Programming Design In-class Practices

Pointers

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Problem 1: See memory allocation

- Consider the following program:

- Modify this program to see (at least on your computer):
  - How many bytes of memory space are consumed?
  - Where are the allocated memory spaces?

```
#include<iostream>
using namespace std;

int main()
{
  int anInteger = 0;
  for(int i = 0; i < 10; i++)
    int anotherInteger = 0;
  return 0;
}
```

Problem 2: Modify a variable

- Consider the following program:

- Modify this program to:
  - Have a pointer pointing to **a**.
  - Let the user modify the value of **a** through the pointer.

```
#include<iostream>
using namespace std;

int main()
{
  int a = 0;
  cin >> a;
  cout << a << "\n";

  return 0;
}
```

Problem 3: Pass pointers into a function

- Correct the following two programs to find the maximum of **a** and **b**:

```
#include<iostream>
using namespace std;

int* maxPtr(int* a, int* b)
{
  return *a > *b ? a : b;
}

int main()
{
  int a = 0, b = 0;
  cin >> a >> b;
  cout << &maxPtr(*a, *b) << "\n";
  return 0;
}
```

```
#include<iostream>
using namespace std;

int* maxPtr(int* a, int* b)
{
  return *a > *b ? &a : &b;
}

int main()
{
  int a = 0, b = 0;
  cin >> a >> b;
  cout << maxPtr(&a, &b) << "\n";
  return 0;
}
```

Problem 4: Find an error, if any

- Find an error in the following program or conclude that there is none.
  - The program should print out the maximum of two input values.
  - **maxAddr** should return the address of the maximum value.
  - Including potential run-time error and bad implementation.

```
#include<iostream>
using namespace std;

int* max(int a, int b)
{
  int c = a;
  if(b > a)
    c = b;
  return &c;
}
```

```
int main()
{
  int a = 0, b = 0;
  cin >> a >> b;
  int* maxAddr = max(a, b);
  cout << *maxAddr << "\n";

  // many other things

  return 0;
}
```

Problem 5: Find an error, if any (2)

- Find an error in the following program or conclude that there is none.
  - The program should print out the maximum of two input values.
  - **maxAddr** should return the address of the maximum value.
  - Including potential run-time error and bad implementation.

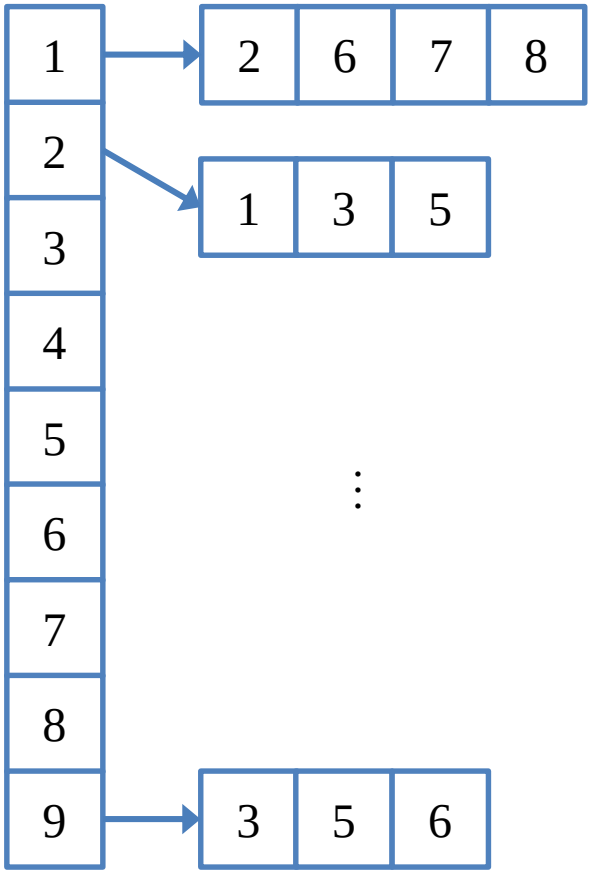
```
#include<iostream>
using namespace std;

int* max(int a, int b)
{
  int* cPtr = new int(a);
  if(b > a)
    *cPtr = b;
  return cPtr;
}
```

