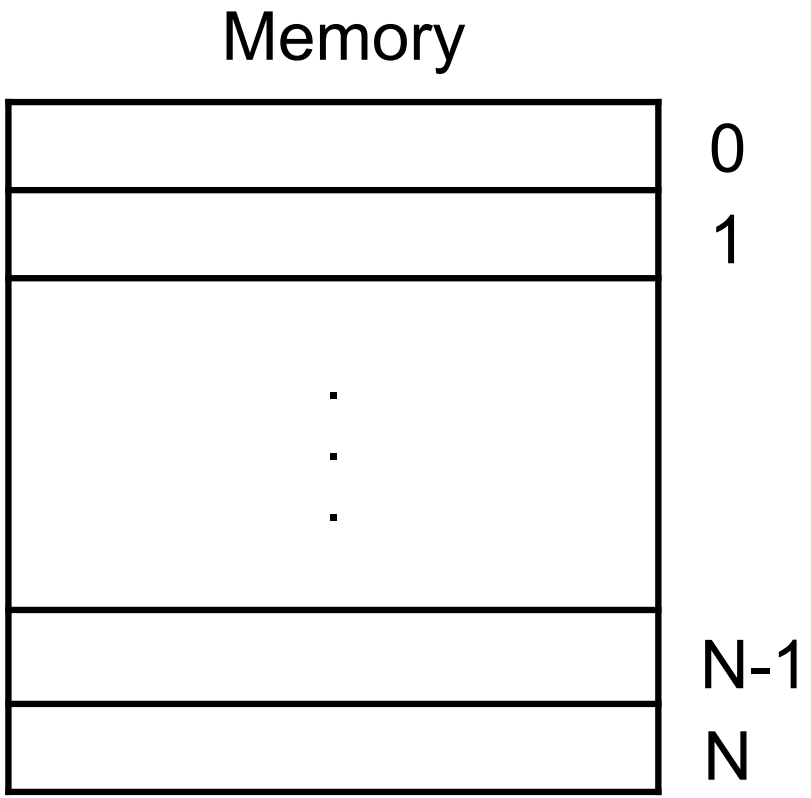
Check the return value against NULL!
In that case, an error during allocation arises, e.g., out of memory!

It

allo

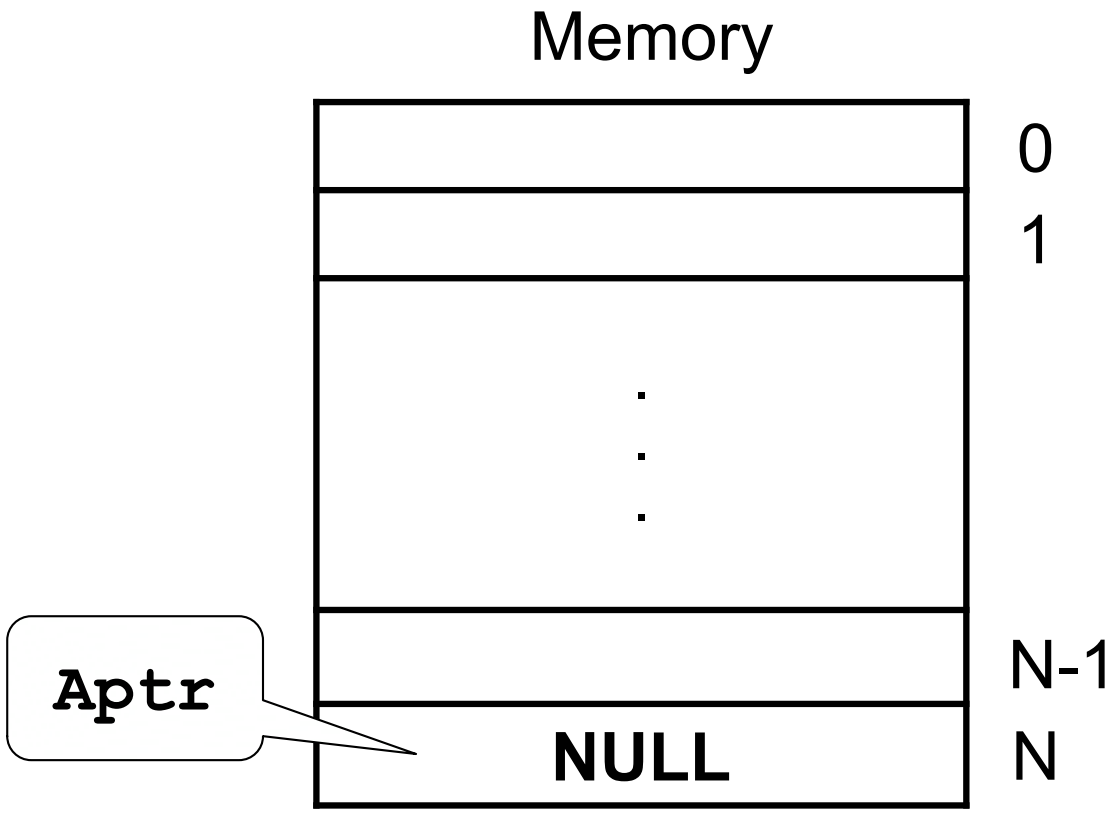
malloc ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
```



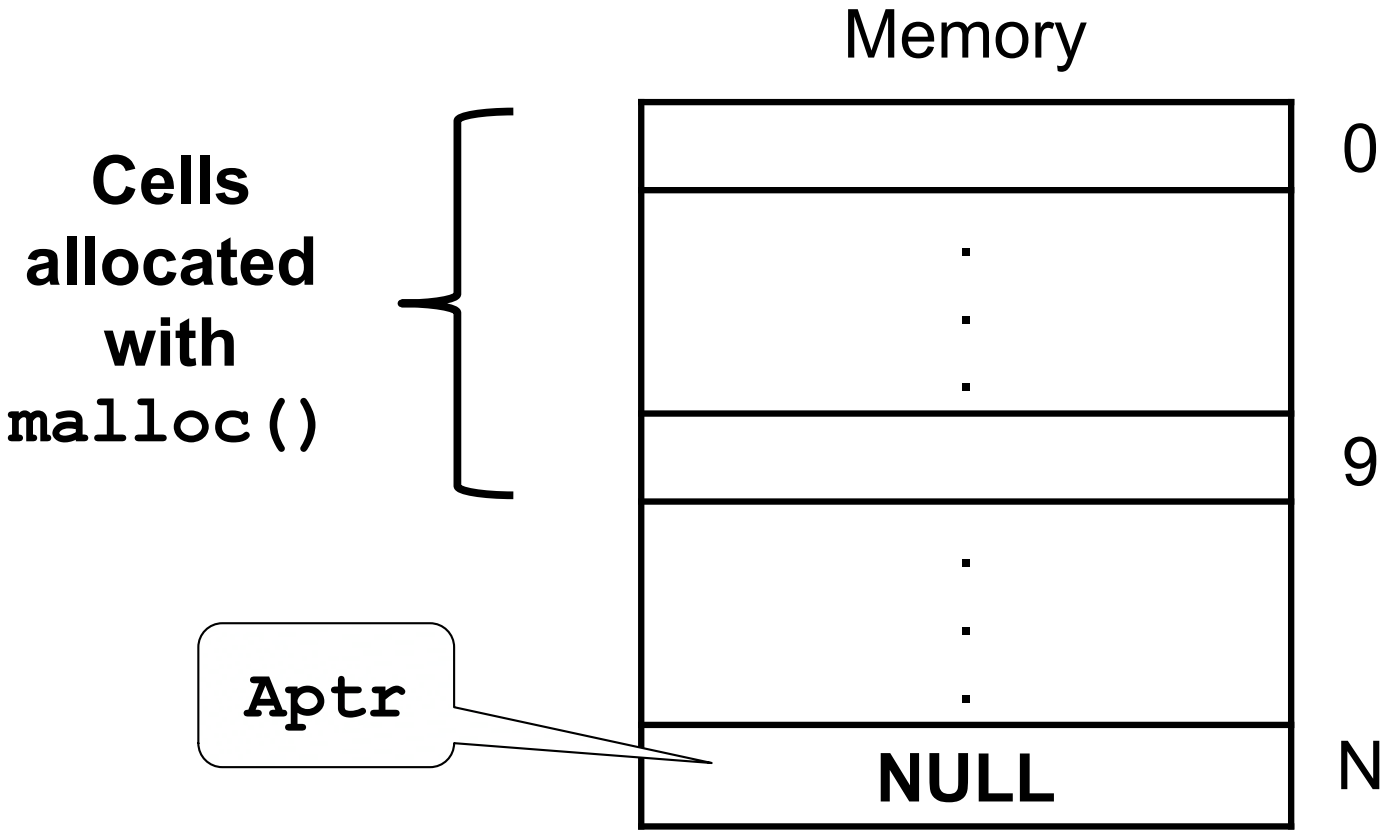
malloc ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
```



