

Oregon State University

School of Electrical Engineering and Computer Science

CS 261 – Recitation 8



Spring 2016

Outline

- Heaps and Priority Queues
- Command-line arguments
- File I/O

Heaps and priority queues

- What is a priority queue?

- A **priority queue** is a collection designed to make it easy to, find the element with highest priority.

- What are the common functions that a priority queue supports?

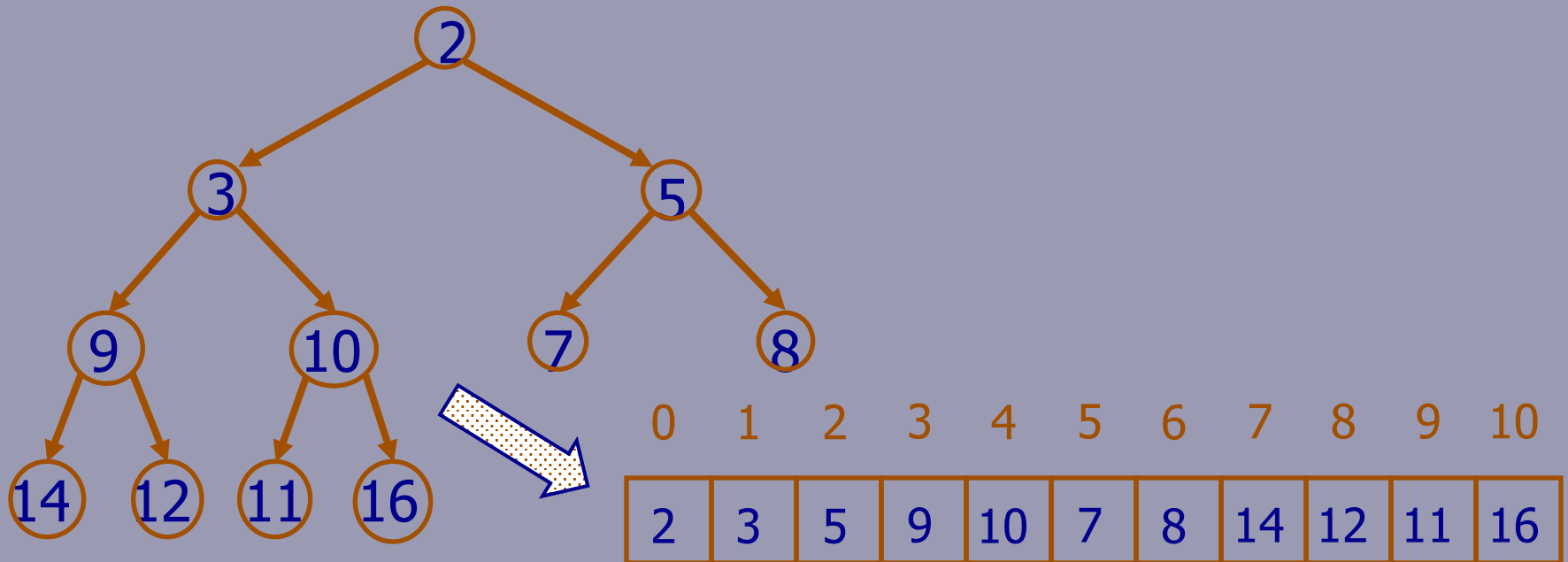
```
void add (EleType newValue);  
EleType getFirst ();  
void removeFirst ();
```

- What is a **binary heap**?

- A **complete binary tree** in which every node's value is less than or equal to the values of its children.