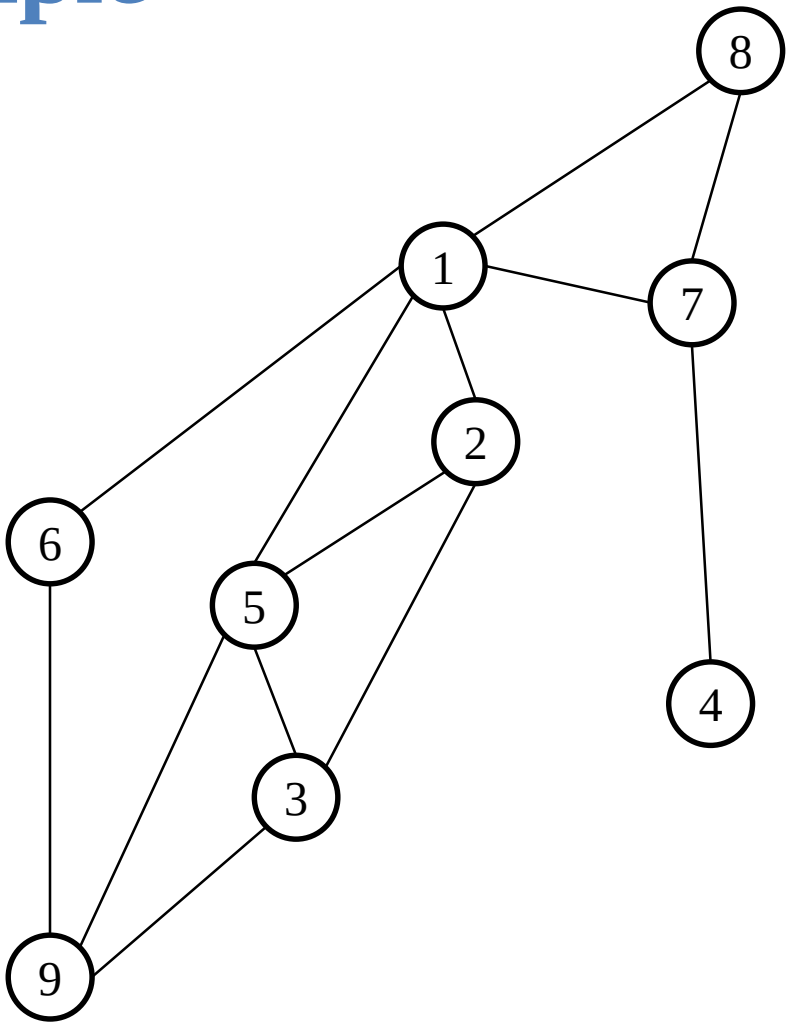
```
int main()
{
  int a = 0, b = 0;
  cin >> a >> b;
  int* maxAddr = max(a, b);
  cout << *maxAddr << "\n";

  // many other things

  return 0;
}
```

Adjacency list

- An **adjacency list** of a graph may be constructed as follows.
  - Given the number of nodes n, create a **static array** nodes of length n.
  - Each array element is an integer pointer pointing to a **dynamic array** whose length is the node degree.
  - In a node's dynamic array, each element is the index of one of its neighbor.