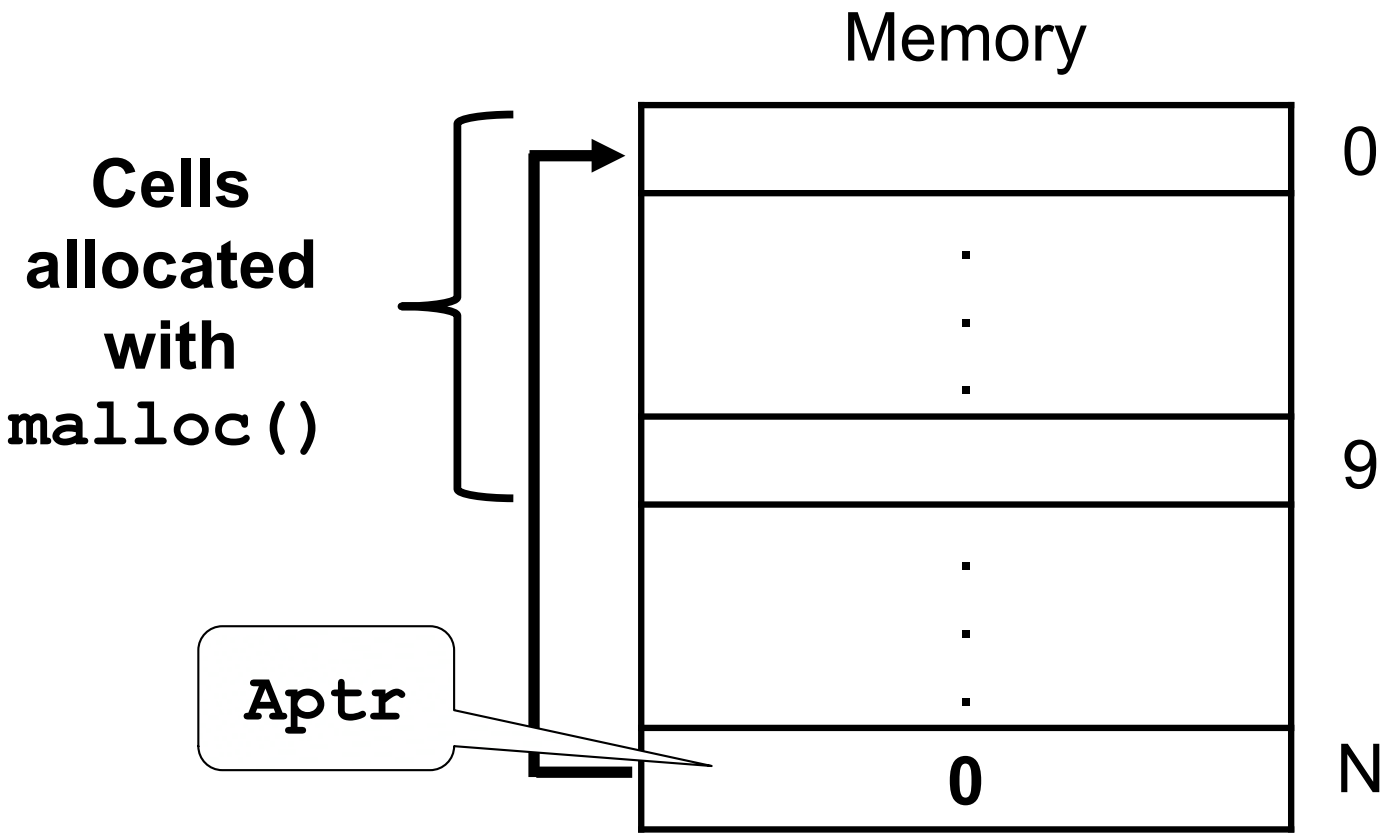
malloc ( )

```
   char *Aptr = NULL;
▷ Aptr = (char*) malloc(10 * sizeof (char));
```



malloc ( )

```
   char *Aptr = NULL;
▷  Aptr = (char*) malloc(10 * sizeof (char));
```



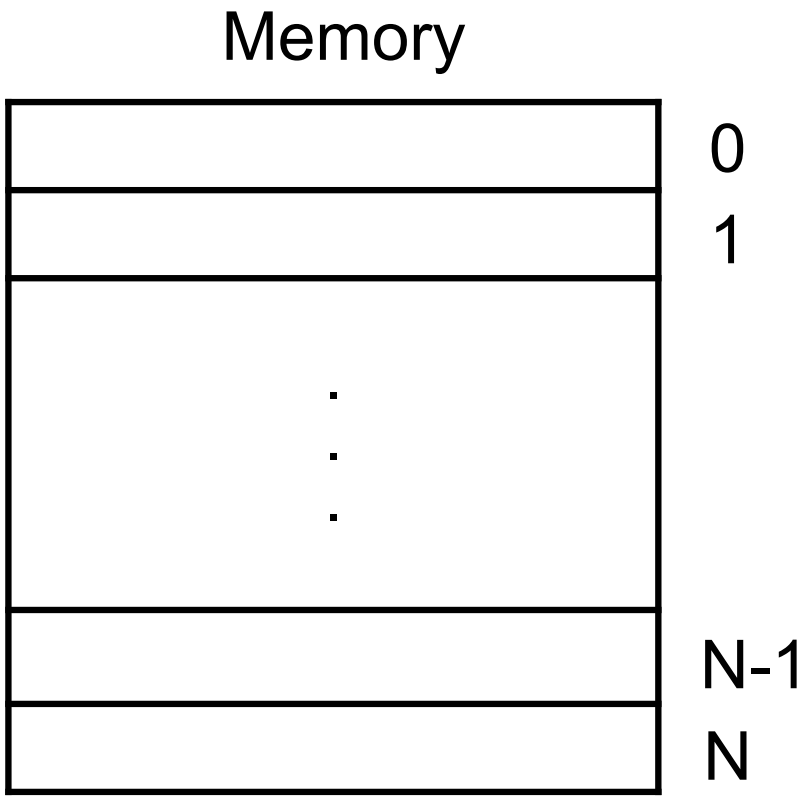
Dynamic Memory Deallocation

- **In C there are two main functions to dynamically deallocate memory:**
  - `void* realloc (void* ptr, <memory_size>);`

It asks the Operating System to resize, accordingly to `<memory_size>`, the memory portion pointed by `ptr`, and returns the pointer to the beginning of the reallocated space

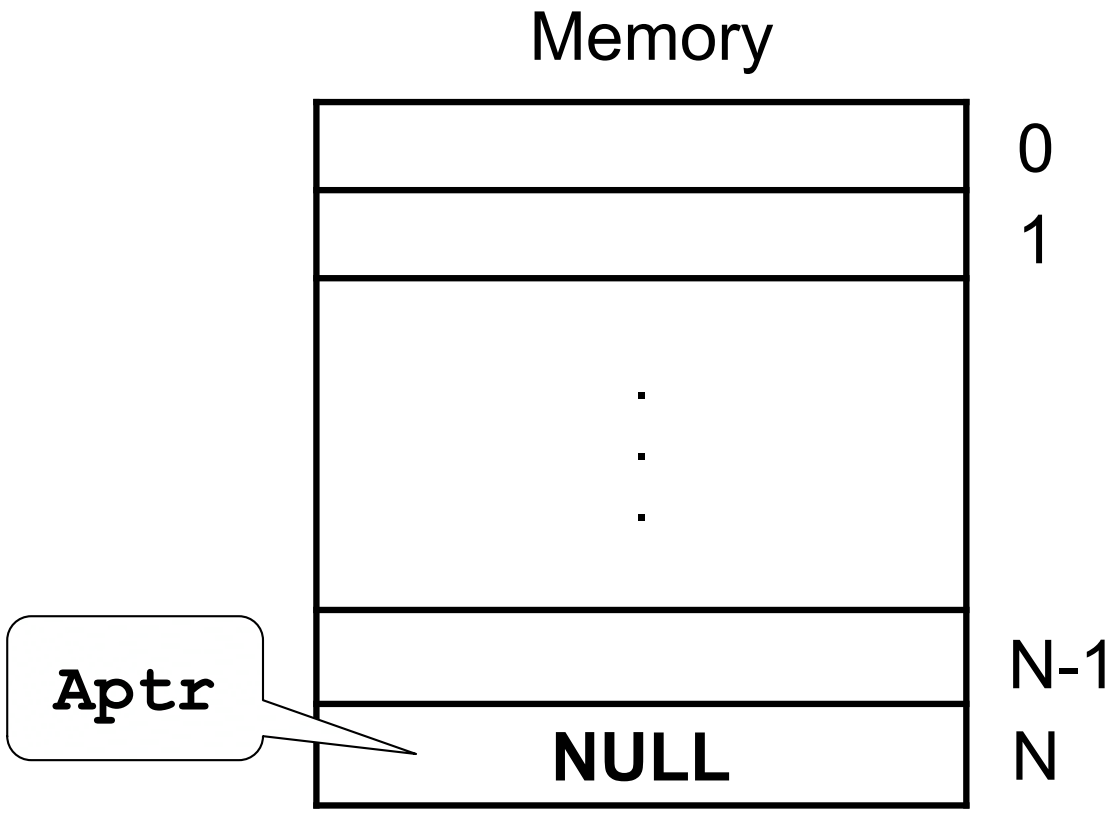
realloc ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
Aptr = (char*) realloc(Aptr, 2* sizeof (char));
```



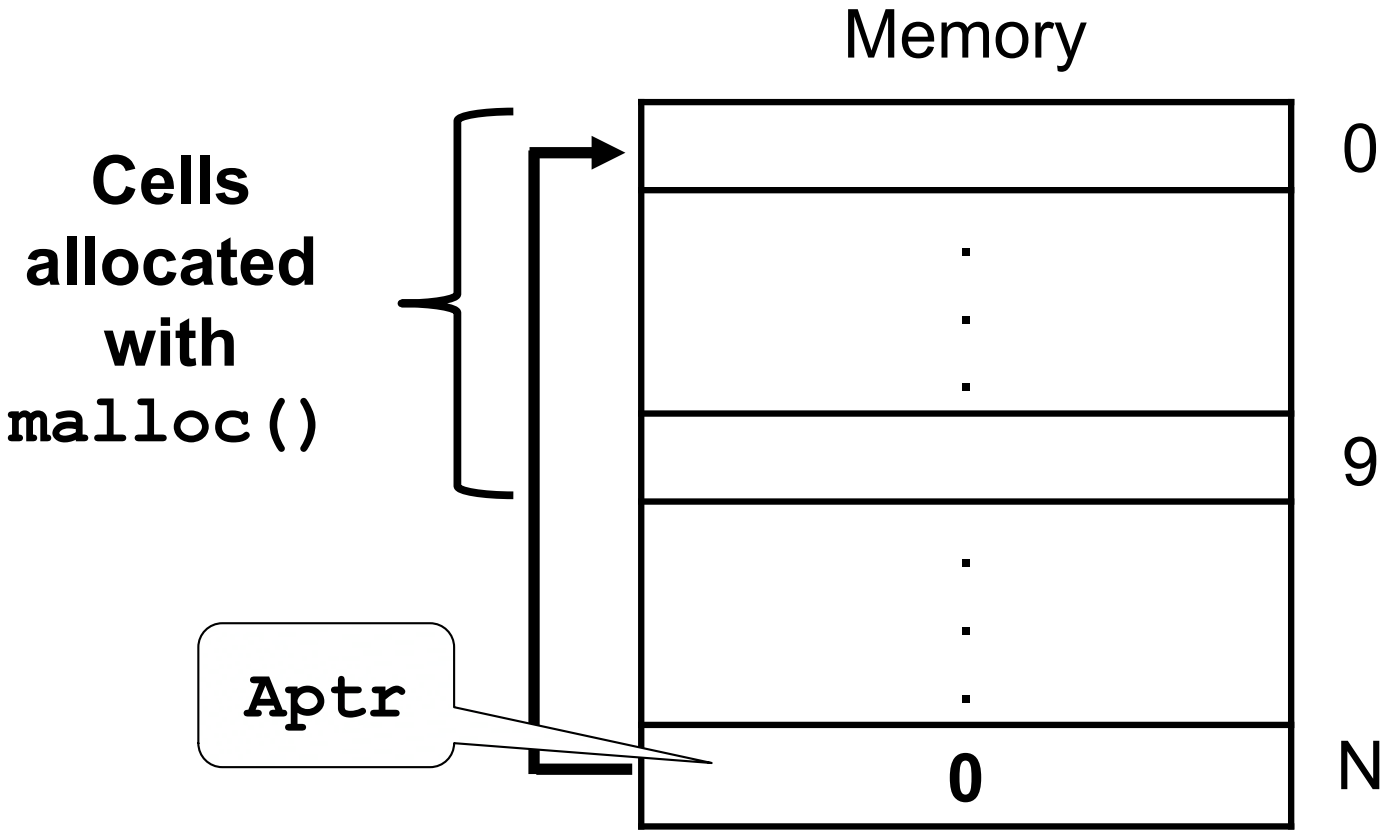
realloc ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
Aptr = (char*) realloc(Aptr, 2* sizeof (char));
```