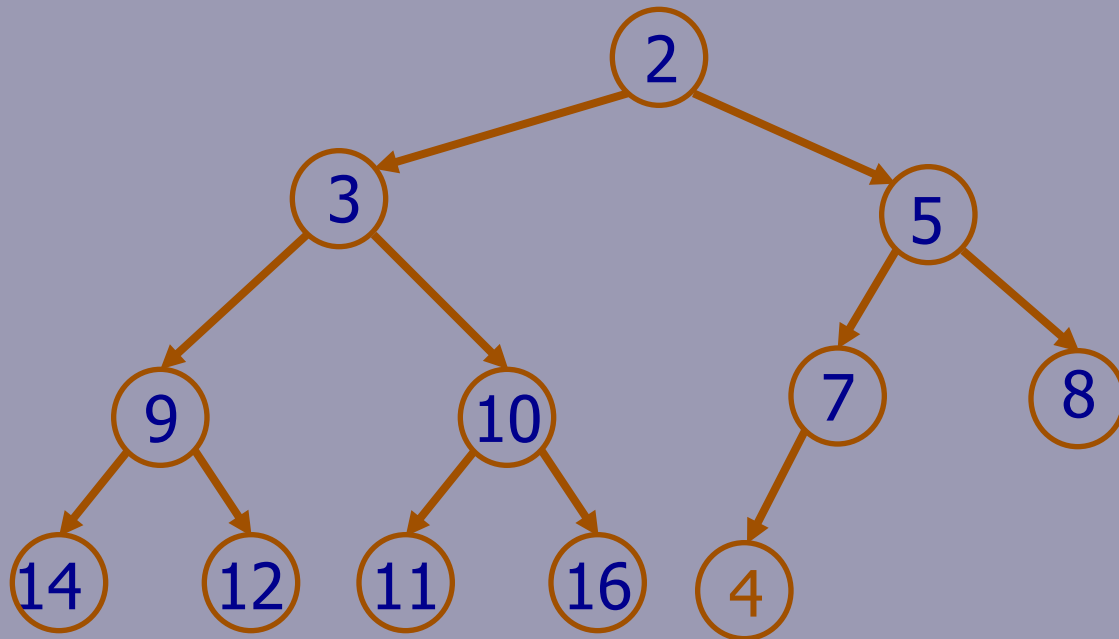
Heaps and priority queues

- How to present a binary heap?
 - Using an array (dyArray)
- Suppose the root has index 0, what are the indices of the 2 children of a node at index i ? $2 * i + 1, 2 * i + 2$
- What is the index of the parent of a node at index i ? $(i-1)/2$



Heaps and priority queues

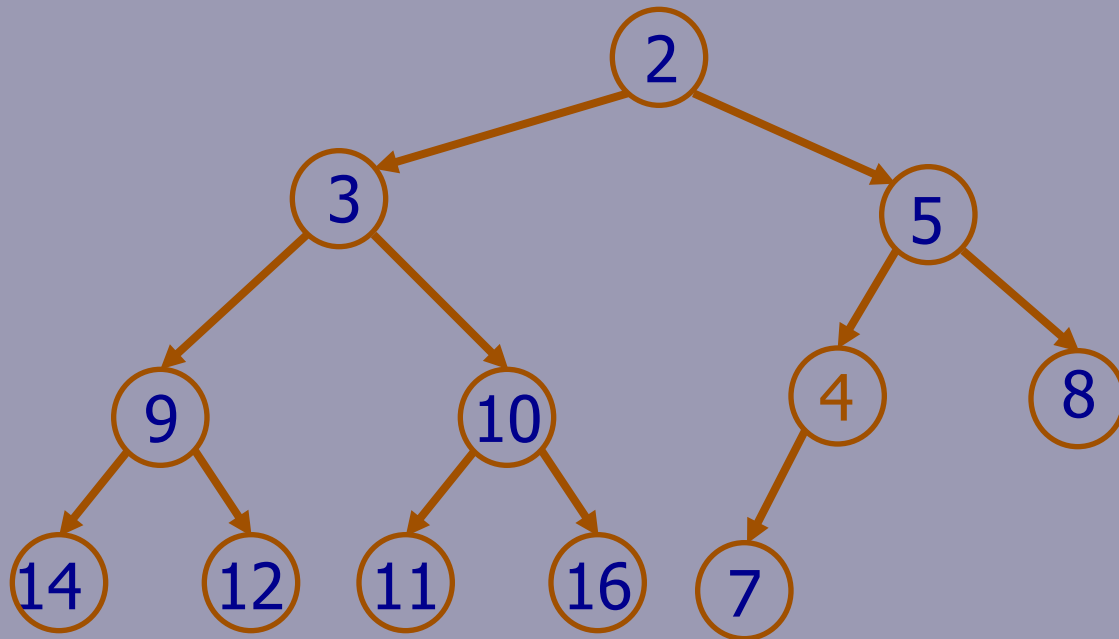
- How to get the smallest element from a binary heap?
 - Return the first element.
- How to add a new element to a binary heap?
 - Insert the new element at the end of the heap
 - Fix the heap order



Add 4 to this heap??