Adjacency list: an example





Adjacency list: implementation

```
#include<iostream>
using namespace std;

const int NODE_CNT_MAX = 100;

int inputGraphInfo(int* neighbors[], int degrees[]);
void printGraph(int* neighbors[], const int degrees[], int nodeCnt);
void releaseMemory(int* neighbors[], int nodeCnt);

int main()
{
  int* neighbors[NODE_CNT_MAX] = {0};
  int degrees[NODE_CNT_MAX] = {0};
  int nodeCnt = inputGraphInfo(neighbors, degrees);
  printGraph(neighbors, degrees, nodeCnt);
  releaseMemory(neighbors, nodeCnt);

  return 0;
}
```

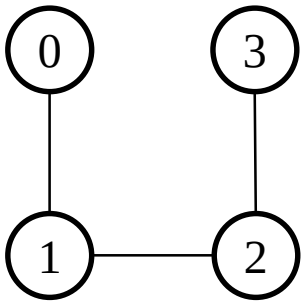
Adjacency list: implementation

```
int inputGraphInfo(int* neighbors[],
                   int degrees[])
{
  int nodeCnt = 0;
  cin >> nodeCnt;
  for(int i = 0; i < nodeCnt; i++)
  {
    // cout << "Node " << i
    //      << "\'s degree is: ";
    cin >> degrees[i];
    neighbors[i] = new int[degrees[i]];
    // cout << "Node " << i
    //      << "\'s neighbors: ";
    for(int j = 0; j < degrees[i]; j++)
      cin >> neighbors[i][j];
  }
  return nodeCnt;
}
```

```
void printGraph(int* neighbors[],
                const int degrees[],
                int nodeCnt)
{
  for(int i = 0; i < nodeCnt; i++)
  {
    // cout << "Node " << i << ": ";
    for(int j = 0; j < degrees[i]; j++)
      cout << neighbors[i][j] << " ";
    cout << "\n";
  }
}
void releaseMemory(int* neighbors[],
                   int nodeCnt)
{
  for(int i = 0; i < nodeCnt; i++)
    delete [] neighbors[i];
}
```

Problem 6: adjacency list to matrix

- Try it:



Input:

```
4
1 1
2 0 2
2 1 3
1 2
```

Output:

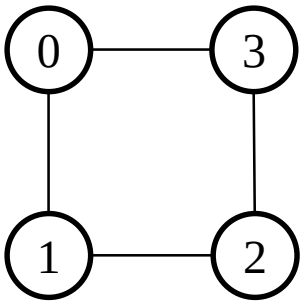
```
1
0 2
1 3
2
```

Input:

```
4
2 1 3
2 0 2
2 1 3
2 0 2
```

Output:

```
1 3
0 2
1 3
0 2
```

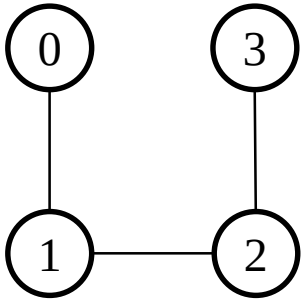


Problem 6: adjacency list to matrix

- Rewrite the function **`printGraph`** to print out the graph information in an adjacency matrix.
- All nodes are labeled as 0, 1, 2, …, $n-1$, where n is the number of nodes.
- Input:
  - Line 1 contains an integer n as the number of nodes.
  - Line $i+2$ contains an integer d_i, the degree of node i, and then d_i integers as the indices of node i's neighbors. Two consecutive values are separated by a white space.
- Output:
  - n lines in total. Line i contains $b_{i,1}$, $b_{i,2}$, …, and b_{in}, where $b_{ij}=1$ if nodes i and j are neighbors and 0 otherwise.
  - Separate two consecutive values by one white space.
  - There is no white space after the last value.

Problem 6: adjacency list to matrix

- Examples:

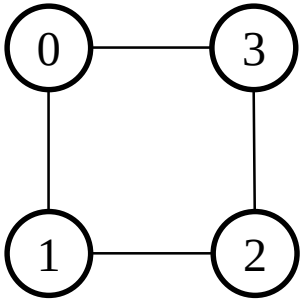


Input:
```
4
1 1
2 0 2
2 1 3
1 2
```

Output:
```
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
```

Input:
```
4
2 1 3
2 0 2
2 1 3
2 0 2
```

Output:
```
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
```



Problem 7: dynamic adjacency list

- Rewrite the previous program to allow the number of nodes to change.
  - Hint: Modify **`int* neighbors[NODE_CNT]`** to **`int** neighbors`**.