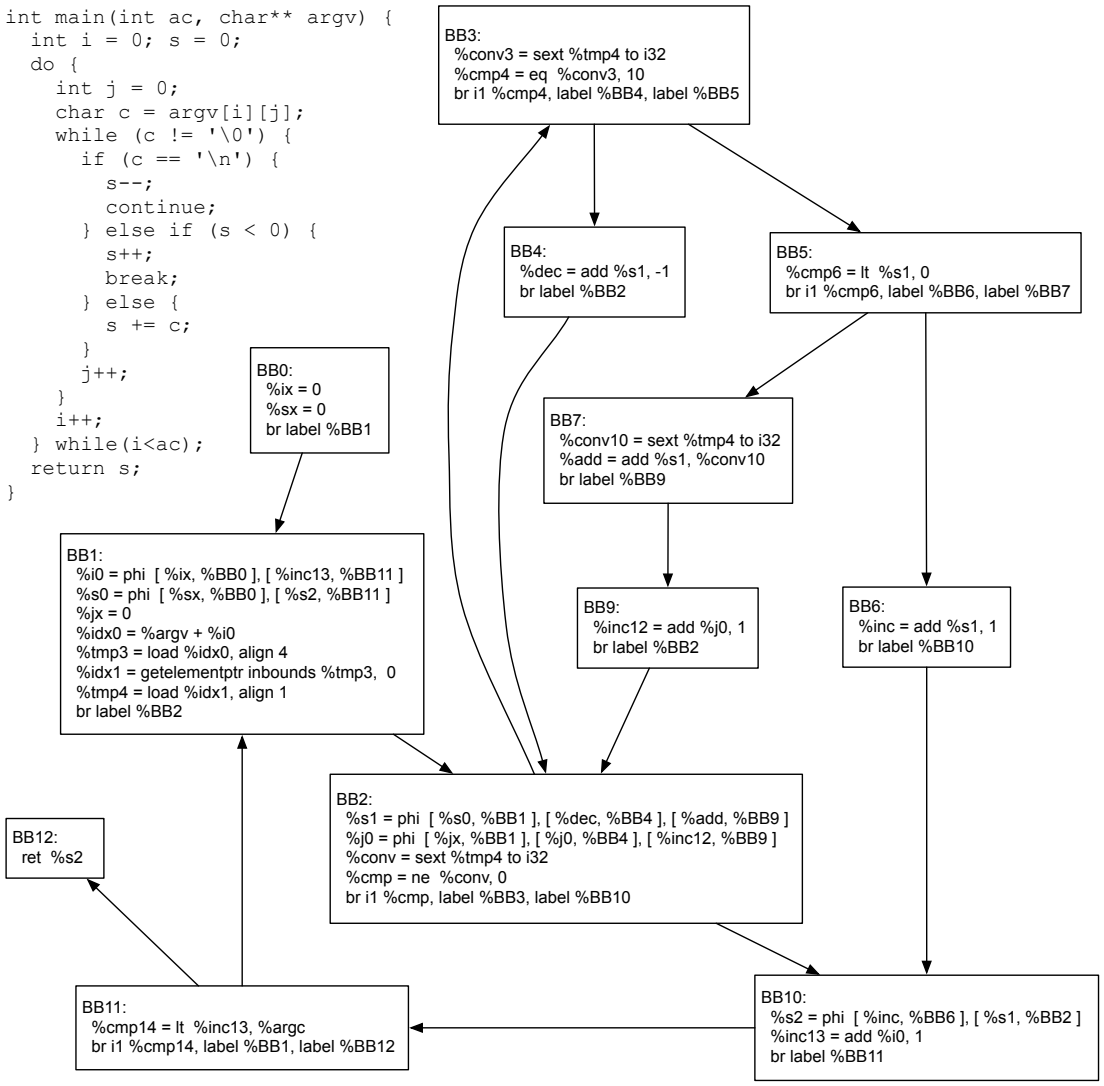
DCC888 – Program Slicing

Name: _____ ID: _____

1. Below we show a program, written in C, and its control flow graph, as produced by LLVM [1]. Draw the dependence graph for the program below. Use the instructions that we show in its control flow graph. Use solid edges to mark **data dependences** and use dashed edges to mark **control dependences**.

```
int main(int ac, char** argv) {
  int i = 0; s = 0;
  do {
    int j = 0;
    char c = argv[i][j];
    while (c != '\0') {
      if (c == '\n') {
        s--;
        continue;
      } else if (s < 0) {
        s++;
        break;
      } else {
        s += c;
      }
      j++;
    }
    i++;
  } while(i<ac);
  return s;
}
```