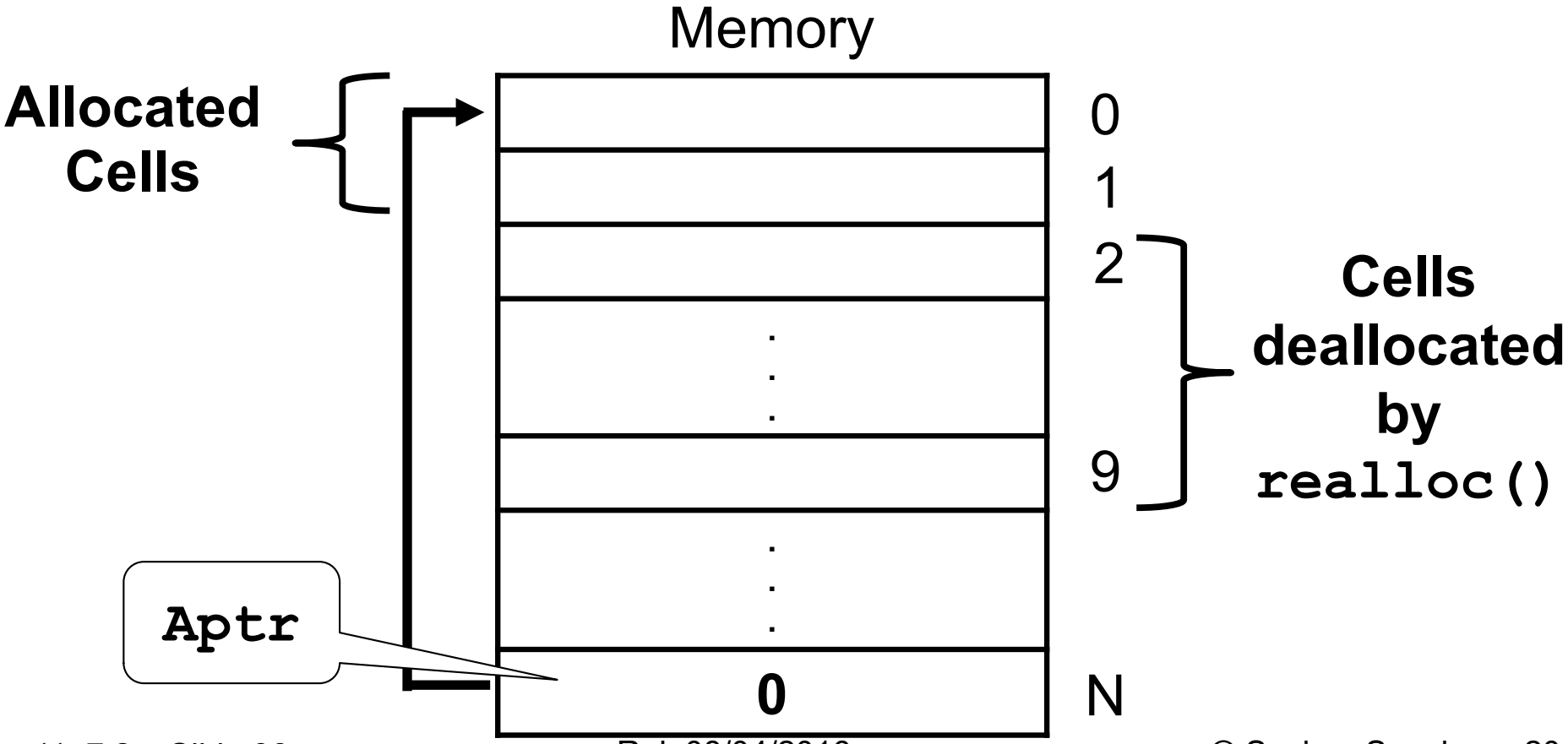
realloc ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
Aptr = (char*) realloc(Aptr, 2* sizeof (char));
```



realloc ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
Aptr = (char*) realloc(Aptr, 2* sizeof (char));
```



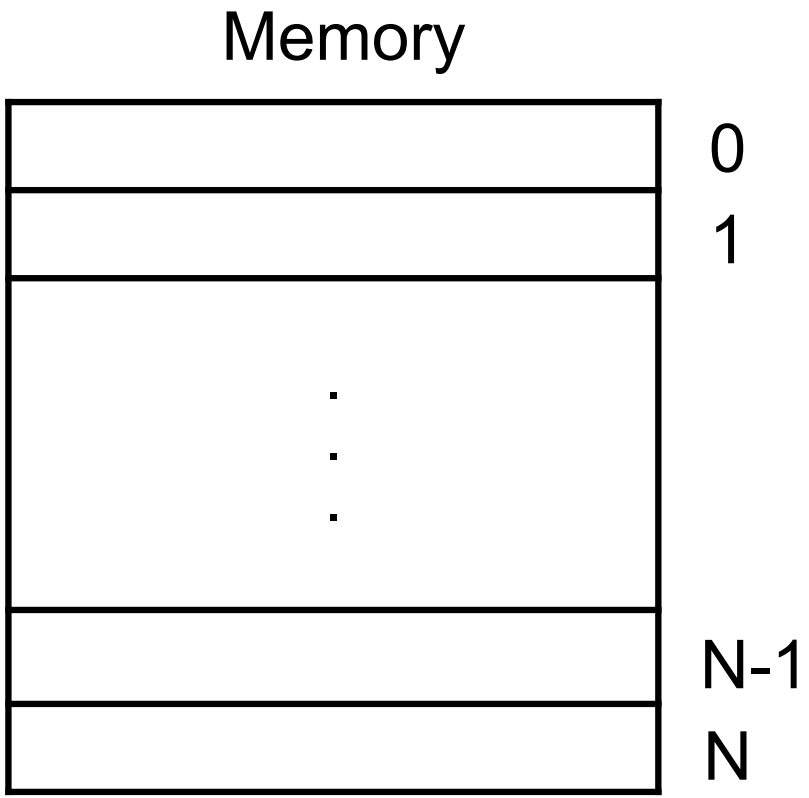
Dynamic Memory Deallocation

- **In C there are two main functions to dynamically deallocate memory:**
  - `void* realloc (void* ptr, <memory_size>);`
  - `void free (void* ptr);`

