Math 4997-1

Lecture 20: Managing memory and low-level data structures

Patrick Diehl 

https://www.cct.lsu.edu/~pdiehl/teaching/2020/4997/

Reminder

Pointer

Arrays

Command line arguments

Memory management

Summary

Reminder

Lecture 19

What you should know from last lecture

- Running distributed HPX applications
- Using the slurm environment and modules on clusters

Why do we talk about pointers so late?

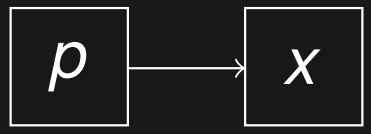
Person A: Would you teach a toddler how to eat with a butcher's knife?

Person B: No?

Person A: So stop mentioning pointers to people barely starting with C++

Pointer

Pointer



- A pointer *p* is a value that represents the address of an object
- Every object *x* has a distinct unique address to a part of the computer's memory.

Operators

- The address of object *x* can be accessed using the & address operator
- The deference operator * provides the object the pointer is pointing to

Example

```cpp
// Initialize
int x = 42;

// Get the pointer to the object x
int* p = &x;

// Get the object the pointer is pointing to
int tmp = *p;

// Using pointers to manipulate objects
std::cout << x << std::endl;
*p = 43;
std::cout << x << std::endl;
```

Pointer arithmetic

```
int* array = new int[3];
*array = 1;
*(array + 1) = 2;
*(array + 2) = 3;

// Accessing the first element
int first = *array;

// Accessing the second element
int second = *(array + 1);

// Getting the distance between two pointers
ptrdiff_t dist = array+2 - array;
```

Note that `ptrdiff_t` is a signed type because the distance can be negative

Pointers to functions

```
int square(int a)
{
return a * a;
}

// Generating a function pointer
int (*fp)(int) = square; //We need the (int) for
int (*fp2)(int) = &square; // the return type

// Calling the function using its pointer
std::cout << (*fp)(5);
std::cout << fp2(5);
```

Note that each of two lines to get the pointer or call the function are equivalent.

Pointer to classes and structures

```
struct A {

public:

A(double in)
{
        value = in;
}

double value;
};

A* a = new A(20);
double val = a->value;
```

Note that we use `->` expression to access the variable of a struct `a` using its pointer.

Arrays

Array vs Vector

Array[1][2]

- Language feature
- Number of elements must be known at compile time
- Can not grow or shrink dynamically

Vector

- Part of the standard library
- Can grow or shrink dynamically

An array is a kind of container, similar to a vector but less powerful.

---

[1] https://en.cppreference.com/w/cpp/container/array

[2] https://en.cppreference.com/w/cpp/language/array

Working with arrays

```
//Define the length
size_t size = 6;

//Generate a double array of size 6
double array[size];

//Access all elements
for(size_t i = 0; i < size ; i++){
        array[i] = i;
        std::cout << array[i] << std::endl;
    }

//Access the first element
*array = 42;
std::cout << array[0] << std::endl;

//Initializing
double array = {1,2,3.5,5};
```

Command line arguments

Arguments to `main`

```
int main(int argc, char** argv)
{

        return EXIT_SUCCESS:
}
```

Parameters

- `int argc` – Number of pointers in the `char** argv`
- `char** argv` – Initial pointer to an array of pointers for each command line option. Note that the first entry is the name of the executable.

Note that these parameters can have any name, but the two presented are very common.

Example

```
#include<iostream>


int main(int argc, char** argv)
{
    std::cout << argc << " argument(s)" << std::endl;

    for (size_t i = 0; i < argc ; i++)
    {
         std::cout << argv[i] << std::endl;
    }

    return EXIT_SUCCESS;
}
```

Memory management

Two kind of memory management

Automatic memory management

- What we have done so far
- The system is allocating the memory for a local variable
- The system is deallocating the memory if the variable goes out of scope

Dynamic memory management

- The programmer allocates the memory with the `new`[3] keyword
- The programmer deallocates the memory with the `delete`[4] keyword

---

[3] https://en.cppreference.com/w/cpp/language/new

[4] https://en.cppreference.com/w/cpp/language/delete

Memory management of an object

Allocation

```
int* p = new int(42);
```

Deallocation

```
delete p;
```

Memory management of an array

Allocation

```
int* p = new int[5];
```

Deallocation

```
delete[] p;
```

Summary

Summary

After this lecture, you should know

- Pointer and Arrays
- How to read command line arguments
- Allocating and deallocating memory