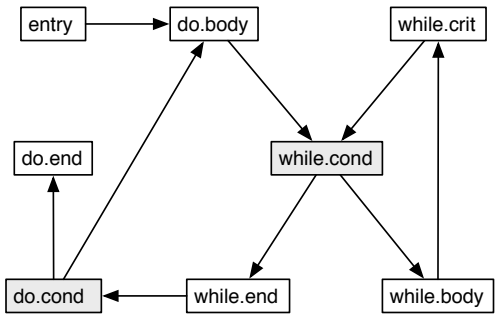


[1] For the sake of readability, we have simplified the control flow graph

2. Below we show several programs, and their control flow graphs. For each branch, in each control flow graph, mark this branch with an H if its influence region determines a hammock graph, and with an N, if it does not.

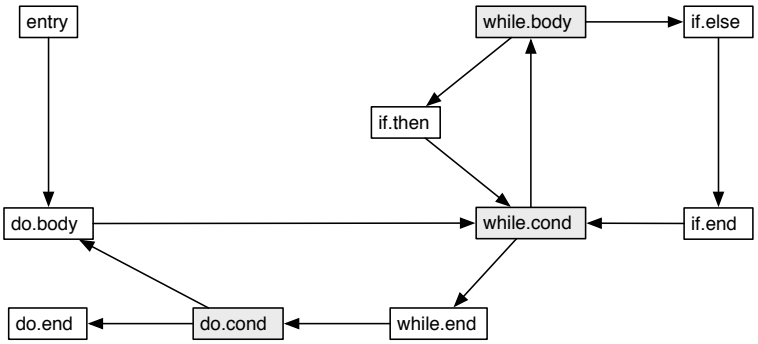
```
int main(int argc, char** argv) {
 int i = 0;
 int s = 0;
 do {
  char* p = argv[i];
  while (*p != '\0') {
   s += *p;
  }
  i++;
 } while(i < argc);
 return s;
}
```

(a)



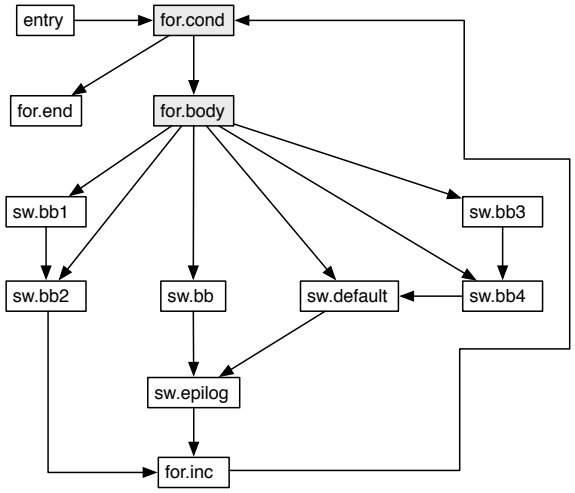

```
int main(int argc, char** argv) {
 int i = 0;
 int s = 0;
 do {
  char* p = argv[i];
  while (*p != '\0') {
   if (*p == '\n')
    continue;
   else
    s += *p;
  }
  i++;
 } while(i < argc);
 return s;
}
```

(b)



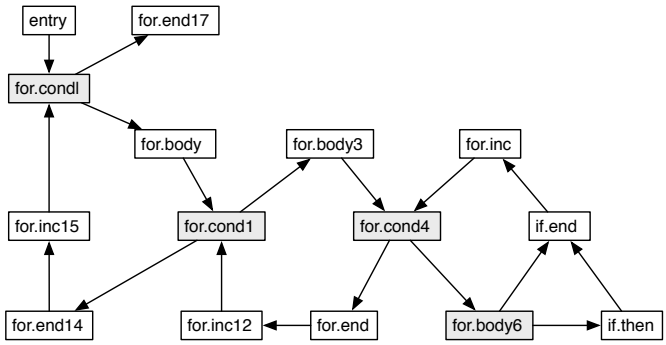

```
int main(int argc, char **argv) {
 int s = 0;
 int i = 0;
 for (; i < argc; i++) {
  switch(i) {
   case 1: s = 2; break;
   case 2: s = 3;
   case 3: s = 5; continue;
   case 4: s = 7;
   case 5: s = 11;
   default: s = 13;
  }
 }
 printf("%d\n", s);
 return 0;
}
```