It asks the Operating System to deallocate the memory portion pointed by `ptr`

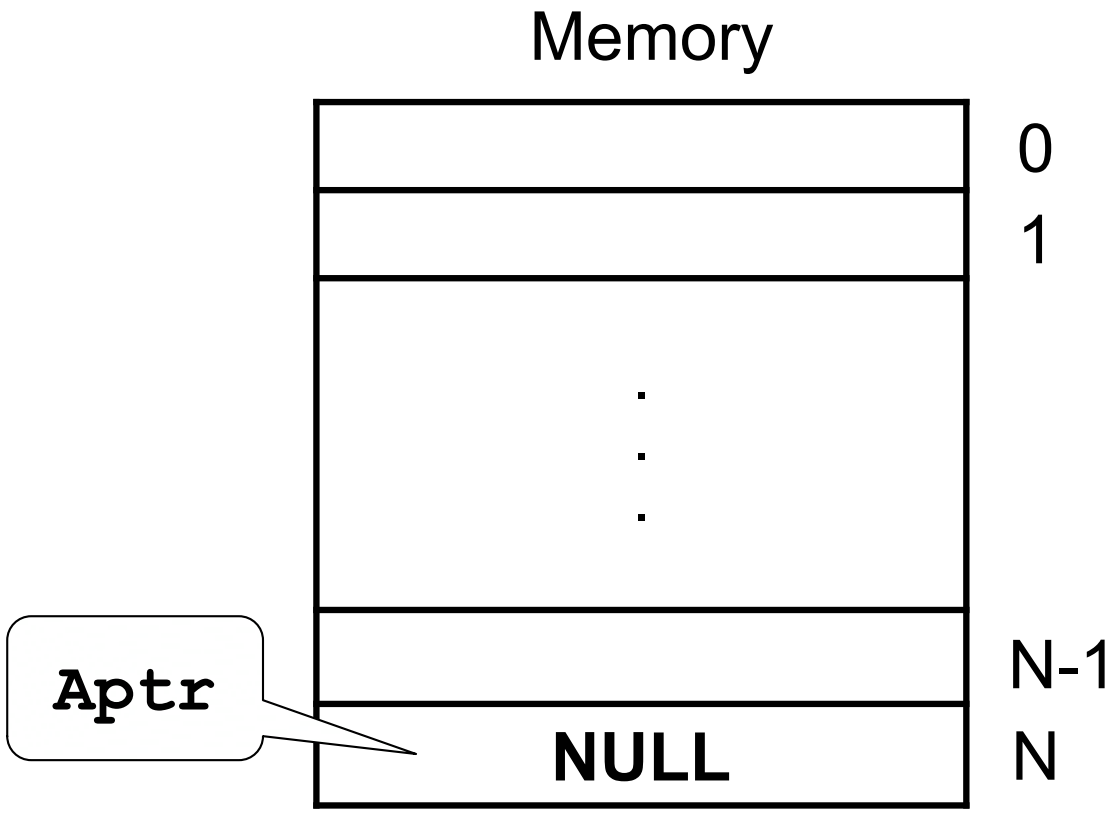
free ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
free(Aptr);
```



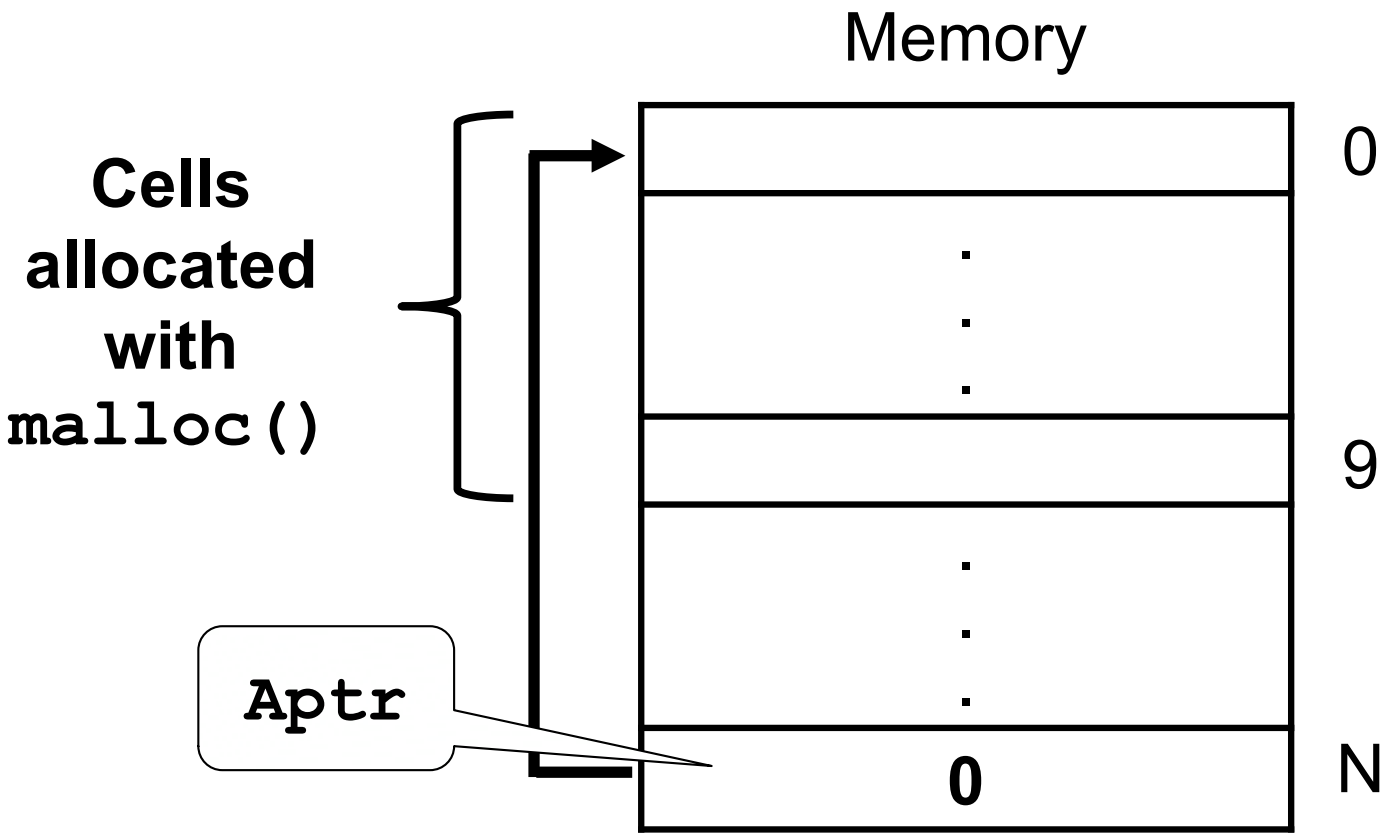
free ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
free(Aptr);
```



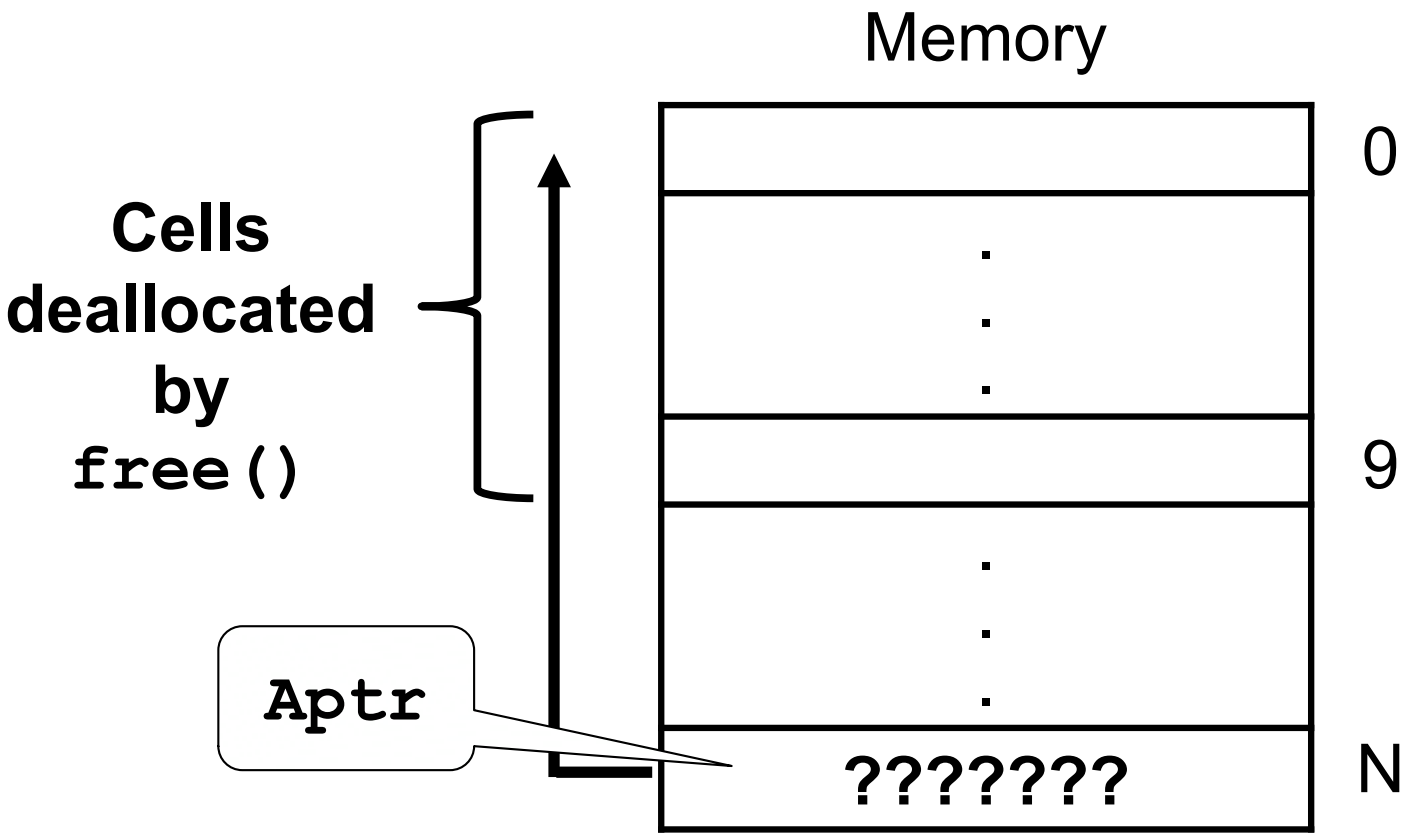
free ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
free(Aptr);
```



free ( )

