Storage

Storage Classes

- Storage class relates to how data are stored
  - static objects
  - automatic objects
  - dynamic objects
- All variables (or objects) belong to a storage class
  - depends on where object is declared
  - depends on qualifications given during declaration
- Register variables
  - useful when a variable is heavily used
  - hint to the compiler to assign a physical register
  - not often supported or acted upon by compilers

Automatic Objects

- Automatic variables are stack variables
  - created when entering a function, destroyed on leaving
  - exist on the stack in a 'stack frame'
  - function parameters are also automatics

i and local are autos on func's stack frame →

auto is an automatic on main's stack frame →

```
#include <iostream>

void func (int i)
{
    int local;
}

int main ()
{
    int auto_on_stack;
}
```

Static Objects

- Static objects are established by the compiler and exist in the data segment of the program.

implicitly static →

static, but in the scope of main() →

```
#include <iostream>

int global;

int main ()
{
    int auto_on_stack;
    static int data_seg;
}
```

- determined by the compiler
- variables exist for the duration of the program
- variables are global or declared with 'static' qualifier
- static data is initialised by the compiler to 0

Using Static Objects

```
#include <iostream>

int counter ();

int main()
{
    std::cout << "1st call: " << counter() << std::endl;
    std::cout << "2nd call: " << counter() << std::endl;
}

int counter()
{
    static int calls = 0;
    return ++calls;
}
```

```
$ ./a.out
1st call: 1
2nd call: 2
```

static storage is preserved between function calls, but is still within the scope of the function

Using Static Objects

- A function to convert from month number to name

```
const std::string& getMonth (int mon)
{
    static std::string name[] = {
        "curious month",
        "Jan", "feb", "mar", "apr",
        "may", "jun", "jul", "aug",
        "sep", "oct", "nov", "dec"
    };

    return (mon < 1 || mon > 12) ? name[0] :
    name[mon];
}
```

- Note the return type
  - Reference to static is safe (as not on local stack)
  - Const inhibits caller modifying data

Using Static Objects

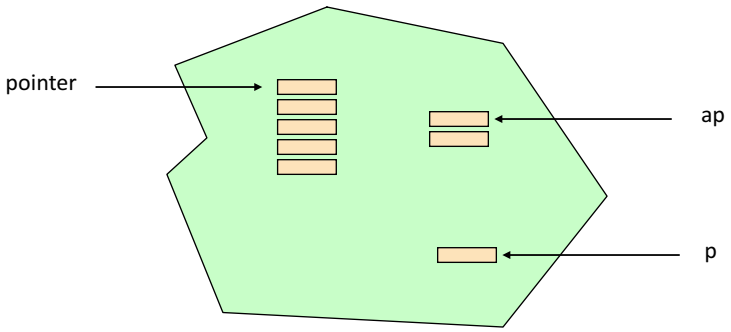
- A function to dispatch a function from a static table
  - useful for building finite state machines

```
#include <iostream>

void zero () { std::cout << "zero" << std::endl; }
void one () { std::cout << "one" << std::endl; }
void two () { std::cout << "two" << std::endl; }
void three () { std::cout << "three" << std::endl; }
void four () { std::cout << "four" << std::endl; }

void foo(int i) {
    static void (*lookup[])() = {zero, one, two, three, four};
    lookup[i]();
}

int main() {
    foo (1);
    foo (3);
}
```

```
$ ./a.out
one
three
```

Dynamic Memory

- Memory claimed and released on demand
  - maintained in a 'free-store heap'
  - using the `new` and `delete` operators




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

Dynamic Objects

- Dynamic objects are created on the heap
  - heap space is memory which may be randomly allocated and released by the programmer
  - dynamic objects must be explicitly allocated and explicitly released, and are not related to the stack or data segment
  - dynamic objects do not have identifiers; they simply have addresses which are stored in pointer variables

```
#include <iostream>

int main ()
{
    int *pointer;
    pointer = new int;
    *pointer = 3;
    delete pointer;
}
```

Dynamic Memory Operators

- Controlling the heap with new and delete

```
int main()
{
    char *ptr1, *ptr2, *aptr;

    ptr1 = new char;         // allocate a single char
    ptr2 = new char ('c');   // allocate and initialise char
    aptr = new char [300];   // allocate an array of char

    delete ptr1, ptr2;      // delete a single item
    delete [] aptr;         // delete an array of items
}
```

dynamic arrays are not initialised

delete ptr1 (where ptr1 = 0) is legal

always use [] when deleting an array

Using Dynamic Memory

- Supporting variable length records
  - with static arrays the size is determined at compile time

```
#include <iostream>
#include <iomanip>
#include <cstring>

char *getName ()
{
    char name [100];
    std::cin >> std::setw(100) >> name;
    char *p = new char [std::strlen(name) + 1];
    std::strcpy (p, name);
    return p;
}

int main()
{
    char *n = getName();
    std::cout << n << std::endl;
    delete [] n;
}
```

temporary array on stack

points to name

return memory to free-store heap

Object Lifetimes

```
int global;
static int file;
int func (int arg)
{
    static int stat;
    int local;
    int *p = new int;
    {
        int block;
    }
}
```

	storage class	*scope*	*memory*	*object lifetime*
`int global;`	static	global	data seg	program
`static int file;`	static	file	data seg	program
`int func (int arg)`	auto	function	stack	function
`static int stat;`	static	function	data seg	program
`int local;`	auto	function	stack	function
`int *p = new int;`	dynamic	pointer	heap	delete
`int block;`	auto	block	stack	block