Programming Languages and Tools: Programming with C++ CS:3210:0003

Lecture/Lab #17

Address-of (&) and Indirection (*) Operators

- The address-of operator & returns the memory address of its operand
- Syntax:

```
int x;
std::cout << &x;        //Address storing x
```

- The dereference or indirection operator * returns the value at a given address
- Syntax:

```
std::cout << *(&x);     //Value stored at address of x
```

Pointer

- Object that holds a memory address of another object
- Syntax:

```
int* ptr;    //Pointer to int: holds address of an int
```

- Dereference pointer using * to access value at the address
- Initialize pointers using address of (&) operator
- Uninitialized pointer holds a garbage address
- Dangling pointer holds address of an object that is no longer valid
- Dereferencing dangling pointer leads to undefined behavior

Syntax Clarification

Syntax	Category	Description	Stores/Returns	Example

`int*`	Type Modifier	Pointer to int type	Stores address of int	`int x{};`
				`int * ptr {&x};`

`*`	Unary Prefix Operator	Dereference	Applied to address, Returns value stored at address	`int* ptr {};`

`int&`	Type Modifier	Reference to int type	Stores reference to int	`int x{};`
				`int& y{x};`

`&`	Unary Prefix Operator	Address of	Applied to var, Returns address of var	`int x{};`
				`int* ptr {&x};`

Null Pointer

- Null value: special value that means something has no value
- Null pointer holds a null value
- Value initialization of a pointer makes it a null pointer:

```
int* ptr{};   //null pointer
```

- nullptr represents a null pointer literal:

```
int* ptr{nullptr}; //null pointer
```

- Dereferencing null pointer leads to undefined behavior

const Pointers

- Regular pointers can’t point to const objects
- Pointer to const value can point to const (also non-const) objects

```
const int* ptr;    //can point to int or const int
```

- Compiler won’t let us change value of object pointed to by pointer to const value
- Pointer to const value is not const, it can be changed
- const pointer: pointer whose address can’t be changed after initialization

```
int x{};
int* const ptr {&x}; //cant be changed to point to another
                     //object
```

Const Pointers

Pointer Type	Can point to const?	Can point to non-const?	Can repoint to different object?	Can change value of pointed object?	Syntax
Regular Pointers	No	Yes	Yes	Yes	int* ptr;
Pointer to Const	Yes	Yes	Yes	No	const int* ptr;

Const Pointers	Yes	Yes	No	Yes	int y;
					int* const ptr{&y};

Const Pointer to Const	Yes	Yes	No	No	int y;
					const int* const ptr{&y};

Pointers to Classes

- Can’t use . to select member of a pointer to a class
- Arrow/member selection from pointer operator (->)
- Syntax:

```
Fraction* f{};
f -> getNum()        //equivalent to (*f).getNum()
```

- Equivalent to dereference then select

`this` Pointer

- Inside every member function, `this` is a const pointer
- `this` holds the address of the current implicit object
- The compiler handles `this` implicitly

Memory Allocation

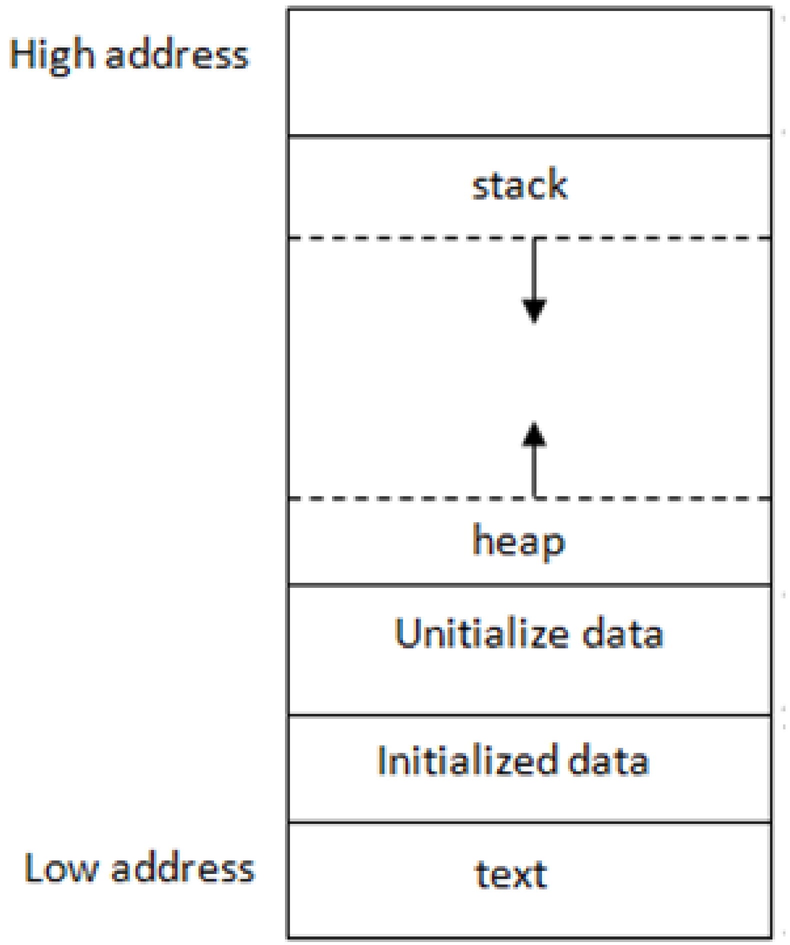
	Static Memory	Automatic Memory	Dynamic Memory
In C++:	Static, global vars	Function args, local vars	
Allocation:	Beginning of program	Declaration	
Deallocation:	End of program	End of block	

Dynamic Memory Allocation

- Static and automatic memory allocation
  - need to know size (and therefore, type) at compile time
  - memory is automatically managed
- Disadvantages of static memory:
  1. Over/underestimation of space needed for input
  2. Difficult to differentiate between initialized and uninitialized memory
  3. Limited memory available to each C++ program statically
- Dynamic memory allocation: memory allocation at runtime through operating system

Dynamic Memory Allocation

- Memory layout
  1. Stack: local variables
  2. Heap: dynamically allocated memory (managed by OS)
  3. Data: global (and static) variables
  4. Text: stores code
- Heap is much larger than stack
- Use `new` and `delete` operators in C++



new and delete

- new
  - Dynamically allocates memory and returns address
  - Can be stored in pointers
  - Memory is allocated on heap
  - Can fail (rarely) with exception
- delete
  - Frees memory pointed to by argument pointer
  - Returns memory to OS
  - Set pointer to null pointer. Otherwise, dangling pointer
  - Deleting a null pointer has no effect

Memory Leak

- When access to dynamically allocated memory is lost before it can be deallocated
- Possible memory leak:
    1. Allocate memory to pointer in a block
    2. Pointer goes out of scope at end of block
    3. Now can’t deallocate memory
- Takes up free memory

Memory Allocation

	Static Memory	**Automatic Memory**	**Dynamic Memory**
In C++:	Static, global vars	Function args, local vars	Dynamically allocated memory
Allocation:	Beginning of program	Declaration	Using `new`
Deallocation:	End of program	End of block	Using `delete`

Command Line Arguments

- Optional string arguments passed to the program when it is launched
- Space separated after call to the binary
- To access command-line arguments:

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

- `argc`: argument count to program
  - argc >= 1, first argument is the name of the program/binary
- `argv`: argument vectors (values)
  - Array of char pointers, each points to C-style string
  - Length of array is argc
- OS parses command line arguments and passes it to program