```
char *Aptr = NULL;
Aptr = (char*) malloc(10 * sizeof (char));
free(Aptr);
```



Outline

- **Pointers Usage:**
  - **Pointers and functions**
  - **Dynamic Memory**
  - **Declare and Scan Arrays**
  - **Dynamic Memory C++ style**

Declare and Scan Arrays with Pointers

- **Pointers and arrays are <u>almost</u> interchangeable in C**

```
int A[10] ≅ int *APtr
```


Rel. 08/04/2016

Why only <u>almost</u> interchangeable ?

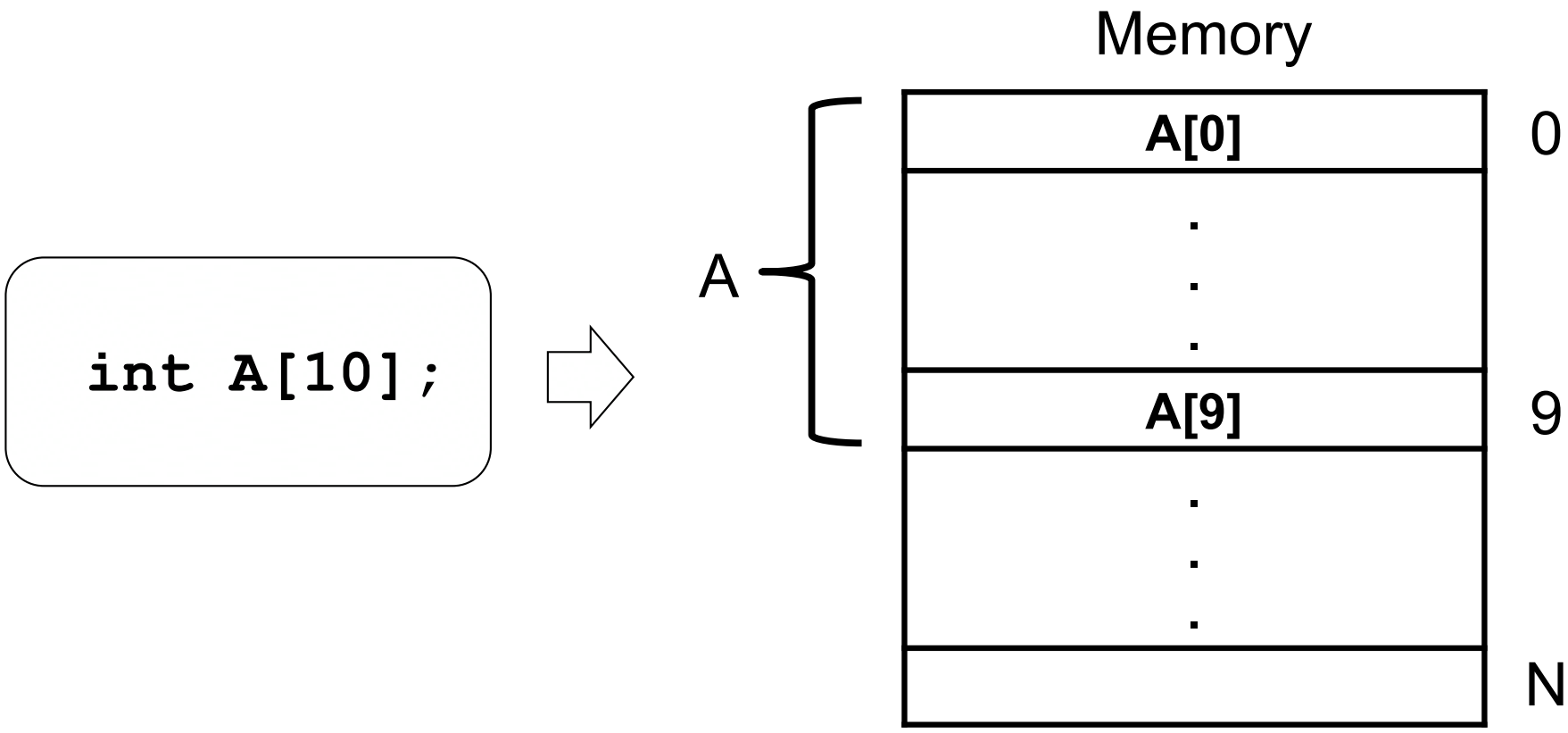
Pointers and Arrays - Difference

- **Main difference:**
  - **When declaring an array, size is specified and memory is allocated <u>statically</u>**

```
int A[10];
```

Pointers and Arrays – Difference *(cnt'd)*

- **Main difference:**
  - **When declaring an array, size is specified and memory is allocated <u>statically</u>**



Pointers and Arrays – Difference (cnt'd)

- **Main difference:**
  - **When declaring an array, size is specified and memory is allocated statically**
  - **When declaring a pointer, no additional memory is initially allocated**

Memory must be allocated exploiting the `malloc()` function

Rel. 08/04/2016

Declare and Scan Arrays with Pointers

- **A pointer can be used to declare and to scan an array exploiting:**
  - **Dynamic memory allocation**
  - **Pointers arithmetic**

Declare and Scan Arrays with Pointers

```
char *Aptr = NULL;
Aptr    = (char*)malloc(10*sizeof(char));
*Aptr   = 'A';
 Aptr   = Aptr + 5;
*Aptr   = 'B';
 Aptr   = Aptr - 4;
*Aptr   = 'C';
*Aptr++ = 'D';
    .
    .
    .
```

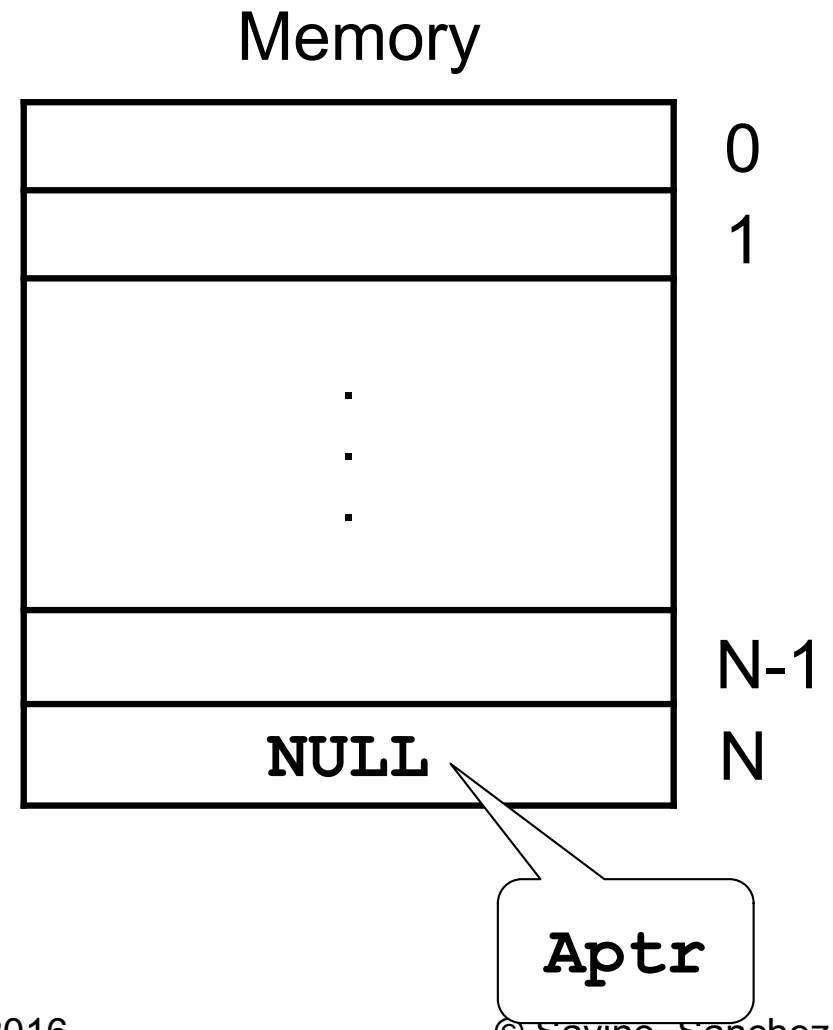
Declare and Scan Arrays with Pointers

```
char *Aptr = NULL;
Aptr    = (char*)malloc(10*sizeof(char));
*Aptr   = 'A';
 Aptr   = Aptr + 5;
*Aptr   = 'B';
 Aptr   = Aptr - 4;
*Aptr   = 'C';
*Aptr++ = 'D';
    .
    .
    .
```

Declare and Scan Arrays with Pointers

Current Statement:

