

12.010 Computational Methods of Scientific Programming 2021

Lecture 17: C/C++ Continued

Continuing C/C++

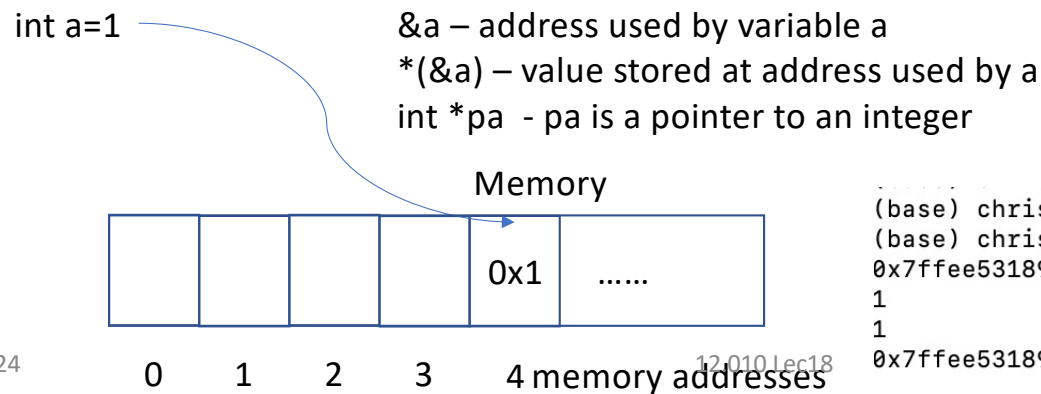
- Pointers
- Structures

C pointers

- Some of Lec17_polyarea.c is a bit clunky e.g. NMAX

```
int read_nodes(double n_xy[2][NMAX], int nmax){
```

- in many C codes pointers to blocks of memory are used instead of arrays, and “indexing” is computed in code.



```
#include <stdio.h>
int main(int argc, char *argv[]){
    int a=1;
    int *pa;
    pa = &a;
    printf("%p\n", pa);
    printf("%d\n", *pa);
    printf("%d\n", *(&a) );
    printf("%p\n", &a );
}
```

```
(base) chriss-MacBook-Pro:Downloads chrishill$ gcc ptr.c
(base) chriss-MacBook-Pro:Downloads chrishill$ ./a.out
0x7ffee531893c
1
1
0x7ffee531893c
```

C pointers

TAHMac[1559] cc Lec18_ptr.c

TAHMac[1560] a.out

Value in a == 7.000000

Memory address of a (in hexadecimal) ==
0x16cef327c

Value stored at address 0x16cef327c == 7.000000

Value in a == 3.000000

Value stored at address 0x16cef327c == 3.000000

Small test program – Lec18_ptr.c

- Shows accessing values versus addresses.

```
#include <stdio.h>
int main() {
    float a;           /* Floating point number */
    float *ptr_to_a;   /* Pointer to a floating point number */

    a = 7.;            /* Write 7. to memory location associated with a */
    printf("Value in a == %f\n",a);

    ptr_to_a = &a;     /* Get the address of the memory location where */
                      /* assignments to a get written. */
    printf("Memory address of a (in hexadecimal) == %p\n",ptr_to_a);

    /* Now use pointer to read value stored at an address */
    printf("Value stored at address %p == %f\n",ptr_to_a,*ptr_to_a);

    /* Write a new value to an address in memory */
    *ptr_to_a = 3.;

    /* What value does a have now? */
    printf("Value in a == %f\n",a);

    /* In C arrays and pointers are the same thing! */
    /* [0] is ptr_to_a + offset of 0*4 bytes */
    /* [1] is ptr_to_a + offset of 1*4 bytes */
    /* etc..... */
    printf("Value stored at address %p == %f\n",ptr_to_a,ptr_to_a[0]);
}
```

C pointers

- Try Lec18_ptr.c

C pointers with arrays

- C pointers are often used as an alternate to multi-dimensional arrays in scientific codes.
- Key parts are
 - array is a “pointer”
 - sizes are runtime values
 - C library functions malloc() and sizeof() are used
 - index arithmetic is part of code
 - C library function free() is used to release memory – prevents “memory leak”

