Theoretical computer science:

27.5 Proposition. $\vdash_{\mathbf{K}+(\mathrm{A3})} \boxdot(A \leftrightarrow B) \rightarrow \boxdot(F(A) \leftrightarrow F(B))$.

27.16 Lemma. $w \models \boxdot(p \leftrightarrow A) \rightarrow \boxdot(\Box C_i(p) \rightarrow \Box C_i(H_i))$.

Also theoretical computer science:

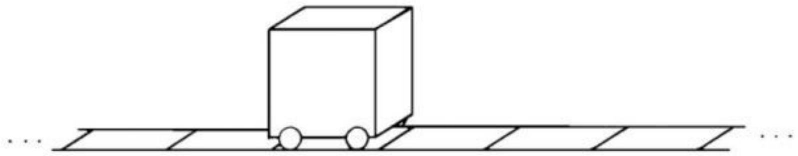

Figure 3-1. A Turing machine.

Lecture 3.2: Dynamically Allocated Memory

CS2308

Gentry Atkinson

Statically Allocated Memory

- Think about a program that needs 3 integers and an array that holds 12 characters.
- That program’s variable will occupy 24 bytes of memory every time it runs, no matter what.
- Static allocation is easy to use but can only hold a fixed amount of information.

Dynamic Memory Allocation

- Some programs need to be able to change the amount of information they are storing.
- Think about a High Score list:
  - static allocation: only store the 10 highest scores
  - dynamic allocation: store all scores and show them in an ordered list.

The **new** keyword.

- Used to allocate memory *at run time.*
- Returns a pointer to the new memory location.
- Can be used to allocate memory for any datatype: primitive, user defined, scalar, or array.

Example 1

```
int main(int argc, char** argv){
    int * p = new int;
    *p = 8;
    cout << *p << endl;
} //try to predict the output
```

Memory Management

- Statically allocated memory gets automatically “cleaned up” when it’s no longer in use.
- Dynamically allocated memory is not automatically freed.
- The **delete** keyword is used to release memory that has been dynamically allocated.

Example 2

```
//This function "leaks" 4 bytes of memory every
//  time it runs.
void leakyFunc(){
    int* p = new int;
    *p  = rand();
    cout << *p << endl;
}

int main(int argc, char** argv){
    for(int i = 0; i < 1000; i++)
        leakyFunc();
} //try to predict the output
```

Example 2.5

```
//This function de-allocates its memory
void notLeakyFunc(){
    int* p = new int;
    *p  = rand();
    cout << *p << endl;
    delete p;
}

int main(int argc, char** argv){
    for(int i = 0; i < 1000; i++)
        notLeakyFunc();
} //try to predict the output
```

Garbage Collection

- Allocated memory that cannot reached by a pointer is called “Garbage”.
- Removing wasteful garbage from memory is called garbage collection.
- Some languages have automatic garbage collection but not C++.

Allocating Arrays

- We can create arrays of any size dynamically at runtime.
- The **new** keyword is used for this too.

Example 3

```
int main(int argc, char** argv){
    int * arr = new int[10];
    arr[5] = 2;
    cout << arr[5] << endl;
    return 0;
} //memory leak!
```

De-allocating Arrays

- Arrays have to be cleaned up just like any other dynamically allocated memory.
- Use [square brackets] following the **delete** keyword to delete the whole array, not just the address pointed to.

Example 3.5

```
int main(int argc, char** argv){
    int * arr = new int[10];
    arr[5] = 2;
    cout << arr[5] << endl;
    delete [] arr;
    return 0;
} //nice and tidy!
```

Returning Pointers from Functions

- Dynamically allocated memory is not automatically freed when it goes out of scope.
- Pointers to dynamically allocated memory can be returned from functions (sorry Unit 3.1).

Example 4

```
int* makeArr(){
    int*p = new int[10];
    for(int i = 0; i < 10; i++) p[i] = i;
    return p;
}

int main(int argc, char** argv){
    int * arr = makeArr();
    cout << arr[0] + arr[1] << endl;
    delete [] arr;
} //try to predict the output
```

Using Dynamically Allocated Memory

- We can use DAM to:
  - Make arrays that grow as we add items.
  - Use arrays of pointers to stores large collections of **structs**.
  - Make quasi-2D arrays where each row has a different length.

Questions or Comments?