(c)



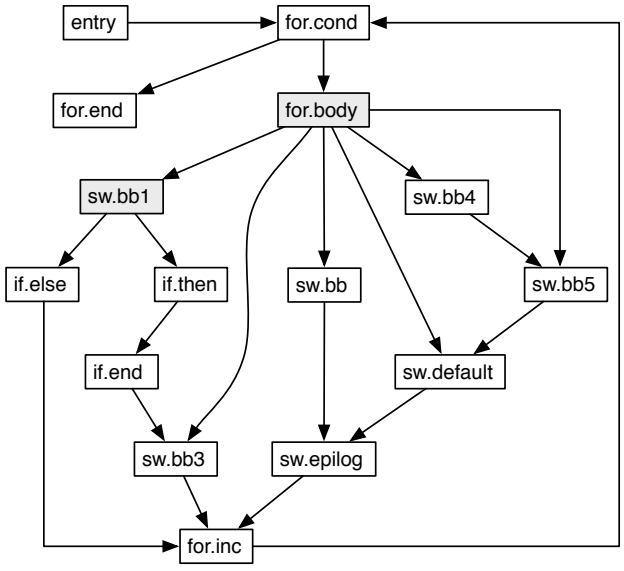
(d)

```
int funcA() {
  int i, j, k, t = 0;
  for(i = 0; i < 100; i++)
    for(j = 0; j < 10; j++)
      for(k = 0; k < 100; k++)
        if(i * i + j * j + k * k % 7 == 0)
          t++;
  printf("%d\n", t);
  return 0;
}
```



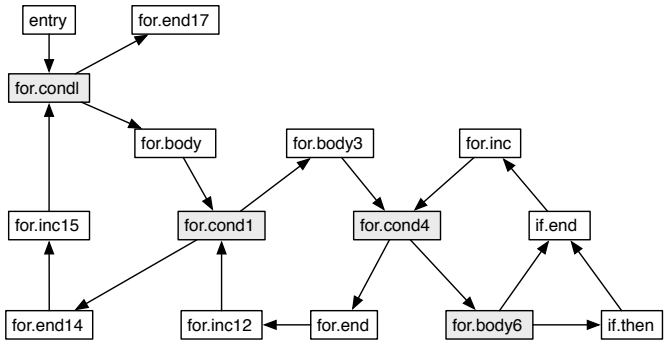
(e)

```
int main(int argc, char **argv) {
  int s = 0;
  int i = 0;
  for (; i < argc; i++) {
    switch(i) {
      case 1: s = 2; break;
      case 2: s = 3;
              if (argc > i * i)
                s++;
              else
                continue;
      case 3: s = 5; continue;
      case 4: s = 7;
      case 5: s = 11;
      default: s = 13;
    }
  }
  printf("%d\n", s);
  return 0;
}
```



(f)

```
int funcA() {
  int i, j, k, t = 0;
  for(i = 0; i < 100; i++)
    for(j = 0; j < 10; j++)
      for(k = 0; k < 100; k++)
        if(i * i + j * j + k * k % 7 == 0)
          t++;
  printf("%d\n", t);
  return 0;
}
```



3. Below we show a program, written in C, that performs some memory accesses. The rest of this question is about this program.

```
1.    void read_matrix(int* data, char w, char h) {
2.      char buf_size = w * h;
3.      if (buf_size < BUF_SIZE) {
4.        int c0, c1;
5.        int buf[BUF_SIZE];
6.        for (c0 = 0; c0 < h; c0++) {
7.          for (c1 = 0; c1 < w; c1++) {
8.            int index = c0 * w + c1;
9.            buf[index] = data[index];
10.         }
11.       }
12.       process(buf);
13.     }
14.   }
```

(a) Assume that we have `h = 6` and `w = 22` in line 2 of our program. What will be the value of `buf_size` in this case? Explain your answer.

(b) If we have `h = 6` and `w = 22`, then we might get an invalid memory write at line 9 of our example. Why?

(c) Imagine that we want to sanitizer this program against arithmetic overflows. In this case, we want to instrument every arithmetic operation that interferes with a memory accesses. The only memory access that we have in this program happens at line 9. If we slice this program given line 9 as the criterion, would we be able to remove any part of it away?