C++ Variables

Variables in C++

- The variable
- Kinds of Variables
- Memory storage
- Variable qualifiers

The variable

- A variable declaration is a request for space in memory
  - Memory associated with a variable with size based on the kind of the variable.
  - Variable declarations are “executable” statements
    - Memory is allocated when declaration is made

The variable

- Basic data types
  - `int`, `short, long, unsigned`
  - `bool`
  - `char`
  - `float`
  - `double`

The variable

Variable declarations:

```
int foo;
float f = 7.0;
char c = ‘d’;
```

The variable

- Data type sizes (using CC on Sun)
  - sizeof (char) = 1
  - sizeof (bool) = 1
  - sizeof (short) = 2
  - sizeof (int) = 4
  - sizeof (unsigned) = 4
  - sizeof (float) = 4
  - sizeof (double) = 8
  - sizeof (long double) = 16

Kinds of variables

- Basic variable
- Pointer variable
- Reference variable

Variables in C++

- Basic variable
  - Memory associated with a variable with size based on the data type of the variable

```
int foo;
double f = 7.0;
double ff = f;
```

Variables in C++

- Pointer Variables
  - Stores the memory address of an object.
  - Can have pointers to basic data types.
  - C++ has no garbage collection!
  - NULL pointer takes value 0.

Variables in C++

Pointer variable

```
int *foo;
float *f = 7.0; // Invalid
float *g = 0;  // okay
float *h = 0x12345; // illegal!!
```

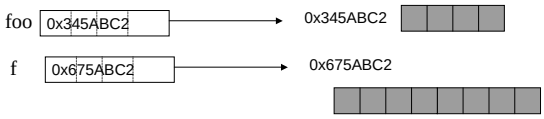
The variable

- Data type sizes (using CC on Sun)
  - sizeof (char *) = 4
  - sizeof (bool*) = 4
  - sizeof (short*) = 4
  - sizeof (int*) = 4
  - sizeof (unsigned*) = 4
  - sizeof (float*) = 4
  - sizeof (double* = 4
  - sizeof (long double*) = 4

Variables in C++

Pointer variable

```
int *foo;
double *f;
```



Variables in C++

- Address of operator
  - You can always get the address of any variable or object by using the address of operator &.
    - `float f = 7.0;`
    - `float *fptr = &f;`

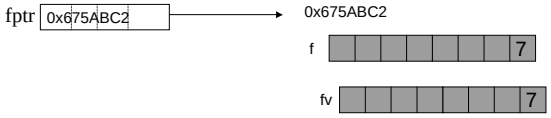
Variables in C++

- Pointer Variables
  - Dereference operator *
  - If `ptr` is a pointer
    - i.e A variable whose contents is a memory address
  - then `*ptr` refers to the object or data item that is pointed to by ptr
    - Can be interpreted as:
      - The data item or object at `ptr`
      - The object or data item pointed to by `ptr`

Variables in C++

Pointer variable

- `double f = 7.0;`
- `double *fptr = &f;`
- `double fv = (*fptr);`



Functions

- In C++ function arguments are pass by value:

```
int i = 7;
foo (i);
cout << i;
```

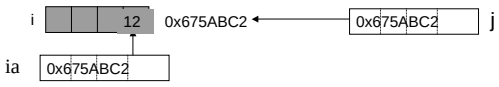
```
void foo (int j)
{
    cout << "Arg is " << j << endl;
    j = 12;
}
```

void – indicates that a function does not return a value

Functions

```
int i = 7;
int *ia = &i
foo (ia);
cout << i;
```

```
void foo2 (int *j)
{
    cout << "Arg is " << *j << endl;
    *j = 12;
}
```



Memory Storage Architecture

- Typically, a C++ program maintains 4 memory areas:

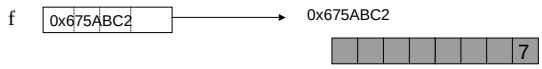
Static	For global variables
Stack	For function calls
Heap	Free store
Code	Executable code

Heap storage

- To allocate a variable on the heap, use `new`
  - `new` returns a pointer to the newly allocated space for the variable.

```
float *f = new float;
(*f) = 7.0;
```

  - All variables allocated on the heap using `new` must be deallocated using `delete`.



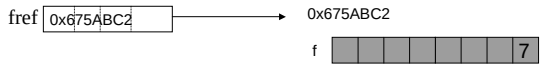
Variables in C++

- Reference Variables
  - Alias for an already existing object
  - Usually used to pass function arguments by reference.
  - Syntactically, references are treated like basic variables, yet they do contain memory addresses.

Variables in C++

- Reference variable

```
double f = 7;
double &fref = f;
cout << "The value is " << fref <<
     " and not " << (*fref);
```



Variables in C++

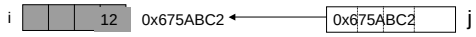
```
int &foo; // Invalid
int i;
int &iref = i; // okay
int &ref7 = 7;  // Invalid

int &badref = &i; // invalid
```

Functions

```
int i = 7;
foo (i);
cout << i;
```

```
void foo2 (int &j)
{
    cout << "Arg is " << j << endl;
    j = 12;
}
```



Variables in C++

- Questions?

Variable qualifiers

- const
  - A variable that cannot be modified. (oxymoron?)
  - When used with pointers – cannot modify the data the variable is pointing to.

Variable qualifiers

```
const int i = 7; // okay
i = 12; // not okay
```

Const pointers and functions

```
int i = 7;
int *ia = &i
foo (ia);
cout << i;
```

```
void foo2 (const int *j)
{
    cout << "Arg is " << *j << endl;
    *j = 12;  // Invalid!
}
```

Const pointers and functions

```
int i = 7;
const int *ia = &i
foo (ia);
cout << i;
```

```
void foo2 (int *j)
{
    cout << "Arg is " << *j << endl;
    *j = 12;
}
```

invalid

Global variables

- All variables defined outside a function are global:
  - Global variables are stored in static memory.
  - Global variables are accessible by all (except when declared as `static`)

```
int globI = 7;

  main () { … }
```

Extern

- Used to refer to global variables defined elsewhere.

```
int globI = 7;

main ()
{
   globI = 12;
   …
}
```

File 1

```
extern int globI;

int foo (int i)
{
   globI = i;
   …
}
```

File 2

Static

- Static variables are also stored in static memory
- `static` limits the scope of a variable to a file or function.
- Classes can also have static members
  - But more on that when we get to classes.

Static

```
static int globI = 7;

main ()
{
   globI = 12;
   …
}
```

Static

```
int foo (int i)
{
   static int globI = i;
   …
}
```

Compiler hints

- `volatile`
  - Indicates that the variable can be changed in unseen ways
    - E.g. Changed by another thread
    - Hint to compiler not to optimize away
- `register`
  - Hint to compiler to place variable in computer register
  - Cannot take address of a register variable

Summary

- Variables are a request for memory to store data
- Variable types
  - Basic / Pointer / Reference
- Memory Organization
  - Static / Stack / heap / Code
- Variable Qualifiers
  - const / static / extern / register / volatile

- Questions?

Next time

- Aggregate data structures
  - Arrays
  - union
  - struct

- Have a good weekend.