References –

```
int main()
{
  int i=10;
  int &j=i;
  i++;
  cout<<j<<endl;
  j++;
  cout<<i<<endl;
  int k=100;
  j=k;
  cout<<i<<endl;
}
```

Should be done when the variable is declared.

Output

11

12

100

```
int &j;
j=100;
```

→ Gives error as j should be initialized with some address at the time when it was created.

```
void increment(int &n)
{
  n++;
}

int main()
{
  int n=10;
  n++;
  increment(n);
  cout<<n;
}
```

Pass by reference.

Output

12

```
void increment(int n)
{
  n++;
}
```

Pass by value

For same main f^n.

Output

11

Some bad practices -

```
int &f(int n)
{
  int a = n;
  return &a;  → returns address of local variable.
}

int *f2(){
  int i = 10;
  return &i;
}

int main(){
  int *p = f2();
  int &k = f(10);
  k++;
  (*p) = 100;
}
```

→ Here, we are trying to access the memory which is already cleared as soon as the function scope is done. Hence, trying to change the value pointed by that address is a bad practice.

Dynamic Memory Allocation

```
int *p = new int[50];
```

- `p` → 8 bytes
- `new int[50]` → 50 × 4 = 200 bytes

```
delete p;
```

↳ deletes the memory created in heap (200 bytes gets cleared)

```
int main()
{
  int *p = new int;
  delete p;
  p = new int;
  delete p;
}
```

Deleting arrays created dynamically:

```
int *arr = new int[100];
delete [] arr;
```