11/14/2024

```
int main(int argc, char *argv[]) {
    /* Very common to use pointers instead of multi-dimensional arrays
    /* in many large C projects this is quite a common approach for
    double *A;
    int nx, ny;

    /*
    Now we use the standard library functions malloc() and sizeof
    malloc - requests a block of contiguous memory from the operating system
    sizeof - returns the size (in bytes) of a type, so that nx*ny
    return memory needed for nx*ny doubles (*A).

    */
    printf("Allocating memory for double array of size %d x %d\n",
        nx, ny);
    A = (double *)malloc(nx*ny*sizeof(*A));

    for (int j=0; j<ny; ++j){
        for (int i=0; i<nx; ++i){
            A[nx*j + i] = i*j;
        }
    }

    /* free() - releases the memory back
    free(A);
```

12.010 Lec18

C pointers with arrays

- Try Lec18_array_ptr.c – try and notice “warnings”

TAHMac[1563] cc Lec18_array_ptr.c

Lec18_array_ptr.c:29:26: warning: format specifies type 'int' but the argument has type 'unsigned long' [-Wformat]

```
29 | printf("Sizes %d %d\n",sizeof(*A),sizeof(A));  
   |           ~~      ^~~~~~  
   |           %lu
```

Lec18_array_ptr.c:29:37: warning: format specifies type 'int' but the argument has type 'unsigned long' [-Wformat]

```
29 | printf("Sizes %d %d\n",sizeof(*A),sizeof(A));  
   |           ~~      ^~~~~~  
   |           %lu
```

Lec18_array_ptr.c:32:54: warning: format specifies type 'int' but the argument has type 'unsigned long' [-Wformat]

```
32 | if( A == NULL ) {printf("Cant allocate %d bytes\n",nx*ny*sizeof(A)); exit(-1);}  
   |                        ~~      ^~~~~~  
   |                        %lu
```

3 warnings generated.

C with structs and typedef

- C struct and typedef are a way to make code more modular and very common and useful.
- Key parts are
 - declare a struct and "wrap" in typedef
 - declare a variable with the type
 - access components of the type using "."

```
/* A final important C feature is "typedef" and "struct"  
/* this allows custom types that can gather related information in  
/* variable.  
typedef struct {double *arr;  
                int    NX;  
                int    NY;  
} arr2d;  
  
arr2d A;
```

```
A.NX = atoi(argv[1]);  
A.NY = atoi(argv[2]);  
  
if ( A.NX < 1 || A.NY < 1 ) {  
    printf("ERROR\n");  
    exit(-1);  
}
```

```
printf("Allocating memory for double array of size %d x %d\n",A.NX,A.NY);  
A.arr = (double *)malloc(A.NX*A.NY*sizeof(*A.arr));
```

·
·
·

C pointers with structs

- Try Lec18_array_struct.c – try and make C poly area use malloc() and struct + typedef
- Lec18_poly_area_struct_ptr.c

C pointers, structs and typedefs → C++ with classes

- C++ builds on C structs, typedefs to create object oriented style.
 - A C++ class based polyarea

```
/* Pointer and struct approach */
typedef struct {double *narr;
               int   nnodes;
               } n_xy;

/* Function prototypes */
n_xy *read_nodes();
```



```
int main(int argc, char *argv[]){
    poly poly1;
    poly1.append( ppoint(0.,0.) );
    poly1.append( ppoint(2.,0.) );
    poly1.append( ppoint(2.,2.) );
    poly1.append( ppoint(0.,2.) );
    poly1.append( ppoint(0.,0.) );
    poly1.print();
    printf("Poly area = %f\n",poly1.area());
}
```

C structure and typedef.

With C++ can attach methods (aka functions) to new types, to create higher-level interfaces.

`poly` and `ppoint` are like C types, but they have their own data and methods (`append()`, `area()`, etc...) collected all in one concept (Class).

C++ adds classes to C.

C++

- First main-stream “object oriented” language. Compose complex applications from building blocks.
- Appeared around 1984 (Bjarne Stroustrup, Bell Labs)
- ANSI standard 1997
- Syntax is like C. Getting started: a few extra keywords + few new formalized concepts.
- Book “C++ The Core Language” – O’Reilly
- Successful because you can compose applications from other peoples building blocks. Windows etc....
- V. complex in detail, takes many years to learn everything!!

C++ compared to C

- C language + classes
- Class is a formal way to think about good program design.
 - Modularity, encapsulation, hierarchy, abstraction
- A class has
 - Methods (program logic)
 - Data (variables)
 - can be private or public
- We will look at expressing polyarea using C++ object-oriented classes.
- Start with “ppoint” class – point in a Polygon

```
/* First lets define a polygon point "class". */
class ppoint{
    public:
        ppoint(){};
        ppoint(double, double);
        ~ppoint(){};
        ppoint operator=(ppoint);
        ppoint operator+(ppoint);
        ppoint operator-(ppoint);
        void print();
    private:
        double X;
        double Y;
        friend class poly;
};
```

Example of defining a “class” for a polygon point in 2d.