```
char *Aptr = NULL;
```



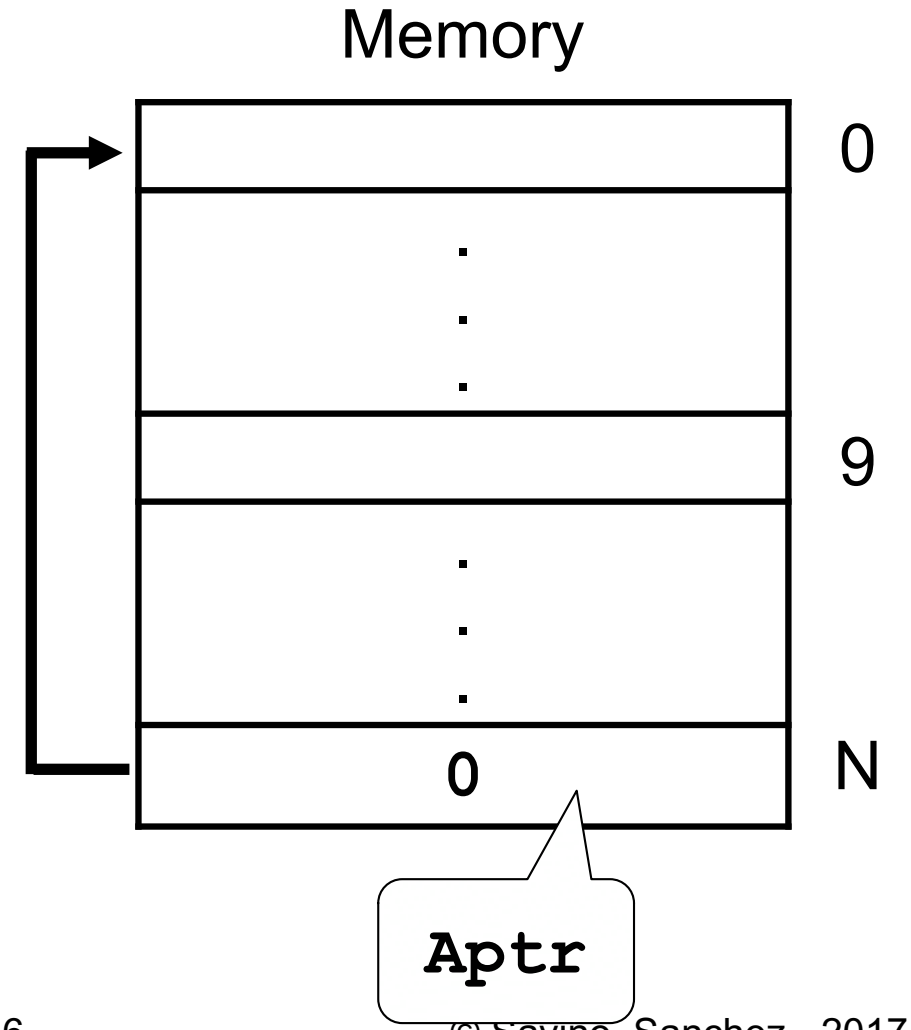
Declare and Scan Arrays with Pointers

```
   char *Aptr = NULL;
▷  Aptr    = (char *)malloc(10*sizeof(char));
   *Aptr   = 'A';
    Aptr   = Aptr + 5;
   *Aptr   = 'B';
    Aptr   = Aptr - 4;
   *Aptr   = 'C';
   *Aptr++ = 'D';
       .
       .
       .
```

Declare and Scan Arrays with Pointers

Current Statement:

```
  Aptr = (char*)malloc(
10*sizeof (char));
```



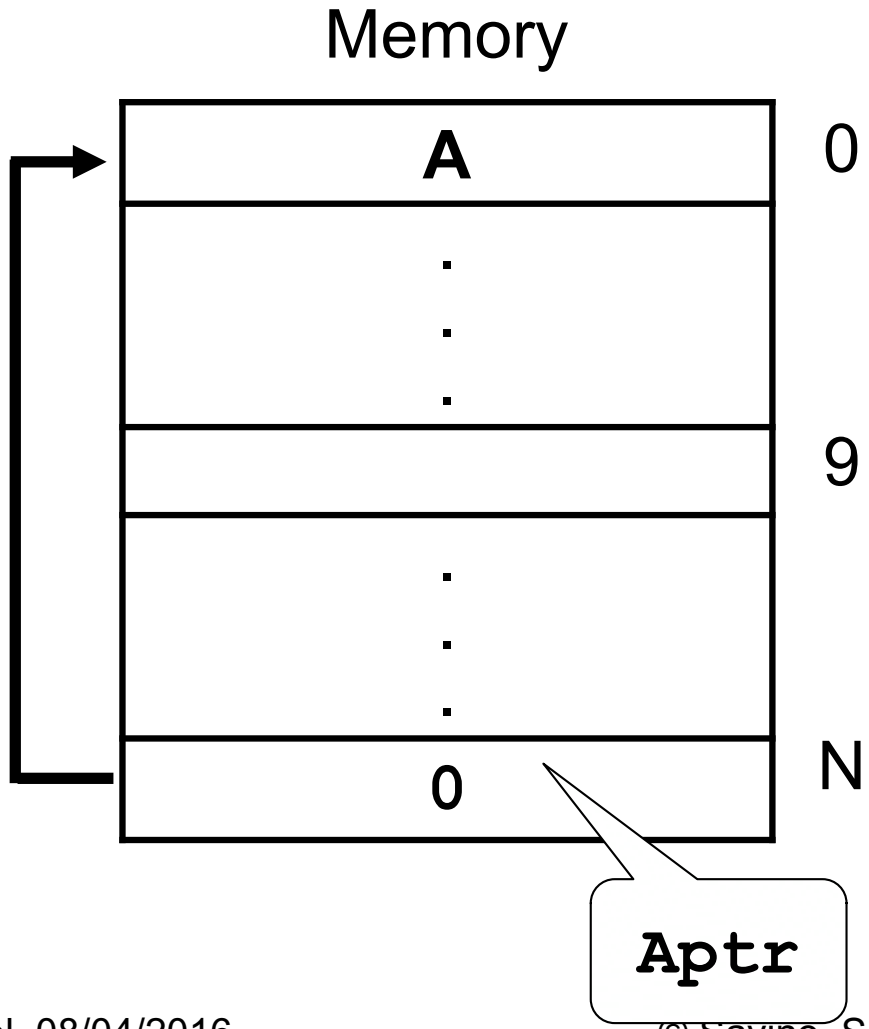
Declare and Scan Arrays with Pointers

```
char *Aptr = NULL;
Aptr    = (char*)malloc(10*sizeof(char));
*Aptr   = 'A';
 Aptr   = Aptr + 5;
*Aptr   = 'B';
 Aptr   = Aptr - 4;
*Aptr   = 'C';
*Aptr++ = 'D';
    .
    .
    .
```

Declare and Scan Arrays with Pointers

Current Statement:

```
*Aptr = 'A';
```



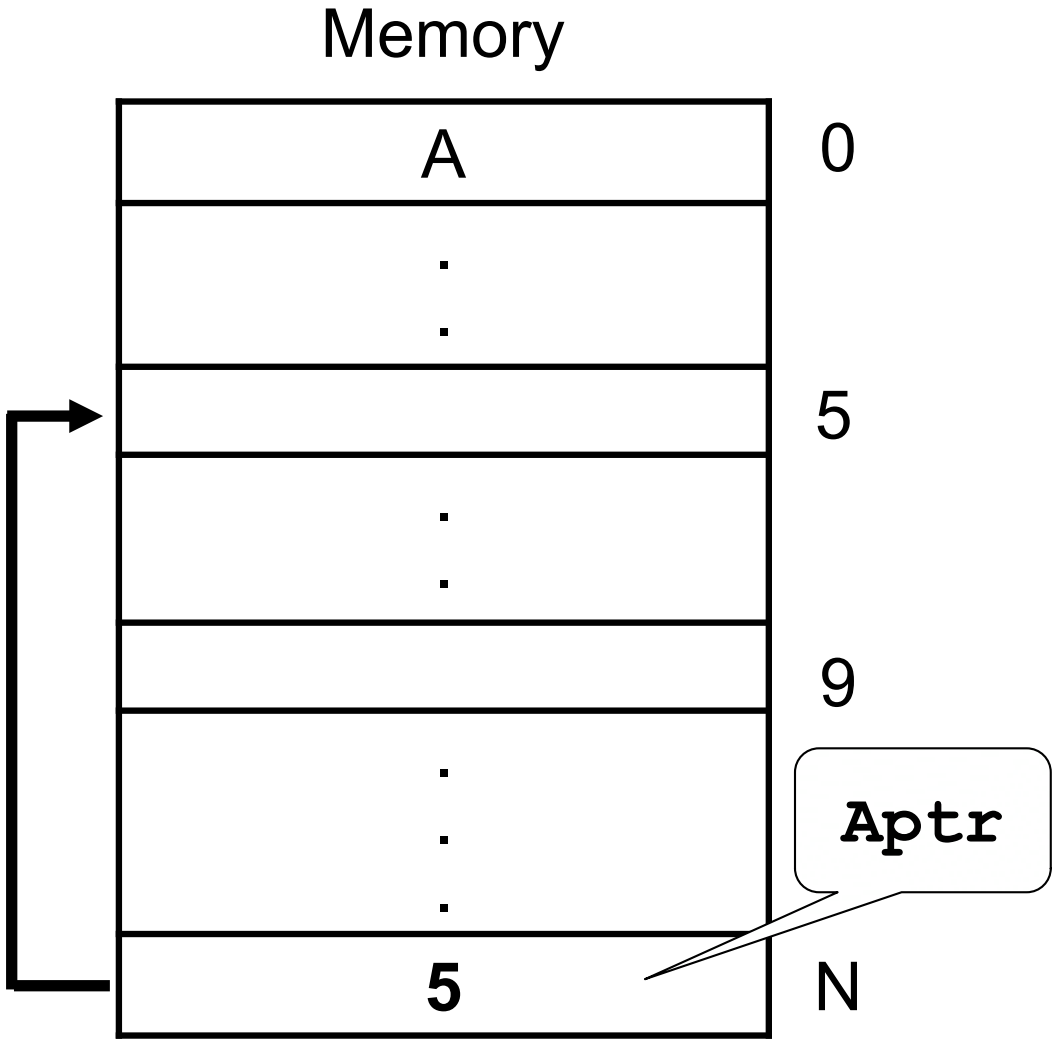
Declare and Scan Arrays with Pointers

```
char *Aptr = NULL;
Aptr    = (char*)malloc(10*sizeof(char));
*Aptr   = 'A';
 Aptr   = Aptr + 5;
*Aptr   = 'B';
 Aptr   = Aptr - 4;
*Aptr   = 'C';
*Aptr++ = 'D';
    .
    .
    .
```

Declare and Scan Arrays with Pointers

Current Statement:

```
Aptr = Aptr + 5;
```



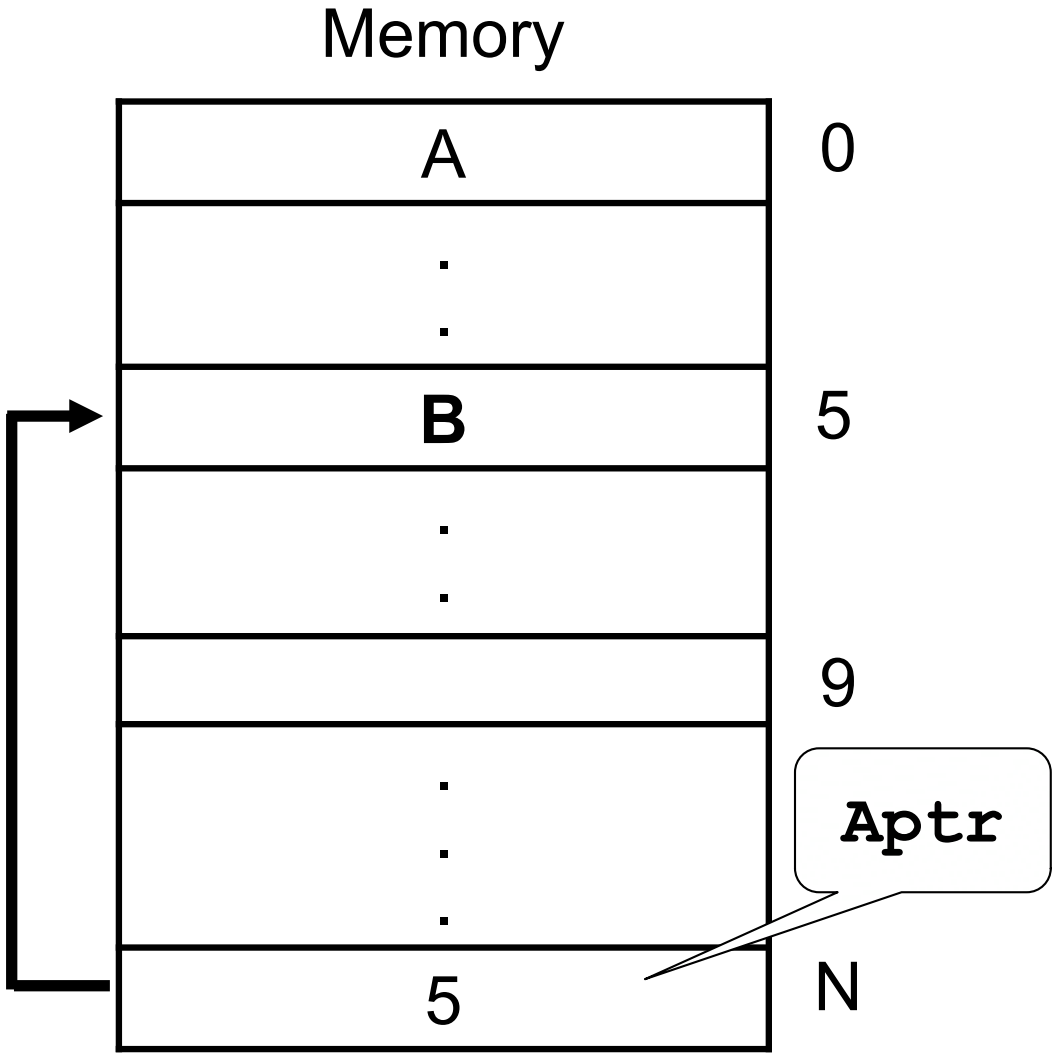
Declare and Scan Arrays with Pointers

```
char *Aptr = NULL;
Aptr    = (char*)malloc(10*sizeof(char));
*Aptr   = 'A';
 Aptr   = Aptr + 5;
*Aptr   = 'B';
 Aptr   = Aptr - 4;
*Aptr   = 'C';
*Aptr++ = 'D';
    .
    .
    .
```

Declare and Scan Arrays with Pointers

Current Statement:

```
*Aptr = 'B';
```



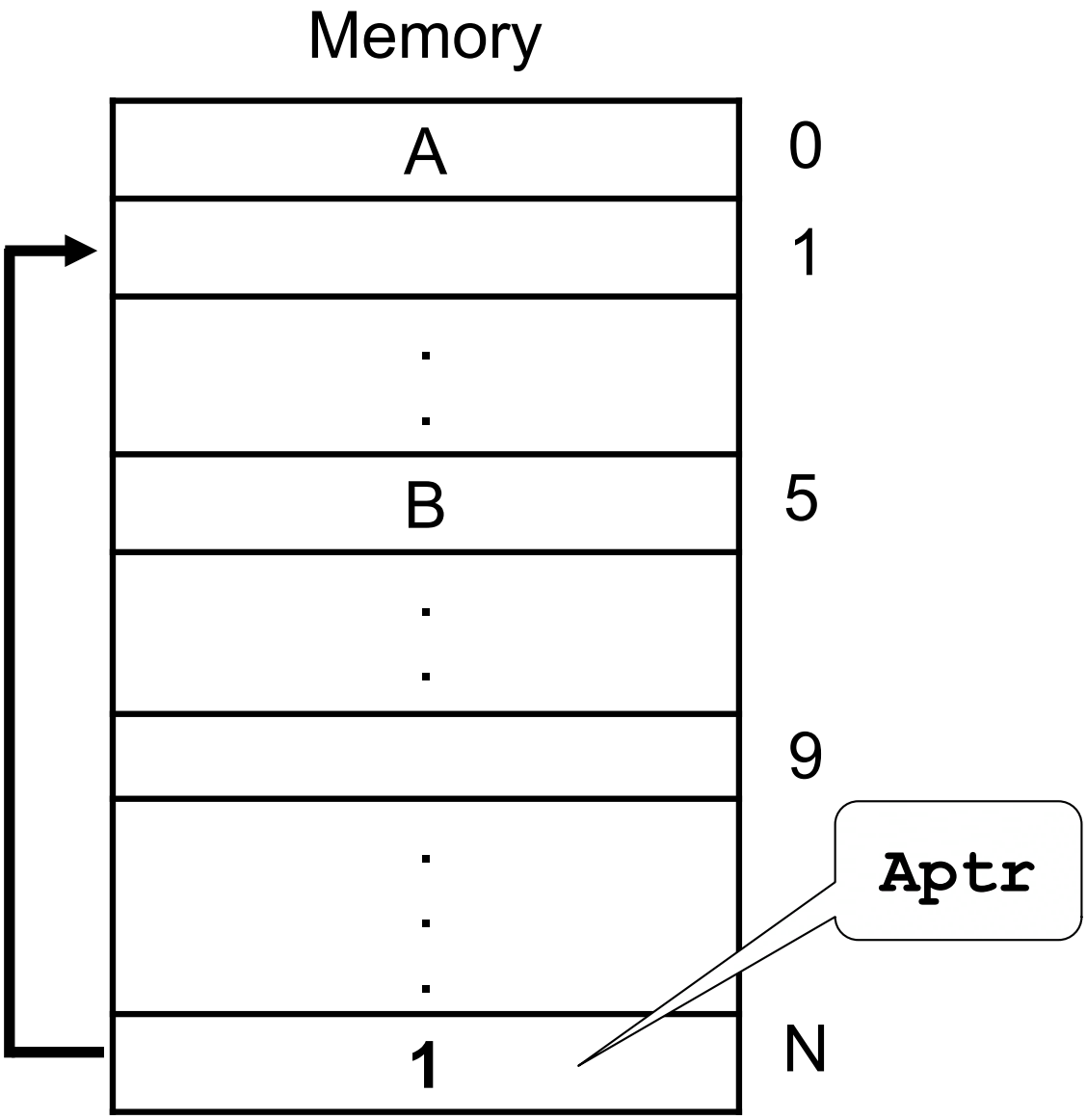
Declare and Scan Arrays with Pointers

```
char *Aptr = NULL;;
Aptr    = (char*)malloc(10*sizeof(char));
*Aptr   = 'A';
 Aptr   = Aptr + 5;
*Aptr   = 'B';
 Aptr   = Aptr - 4;
*Aptr   = 'C';
*Aptr++ = 'D';
    .
    .
    .
```

Declare and Scan Arrays with Pointers

Current Statement:

```
Aptr = Aptr - 4;
```



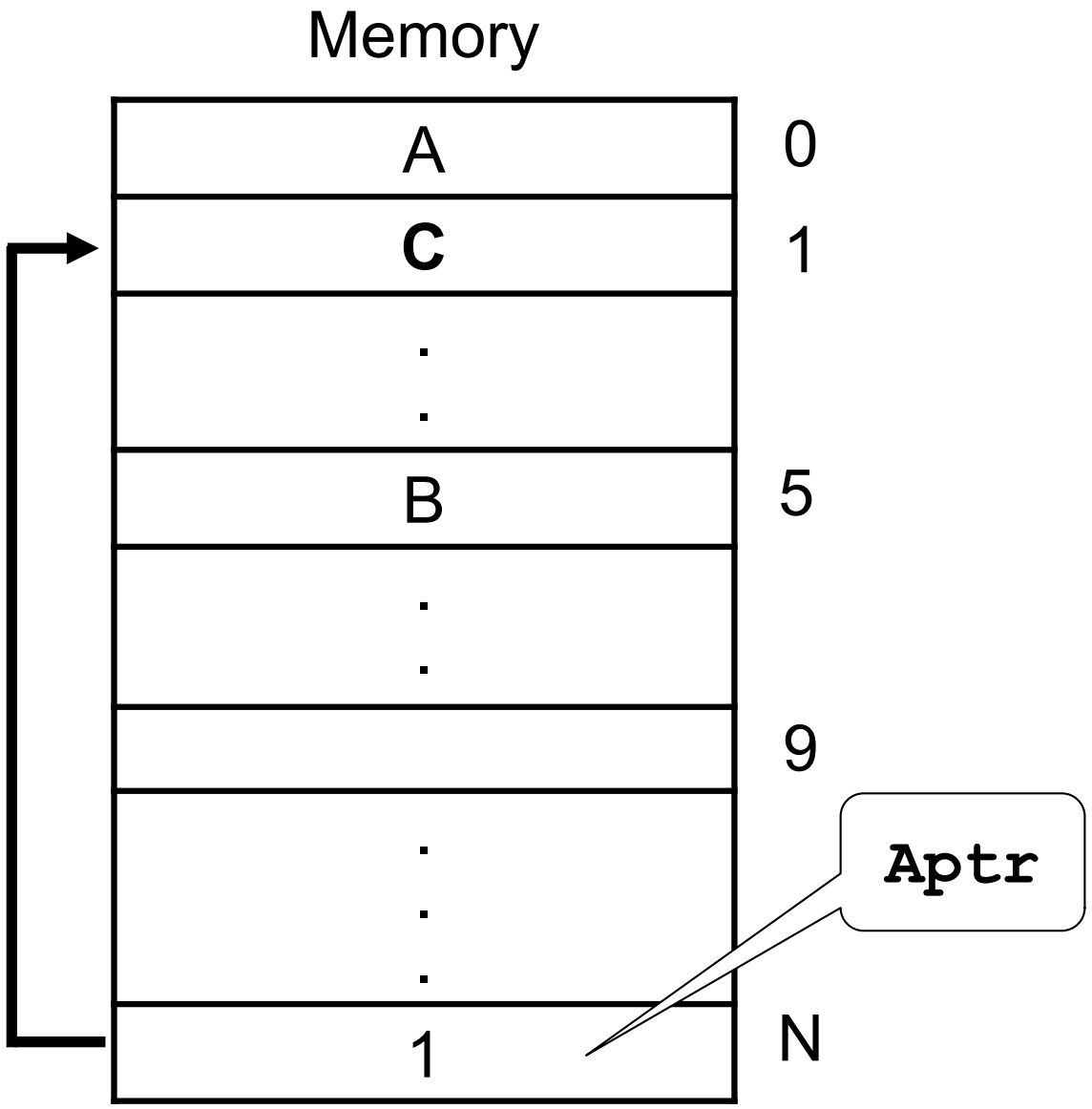
Declare and Scan Arrays with Pointers

```
char *Aptr = NULL;
Aptr    = (char*)malloc(10*sizeof(char));
*Aptr   = 'A';
 Aptr   = Aptr + 5;
*Aptr   = 'B';
 Aptr   = Aptr - 4;
*Aptr   = 'C';
*Aptr++ = 'D';
    .
    .
    .
```

Declare and Scan Arrays with Pointers

Current Statement:

```
*Aptr = 'C';
```



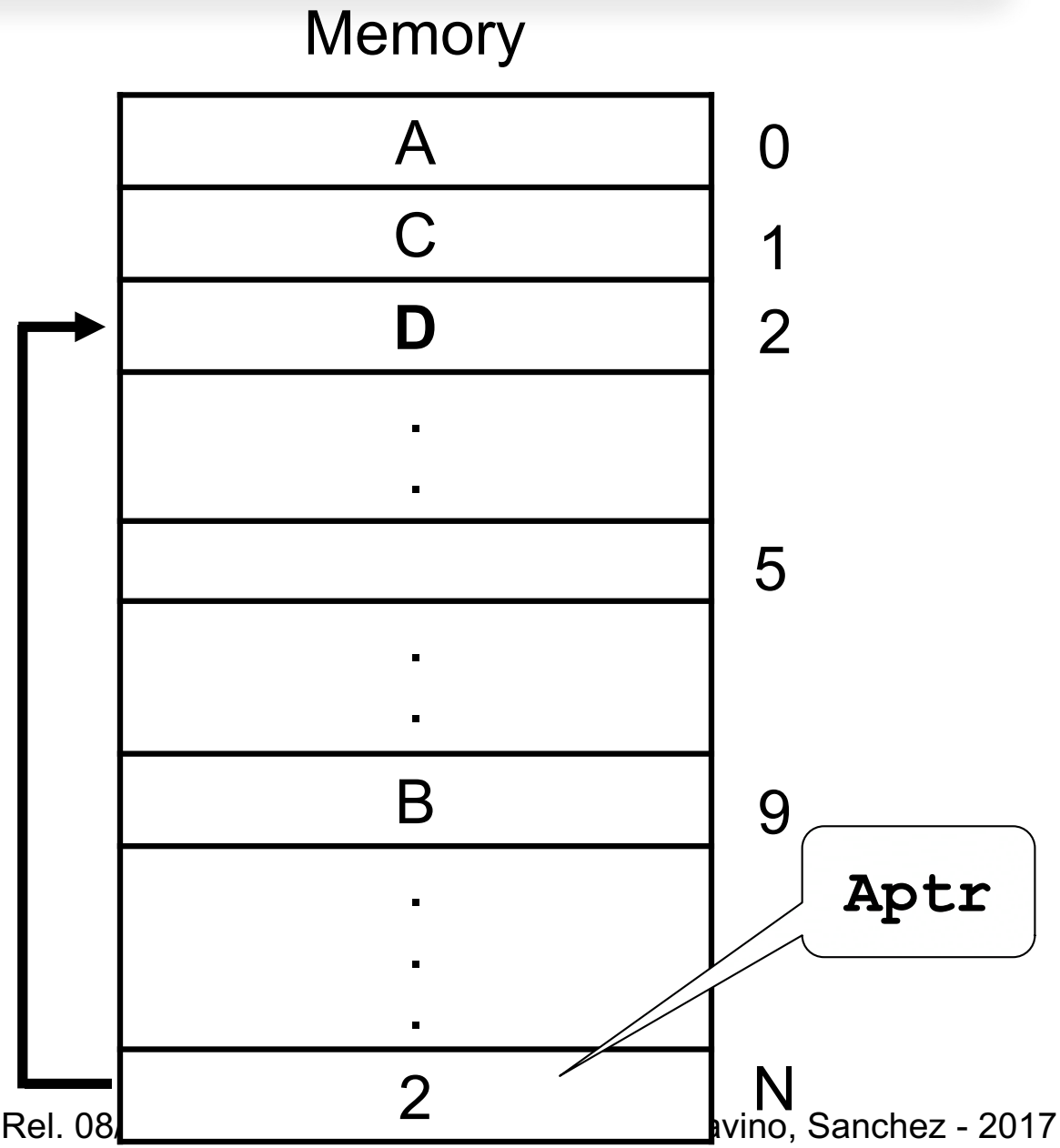
Declare and Scan Arrays with Pointers

```
char *Aptr = NULL;
Aptr    = (char*)malloc(10*sizeof(char));
*Aptr   = 'A';
 Aptr   = Aptr + 5;
*Aptr   = 'B';
 Aptr   = Aptr - 4;
*Aptr   = 'C';
*Aptr++ = 'D';
    .
    .
    .
```

Declare and Scan Arrays with Pointers

Current Statement:

```
*Aptr++ = 'D';
```



Outline

- **Pointers Usage:**
  - **Pointers and functions**
  - **Dynamic Memory**
  - **Declare and Scan Arrays**
  - **Dynamic Memory C++ style**

Dynamic Memory C++ style

- **Dynamic memory management in C++ is performed with the operators `new` and `delete`.**

```
int * intPtr = new int;
Vehicle * CarPtr = new Vehicle;
```

`new` operator:

- **allocates an appropriate memory portion of bytes according to the involved type or object**
- **activates the class constructor**
- **returns the pointer to the beginning of the allocated space or the new object**

Dynamic Memory C++ style

- **Dynamic memory management in C++ is performed with the operators `new` and `delete`.**

```
int * intPtr = new int(42);
Vehicle * CarPtr = new Vehicle(5,60,9);
```

`new` operator:

- **initializes a variable or object accordingly.**

Dynamic Memory C++ style

- **Dynamic memory management in C++ is performed with the operators `new` and `delete`.**

```
delete intPtr;
delete CarPtr;
```

`delete` operator:

- **destroys the allocated memory and frees the space.**

Rel. 08/04/2016

Dynamic Memory C++ style

- **Dynamically allocating arrays**

```
int * vectorPtr = new int[10];
delete [] vectorPtr;
```


Rel. 08/04/2016

Dynamic Memory C++ style

- **Dynamic allocation within classes?**

```
class VectorInt {
...
private:
  int * vectorPtr;
...
}
```

Dynamic Memory C++ style

- **Dynamic allocation within classes?**

```
VectorInt::VectorInt() {
    int * vectorPtr = new int[10];
}

VectorInt::~VectorInt() {
  delete [] vectorPtr;
}
```

Dynamic Memory C++ style

- **Dynamic allocation within classes?**

```
Ve        VectorInt() {

}

Vect
    c
}
```

Do you see any forthcoming issues?

Rel. 08/04/2016

Dynamic Memory C++ style

- **Dynamic allocation within classes?**

```
VectorInt::VectorInt() {
    int * vectorPtr = new int[10];
}

VectorInt::~Ve      Int() {
  delete [] v       tr;
}
```

What if... the new operator is not able to allocate the memory?

Rel. 08/04/2016

Dynamic Memory C++ style

- **Dynamic allocation within classes?**

```
VectorInt::VectorInt() {
    int * vectorPtr = new int[10];
}

VectorInt::~Ve      Int() {
   delete [] v      tr;
}
```

Then try to limit usage of dynamic memory within classes and prefer it only in the program...


Малые Автюхи, Калинковичский район, Республики Беларусь