Heaps and priority queues

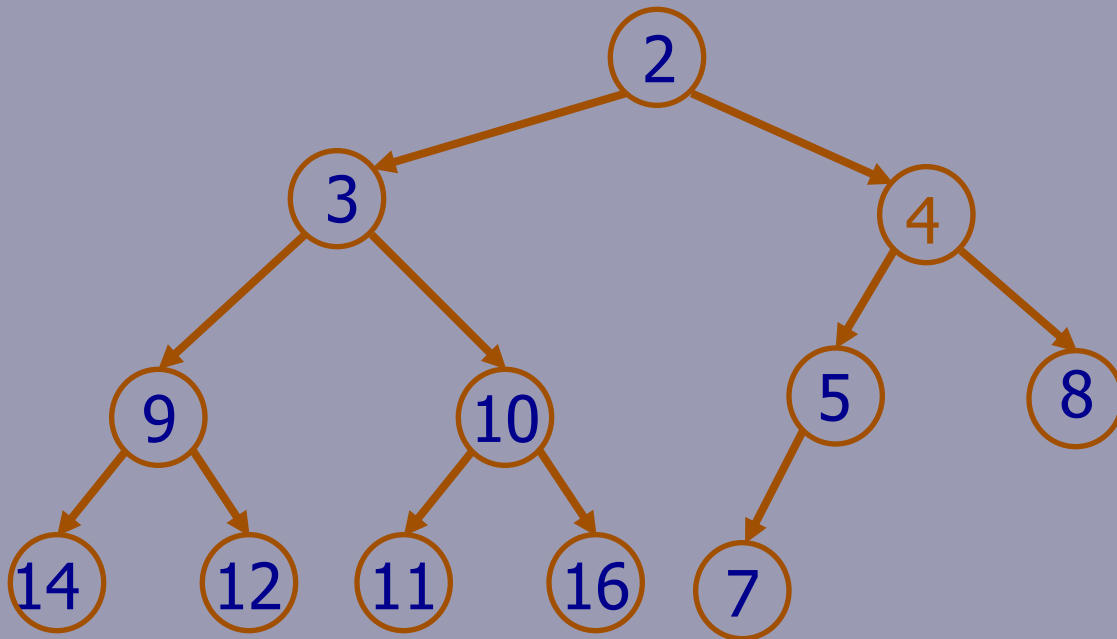
- How to get the smallest element from a binary heap?
 - Return the first element.
- How to add a new element to a binary heap?
 - Insert the new element at the end of the heap
 - Fix the heap order



Add 4 to this heap??

Heaps and priority queues

- How to get the smallest element from a binary heap?
 - Return the first element.
- How to add a new element to a binary heap?
 - Insert the new element at the end of the heap
 - Fix the heap order



Add 4 to this heap??

Heaps and priority queues

- Presenting a binary heap using a dynamic array

```
struct dyArray { /* dyArray Structure */
    EleType *data;
    int size;
    int capacity;
};
```

- Function to swap value of 2 elements in the array:

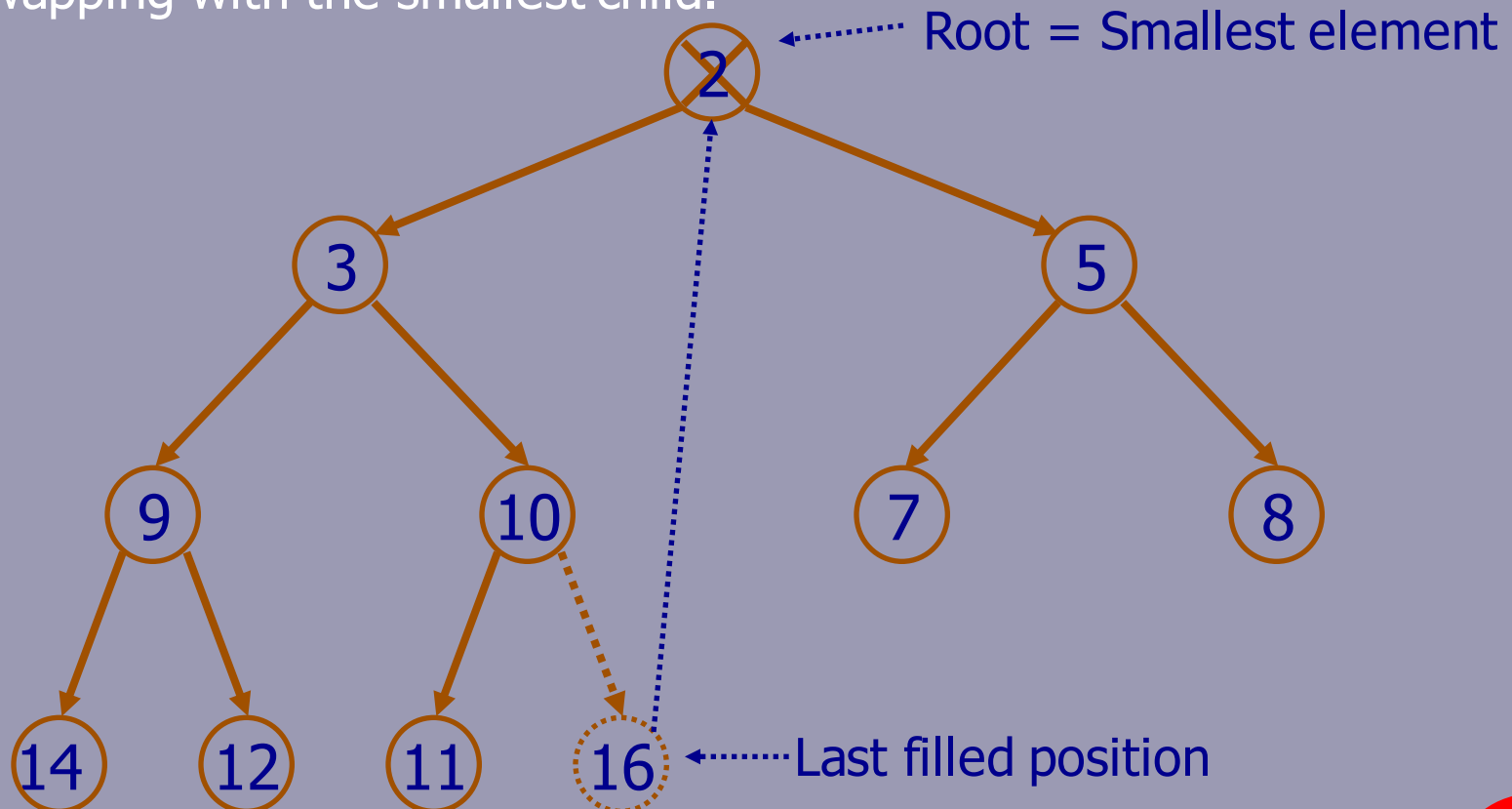
```
void swap (struct dyArray * v, int i, int j);
```

