Pointers and Arrays: 1-D Arrays

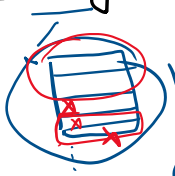
Similarities and difference (Pointers and Arrays)

Dynamic Arrays 1-D Array (Run time)

→ Allocation → take size as input from user / read it from a file.
→ Deallocation
→ Processing → (1) Data (Same as Static Arrays)
(2) Resize whole Array.

```
int size = 5;
int * arr = new int[size];
delete &arr[4];
delete [] arr;
```

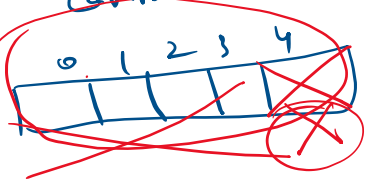
size = size - 1;
size = 4



block
Allocated for array cannot be changed (larger/extend shrink it)

life time

20 bytes = 5×4



4 ✗ size

1) if we change size variable array's size will also change.

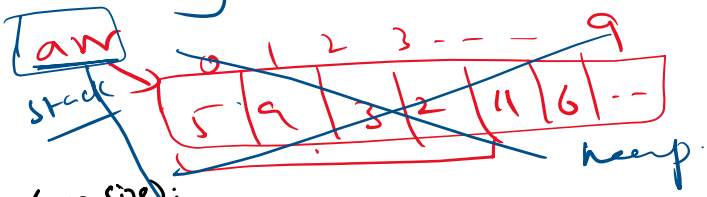
2) deallocate any element

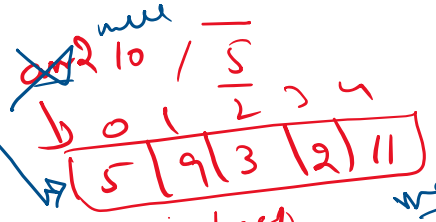
Resize.

1) Create new array of (smaller or larger size)
2) copy data from old array to new one element by element
3) deallocate original old array.
4) update the pointer variable.
5) update the size

Shrink half

```
{ int size = 10;
  int * arr = new int[size];
  for(int i=0; i<size; i++)
    cin>>arr[i];
  Shrink(arr, size);
}
void Shrink(int*& arr, int& size)
{
  (1) int * arr2 = new int[size/2];
  (2) for(int i=0; i<size/2; i++)
        arr2[i] = arr[i];   <- copy
  (3) delete [] arr;
  (4) arr = arr2;
      arr2 = nullptr;
  (5) size = size/2;
}
```





memory leak

```
arr = arr2;
delete [] arr;
```

function and dynamic 1D Arrays → [exist between function
→ pass to functions (reference, value) (pointer)
→ can return Address of array from function. (base)
→ Data is always passed by reference through pointers.

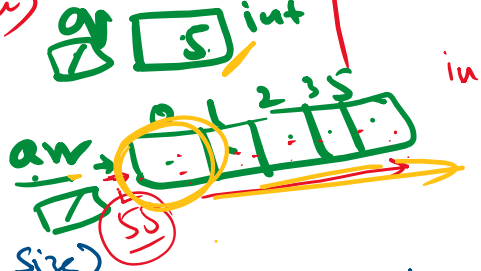
Single variable

Array → Static, dynamic

→ Allocate — pass by reference
→ Input — by value
→ print Data — by value, const
→ processing — by value (when to use const)
→ deallocation — by value (reference)
→ Extend — reference
→ Shrink — ref rules

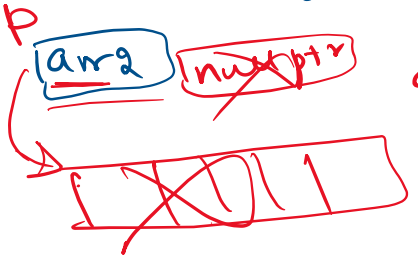
```
void fun(int *p);
void fun2(int *&p);
```

```
int main()
{
  int *qr = new int(5);
  fun(qr); fun2(qr);
  int *arr = new int[*qr];
  fun(arr);
  fun2(arr);
```




```
void allocate(int *&p, int size)
{
  p = new int[size];
}
```


```
int *arr2 = nullptr;
allocate(arr2, 10);
```



arr2 = allocate(5);

```
int * allocate (int size)
{
  int *p;
  p = new int[size];
  return p;
}
```

int *p = return and remove odd (arr2, 10);

```
void Input(int *p, int size)
{
  for(int i=0; i<size; i++)
    cin>>p[i];
}
```

read, write data

```
void print(const int *p, int size)
{
  for loop
    cout<<p[i];
}
```

read data

delete [] p;

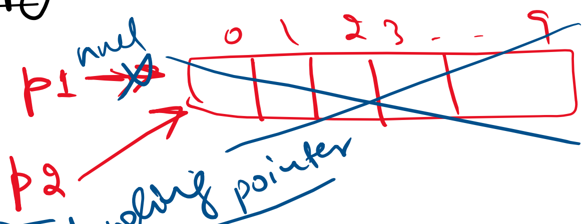
```
void deallocation(int *&p)
{
  delete [] p;
  p = nullptr;
}
```

→ deallocation(arr2);

free

Shallow vs. deep copy.

```
int *p1, int *p2;
p1 = new int [10];
p2 = p1;
delete [] p1;
p1 = nullptr;
p2[5] = 10;
```



Shallow: Two or more pointers point to the same memory. (Address)

(function calls) by value

deep copy → copy { extend / shrink }: pointer point to their own memory

1. new memory for other pointer
2. elements copy.