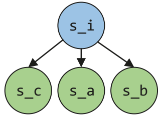
a) **Original C Code**

```
int i = 0;
while (i < N) {
  c[i] = a[i] + b[i];
  i++;
}
```

Stream Decoupled Pseudo Code

```
stream_cfg(s_i, s_a, s_b, s_c);
while (s_i < N) {
  s_c = s_a + s_b;
  stream_step(s_i);
}
stream_end(s_i, s_a, s_b, s_c);
```

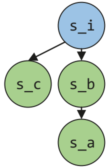
Stream Dependency Graph



b)

```
int i = 0;
while (i < N) {
  c[i] = a[b[i]];
  i++;
}
```

```
stream_cfg(s_i, s_a, s_b, s_c);
while (s_i < N) {
  s_c = s_a;
  stream_step(s_i);
}
stream_end(s_i, s_a, s_b, s_c);
```



Legend
i,   s_i, ●: induction variable stream
a[i], s_a, ●: memory stream

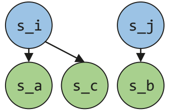
c) **Original C Code**

```
int i = 0, j = 0, v = 0;
while (i < N && j < N) {
  if (a[i] < b[j]) {
    v += c[i];
    i++;
  } else {
    j++;
  }
}
```

Stream Decoupled Pseudo Code

```
stream_cfg(s_i, s_a, s_c, s_j, s_b);
while (s_i < N && s_j < N) {
  if (s_a < s_b) {
    v += s_c;
    stream_step(s_i);
  } else {
    stream_step(s_j);
  }
}
stream_end(s_i, s_a, s_c, s_j, s_b);
```

Stream Dependency Graph



d)

```
int i = 0;
while (i < N) {
  b[i] =    a[i].x
         + a[i].y;
  i++;
}
```

```
stream_cfg(s_i, s_a, s_b);
while (s_i < N) {
  s_b = s_a.x + s_a.y;
  stream_step(s_i);
}
stream_end(s_i, s_a, s_b);
```