- Function to get the index of the smallest element between 2 elements in the heap:

```
int indexSmallest (struct dyArray * v, int i, int j);
```

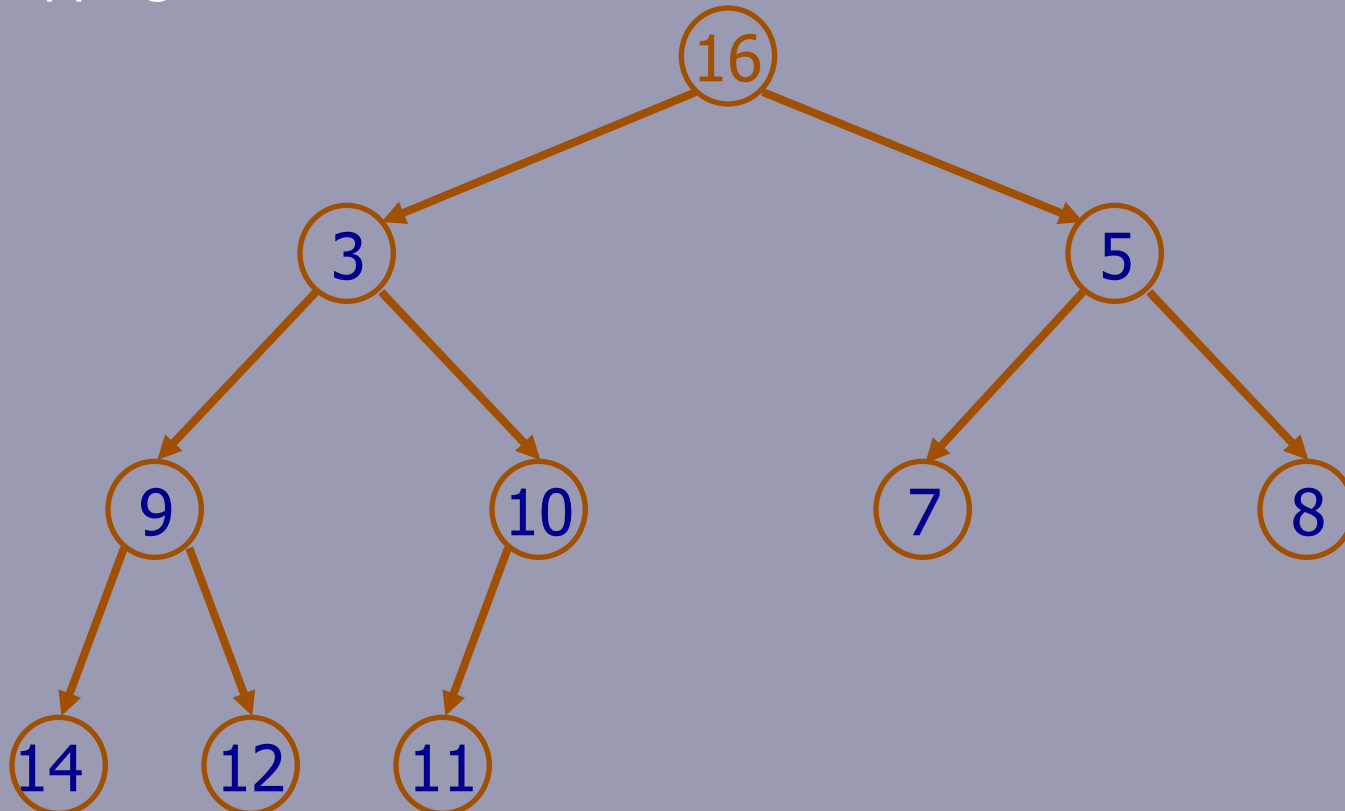

Heaps and priority queues

- When removing the first element, which element will replace it?
 - The last element !
