Input Buffer

- For the user input `cin>>anInteger;` followed by `cin>>str;` on the next line, then the enter key pressed after typing the number is saved as input buffer and later accepted by `cin>>str;` and to avoid this use **`cin.ignore();`** between `cin>>anInteger;` and `cin>>str;`

Enum

- User defined data types such that we can link a commonly used term with a constant integer:
  `enum day {mon,tue,wed,thr,fri,sat,sun};` creates
  0: mon
  1: tue
  2: wed
  3: thr
  4: fri
  5: sat
  6: sun
  which is same as
  `const int mon = 0;`
  `const int tue = 1;`
  `const int wed = 2;`
  `const int thr = 3;`
  `const int fri = 4;`
  `const int sat = 5;`
  `const int sun = 6;`
- day is a user defined data type which can only have names mon,tue,wed,thr,fri,sat,sun automatically it gets a value
  `day d; *`d = jan; // error`* `d = tue;
  `cout << d; // 1`
  `enum day {mon=4,tue,wed,thr,fri,sat,sun};`
  mon =4 tue=5 wed=6 thr=7 fri=8 sat=9 sun=10
  `enum day {mon=4,tue,wed,thr,fri=13,sat,sun};`
  mon=4 tue=5 wed=6 thr=7 fri=13 sat=14 sun=15
  `enum day {mon=4,tue,wed,thr,fri=6,sat,sun};`
  mon=4 tue=5 wed=6 thr=7 fri=6 sat=7 sun=8

Type Definition

- Used to make variables more readable.

- So lets say a school app has `int m1,m2,m3,r1,r2,r3` where mX are marks in subject X and rN is the roll number N
- To make it more readable we use typedef to alias the data type of mX from int to marks and rN from int to roll.
  `typedef int marks;`
  `typedef int roll;`
  `marks m1, m2, m3;`
  `roll r1, r2, r3;`

For Each Loop

- let `int A[] = {1, 2, 3, 4, 5};` then
- `for (int x : A)`
  `cout << x;`
- `for (auto x : A)` where the compiler takes care of the data type of x
- changing the value of x doesn't change the array element because x is a copy
  - To avoid this you can used `int &x : A` instead of `int x : A`
  - `auto &x : A` is also allowed

Binary Search

```cpp
#include <iostream>
using namespace std;
int main()
{
    const int A[] = {4, 8, 24, 42, 101, 404, 1234};
    int key, l = 0, h = (sizeof(A) / 4) - 1;

    cout << "Enter Key: ";
    cin >> key;

    while (l <= h)
    {
        int mid = (l + h) / 2;

        if (key == A[mid])
        {
            cout << "Found at " << mid;
            return 0;
        }
        else if (key < A[mid])
            h = mid - 1;
        else
            l = mid + 1;
    }

    cout << "Not Found";
    return -1;
}
```

Finding Min and Max of an Array

```
#include <iostream>
#include <climits>
using namespace std;
int main()
{
    const int A[] = {4, 8, 24, 42, 101, 404, 1234};
    int min = INT_MAX, max = INT_MIN;

    for (auto x : A)
    {
        min = x < min ? x : min;
        max = x > max ? x : max;
    }

    cout << "Min = " << min << endl;
    cout << "Max = " << max;

    return 0;
}
```

Note on Arrays

- When an Array of length n is created and when m elements are hard coded then remaining n-m elements are automatically initialized to 0.

- When a 2-D Array is created, all the elements are contiguous in the memory just like a 1-D array.

- Need to use reference of x when A is 2-D array in a for each loop but you still need to use nested for each loop.

  ```
  #include <iostream>
  using namespace std;
  int main()
  {
      const int A[2][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}};

  ```

```
    for (auto &i : A)
    {
        for (auto j : i)
            cout << j << " ";
        cout << endl;
    }

    return 0;
}
```

- `const int A[][] = {{1, 2, 3, 4}, {5, 6, 7, 8}};` is invalid

Pointer

- Memory Layout
  - | HEAP |

  - | STACK | ← declarations like `int i = 0` are stored in STACK. Students heavily use it. Automatically deleted when out of scope.

  - | CODE | ← is the read-only section of the memory where the code is loaded after launching the program. No external programs can modify this section.

    - The CODE section can access STACK and itself. Not the HEAP.

    - To access the HEAP from the CODE section you need to create a pointer to a memory address in HEAP and the pointer is created in the STACK from the CODE section.
      - Thus HEAP can only be accessed using pointers.
      - Accessing Files and hardware devices is also done using pointers.

    - Accessing HEAP using `new`
      - example: `int *p = new int[5]`
      - If not freed at the end, we get a Memory Leak. Use `delete[] p` and then `p = nullptr`
        - if you do `p = nullptr` first then you won't be able to free HEAP later.

- once an array is declared in the STACK you cannot change its size but it is possible if it is in the HEAP.

```
#include <iostream>
using namespace std;
int main()
{
    int *p = new int[5];

    for (int i = 0; i < 5; i++)
    {
        p[i] = i + 1;
        cout << p[i] << " ";
    }

        delete[] p; // to avoid leaking of {1, 2, 3, 4, 5}
        p = new int[10]; // new memory gets allocated
pointing to {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

    delete[] p;
    p = nullptr;

    return 0;
}
```

- Pointer Arithmetic
  - dereferencing is not needed.

  - subtracting 2 pointers pointing to different indices of an integer Array can be divided by 4 to get how many indices far are the 2 pointers.

  - this is

```
#include <iostream>
using namespace std;
int main()
{
    int A[] = {1, 2, 3, 4, 5};
    int *p = A; // Same as int *p = &A[0]

    for (int i = 0; i < sizeof(A) / 4; i++, p++)
        cout << *p;
    return 0;
}
```

- same as

```cpp
#include <iostream>
using namespace std;
int main()
{
    int A[] = {1, 2, 3, 4, 5};

    for (int i = 0; i < sizeof(A) / 4; i++)
        cout << *(A + i);

    return 0;
}
```