2 Dynamic Arrays (also see Pointers)

A dynamic array is an array whose size is determined when the program is running, not when you write the program

- Static arrays require that the programmer determine the size of the array when the program is written
  - What if the programmer estimates too large?
    - Memory is wasted
  - What if the programmer estimates too small?
    - The program may not work in some situations
- Dynamic arrays can be created with just the right size while the program is running
- Dynamic arrays are created using the **new** operator
  - Example: To create an array of 10 elements of type double:

```
double *d;
d = new double[10];
```

  - or

```
double *d=new double[10];
```

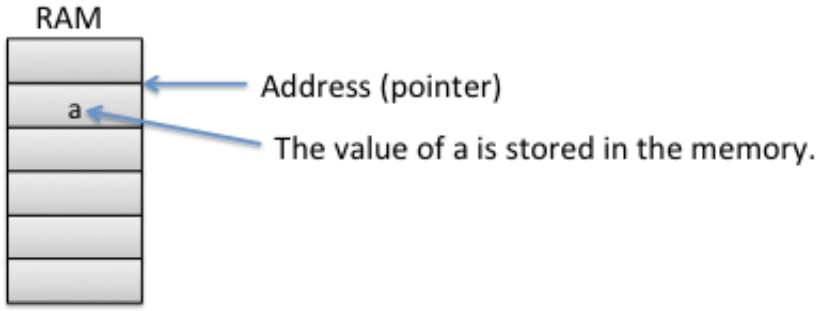
  - d can now be used as if it were an ordinary array!
- Pointer variable d is a pointer to d[0]
- When finished with the array, it should be deleted to return memory to the free-store
  - Example:

```
delete [] d;
```

  - The brackets tell C++ a dynamic array is being deleted so it must check the size to know how many indexed variables to remove
    - Forgetting the brackets, is not legal, but would tell the computer to remove only one variable.

2.1 Pointers

A pointer is the memory address of a variable.



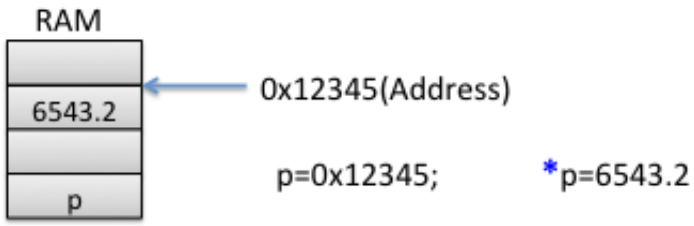
Pointer Variable has an address in memory and can be used to refer the value stored in the memory address.

2.1.1 Declaring Pointers

- Pointer variables must be declared to have a pointer type
  - Example: To declare a pointer variable p that can "point" to a variable of type double:

```
double *p;
```

  - The asterisk identifies p as a pointer variable



2.1.2 The address of Operator (& ampersand)

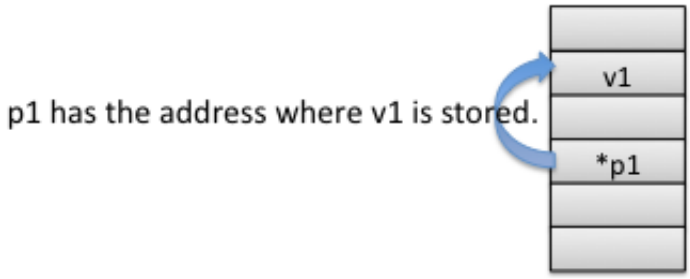
- The **&** operator can be used to determine the address of a variable which can be assigned to a pointer variable
  - Example:

```
p1 = &v1;
```

  - **p1** is now a pointer to **v1**
  - **v1** can be called **v1** or "the variable pointed to by **p1**"

```
v1 = 0;
p1 = &v1;
*p1 = 42;
cout << v1 << endl;
cout << *p1 << endl;
```

Output:
42
42



Operation to *p1 may modify the content in the address p1 points.

2.1.3 "new" operator

- Using pointers, variables can be manipulated even if there is no identifier for them
  - To create a pointer to a new "nameless" variable of type int:

```
p1 = new int;
```

- The new variable is referred to as *p1
- *p1 can be used anyplace an integer variable can

```
cin >> *p1;
*p1 = *p1 + 7;
```

2.1.4 "delete" Operator

- When dynamic variables are no longer needed, delete them to return memory to the free-store
  - Example:

```
delete p1;
```

- The value of p is now undefined and the memory used by the variable that p pointed to is back in the free-store

2.2 Returning An Array from function

- If function takes array

```
void function(int score[], int size)
{
        for (int i = 0 ; i < size ; i++)
                score[i] = 1;
}
```

  - The values assigned in the function body return to upper routine.

- This doesn't work:

```
int *function()
{
        int score[5];
        for (int i = 0 ; i < 5 ; i++)
                score[i] = 1;
        return score;
}
```

  - The local array variable "score" will be deleted after program exits from the function.

- This works:

```
int *function()
{
        int *score = new int[5];
        for (int i = 0 ; i < 5 ; i++)
                score[i] = 1;
        return score;
}
```

  - It is your responsibility to delete the memory block when it is no longer used.