- After removing, we need to call adjust heap to adjust the heap by swapping with the smallest child.



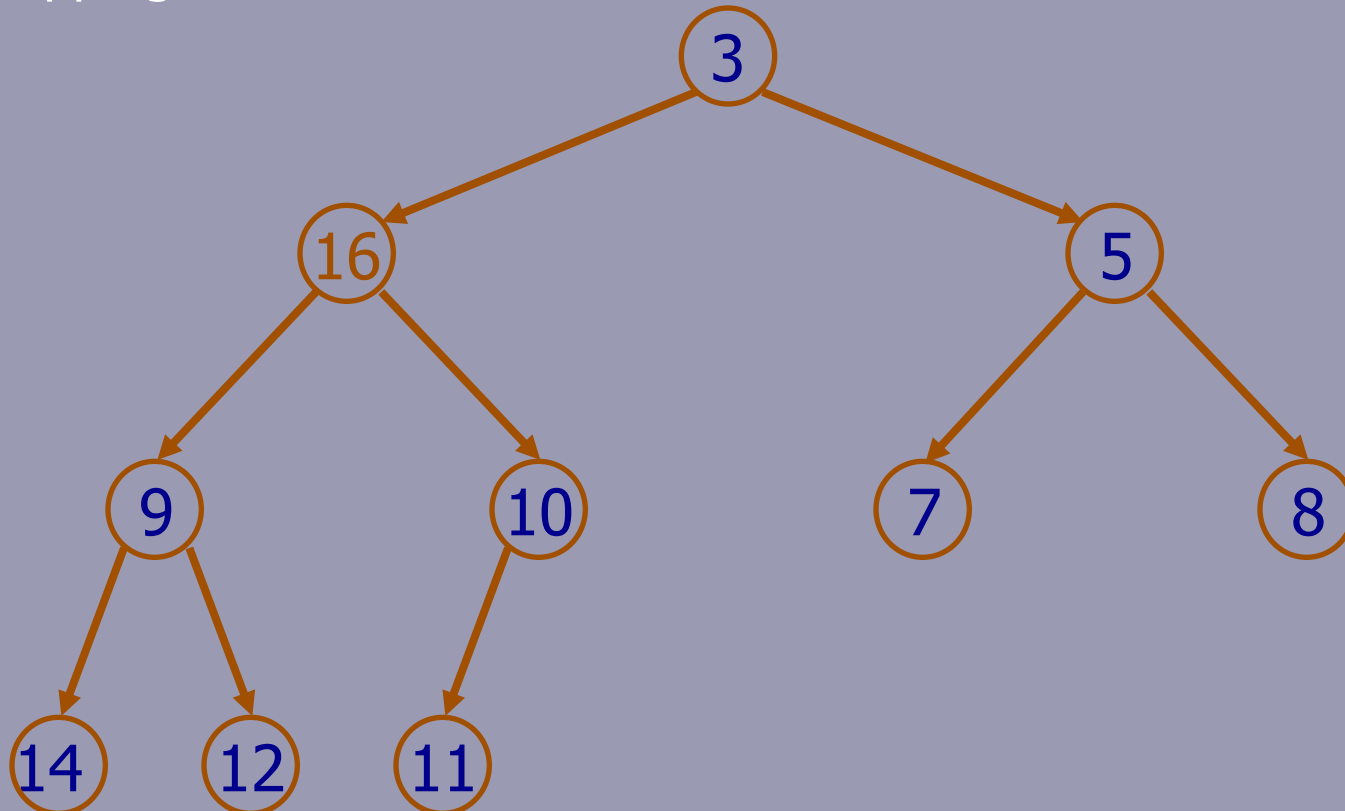
Heaps and priority queues

- When removing the first element, which element will replace it?
 - The last element !
- After removing, we need to call adjust heap to adjust the heap by swapping with the smallest child.



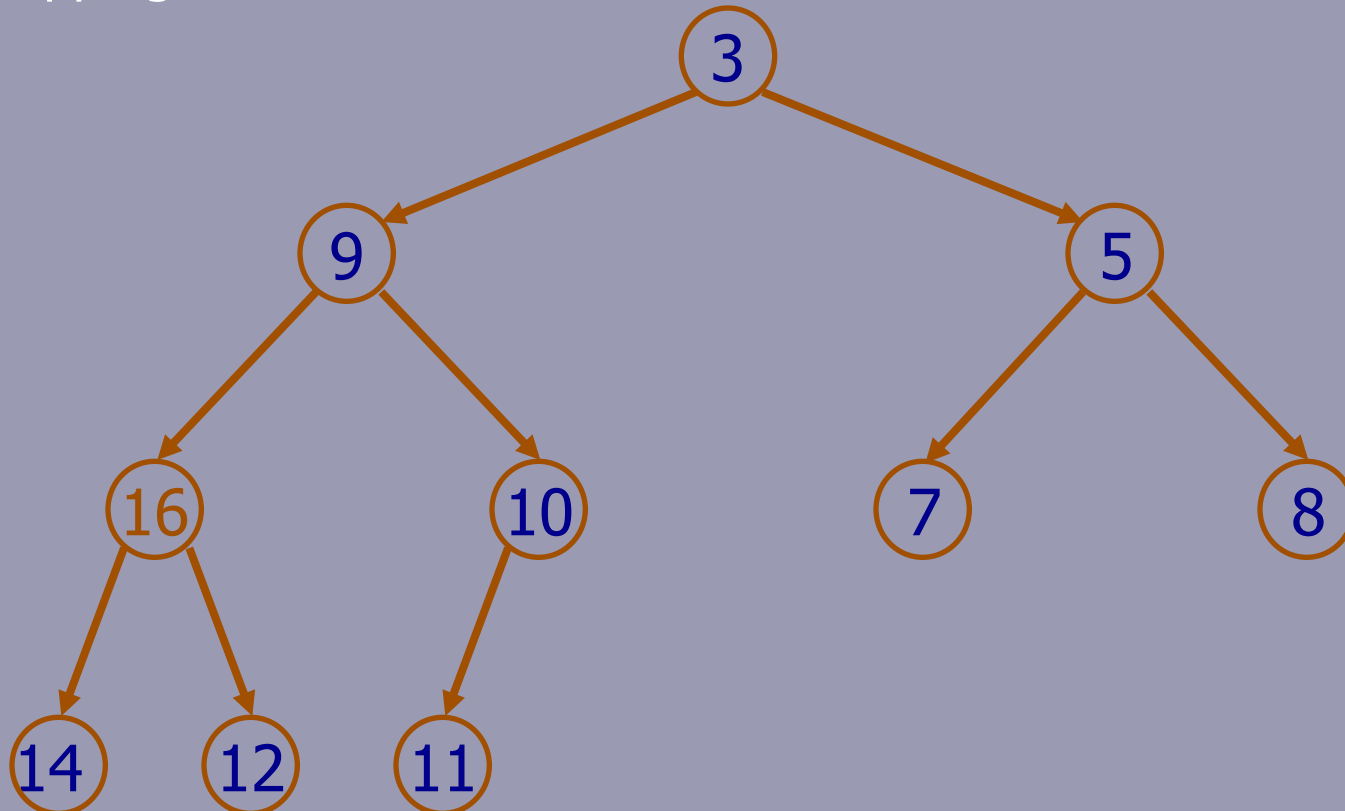
Heaps and priority queues

- When removing the first element, which element will replace it?
 - The last element !
- After removing, we need to call adjust heap to adjust the heap by swapping with the smallest child.



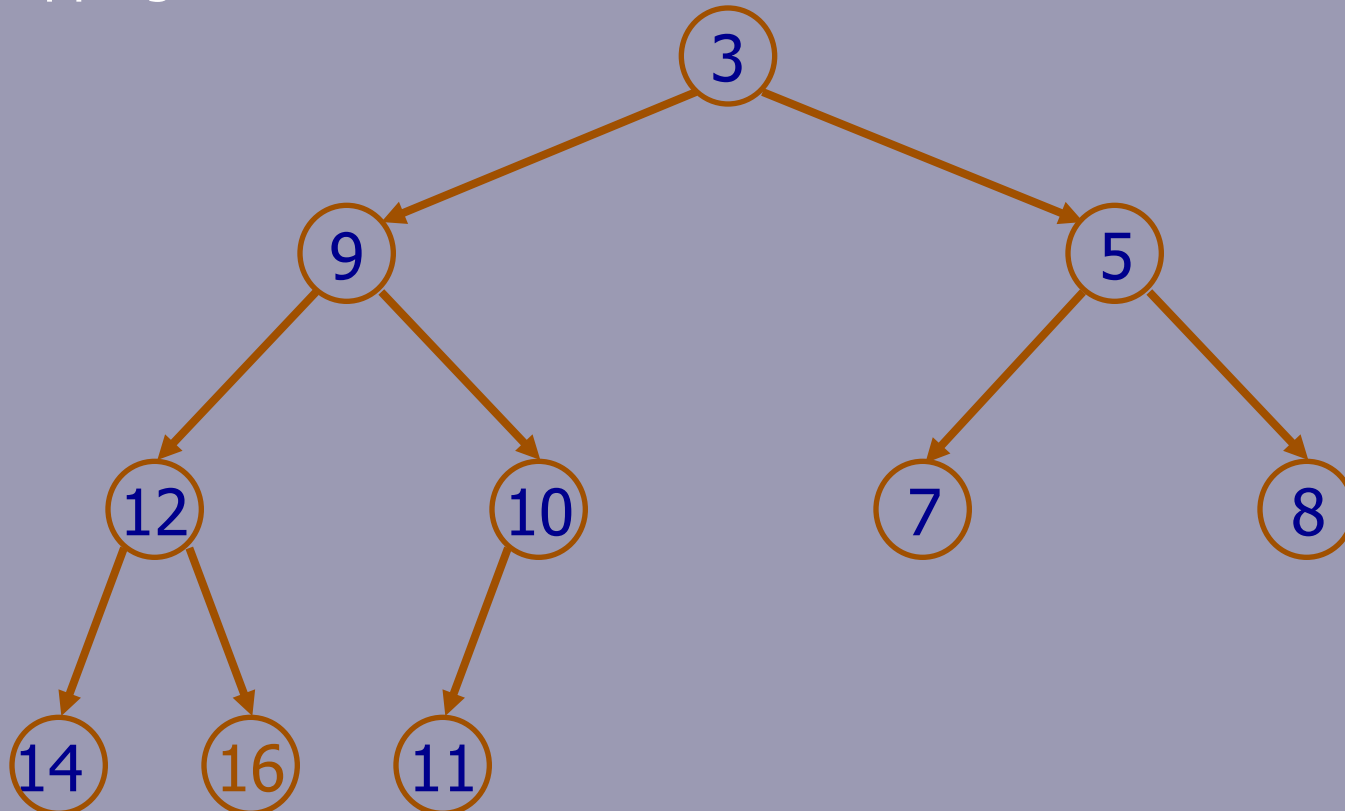
Heaps and priority queues

- When removing the first element, which element will replace it?
 - The last element !
- After removing, we need to call adjust heap to adjust the heap by swapping with the smallest child.



Heaps and priority queues

- When removing the first element, which element will replace it?
 - The last element !
- After removing, we need to call adjust heap to adjust the heap by swapping with the smallest child.



Command line arguments

- New prototype for main().. Similar to Java

```
int main (int argc, const char * argv[])
```

argc - number of arguments

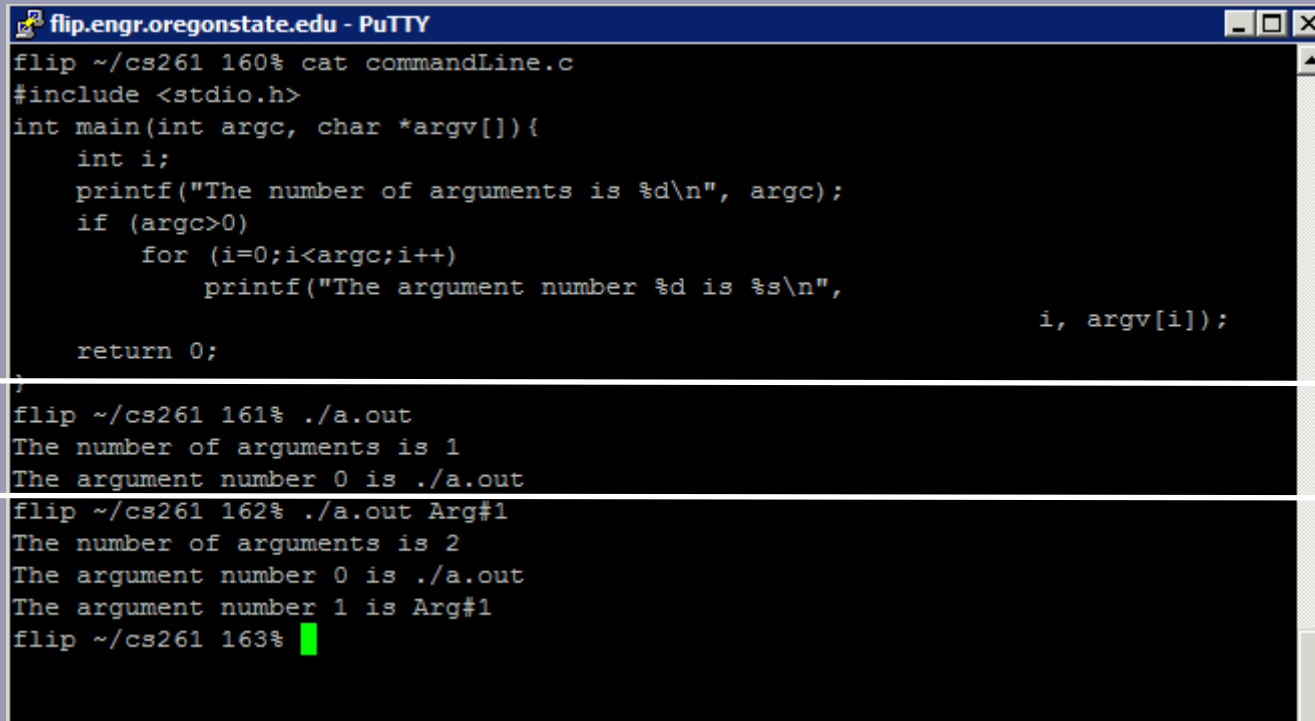
argv[] - array of arguments.

- Example:

```
int main(int argc, char *argv[]){
    int i;
    printf("The number of arguments is %d\n", argc);
    if (argc>0)
        for (i=0;i<argc;i++)
            printf("The argument number %d is %s\n",
                i, argv[i]);
    return 0;
}
```

Command Line Arguments

- Running the command line:



```
flip.engr.oregonstate.edu - PuTTY
flip ~/cs261 160% cat commandLine.c
#include <stdio.h>
int main(int argc, char *argv[]){
    int i;
    printf("The number of arguments is %d\n", argc);
    if (argc>0)
        for (i=0;i<argc;i++)
            printf("The argument number %d is %s\n",
                i, argv[i]);
    return 0;
}
flip ~/cs261 161% ./a.out
The number of arguments is 1
The argument number 0 is ./a.out
flip ~/cs261 162% ./a.out Arg#1
The number of arguments is 2
The argument number 0 is ./a.out
The argument number 1 is Arg#1
flip ~/cs261 163%
```

=> There is always a default argument, which is the name of the executable file.

File processing in C

- C communicates with files using a new datatype called a **file pointer**.
- This type is defined within **stdio.h**, and written as **FILE ***
- Usage:

```
FILE *output_file;
```


Opening a file pointer

- Your program can open a file using the **fopen** function, which returns the required file pointer.
- If the file cannot be opened for any reason then the value **NULL** will be returned.

- Usage:

```
output_file = fopen("filename.txt", "w");  
if (output_file != NULL) {  
    .... /* do something */  
}
```



What is 'w' ?

Opening a file pointer

- **fopen** takes two arguments:
 1. the name of the file to be opened (filename.txt).
 2. an access character, which is usually one of:
 - **"r"** - open for reading
 - **"w"** - open for writing (creates file if it doesn't exist). Deletes content and overwrites the file.
 - **"a"** - open for appending (creates file if it doesn't exist)
 - Also, r+, w+ & a+. (Please explore on your own)

Reading from a file

- You can read a single character using the function **fgetc**.

```
int fgetc( FILE *fp );
```

- String values are read from a file or from a console using the function **fgets**. The function takes as argument a character array, the size of the array, and a file pointer

```
char buffer[100];  
fgets(buffer, 100, stdin); //stdin means Standard i/p  
printf("You just typed %s\n", buffer);
```

- But, next time you read, **fgets** overwrites the previous value stored in the array (buffer[]). To solve this, we should copy a string value into a new array which can then be stored in data structure

```
char * newStr (char * charBuffer) {  
    char * p = (char *) malloc(1 + strlen(charBuffer));  
    strcpy (p, charBuffer);  
    return p;  
}
```

Writing to a file

- You can use the **fputc** function to write a single character

*int fputc(int c, FILE *fp);*

- To write a line into a file, use **fputs** function

*int fputs(const char *s, FILE *fp);*

Closing a file pointer

- The **fclose** command is used to disconnect a file pointer from a file.
- Usage:

```
fclose(output_file);
```
- Make sure to close any open files once you are done working on them to avoid surprises.