C++ class

- C++ introduces
 - `class NAME{ ... }`
 - instance of a class is called an “object” e.g.
 - `ppoint ppoint1;`
 - class defines functions (methods) for operating on objects of type
 - class defines variables that hold the state of an object
 - in example state is the coordinates X and Y
 - methods are constructor and destructor (`NAME`, `~NAME`), `print()`, `=`, `+` and `-`.

```
/* First lets define a polygon point "class". */
class ppoint{
    public:
        ppoint(){};
        ppoint(double, double);
        ~ppoint(){};
        ppoint operator=(ppoint);
        ppoint operator+(ppoint);
        ppoint operator-(ppoint);
        void print();
    private:
        double X;
        double Y;
        friend class poly;
};
```

C++ ppoint constructor

- C++ constructors

- `class NAME{ ... }`
- all classes have constructor and destructor methods
- these have name that is the same as the class NAME (~ in front for destructor)
- can be defined in class block, or separately prefixed by full name `NAME::NAME`.
- C++ adds `CLASSNAME::FNAME` for full names of functions.
- Same function can have several definitions for different argument types (polymorphism).

```
/* First lets define a polygon point "class". */  
class ppoint{  
    public:  
        ppoint(){};  
        ppoint(double, double);  
        ~ppoint(){};
```

```
ppoint::ppoint(double xi, double yi)  
{ this->X = xi;  
  this->Y = yi;  
}  
.. . . .
```

Keyword “this” is used to indicate the “current” object.

C++ operator overloading

- Can redefine +, -, = etc... to have different definition for different type/class arguments.
- For polyarea we define an “=” and a “-” method for a ppoint object.
- These can be used when computing the vector between two points.

```
ppoint ppoint::operator= (ppoint pp2)
{
    X = pp2.X; Y = pp2.Y;
    return( *this );
}
```

```
ppoint ppoint::operator- (ppoint pp2)
{ ppoint temp;
  temp.X = X - pp2.X; temp.Y = Y - pp2.Y;
  return(temp);
}
```

```
ppoint v;
v = ppoint(2.,0.) - ppoint(0.,0.)
```

C++ creating a polygon class

- We create a second class “poly::”.
 - This can hold a collection of points that make up a polygon
 - Calculate area
 - Print coordinates
 - ppoint:: instances can be added using an “append” method
 - Each instance of poly:: keeps track of its internal storage and size and allocates extra memory when needed.

```
class poly{  
    public:  
        poly();  
        void append(ppoint);  
        void print();  
        double area();  
        ppoint *parr;  
        int ncur;  
        int nnodes_max;  
    private:  
        static const int nblk=10;  
};
```


C++ polygon constructor

- `poly::` class has an array of `ppoint::` class instances
 - the constructor (`poly::poly`) is invoked when an object of type `poly::` is declared.
 - the constructor uses the C++ “new” function to allocate an array of `ppoint::` instances
 - the `poly::` instance has its array pointer set to the new memory and the max elements and current element set.
 - the keyword “this” is used to refer to the current `poly::` instance (e.g. `poly1` in `main()`).

```
poly::poly(){  
    this->parr = new ppoint[this->nblk];  
    this->nnodes_max = this->nblk;  
    this->ncur=0;  
}
```

```
int main(int argc, char *argv[]){  
    poly poly1;  
    ...  
}
```

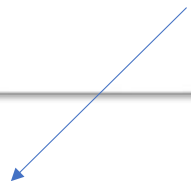
C++ polygon poly::print()

- Each `poly::` instance has a `print()` method
- The `poly::print()` method calls the `ppoint::print()` method for each `ppoint::` in the `parr` array of polygon points.

```
int main(int argc, char *argv[]){  
    poly poly1;  
    poly1.print();  
}
```

```
class poly{  
    public:  
        ppoint *parr;  
        void print();  
};
```

```
void poly::print()  
{  
    for (int i=0;i<this->ncur;++i){  
        this->parr[i].print();  
    }  
}
```



```
void ppoint::print()  
{  
    printf("(%f",X);  
    printf(",%f)\n",Y);  
}
```

C++ polygon poly::area

- Area computation
 - uses cross-product formula
 - the “-” sign is overloaded for ppoint.

```
ppoint ppoint::operator- (ppoint pp2)
{ ppoint temp;
  temp.X = X - pp2.X; temp.Y = Y - pp2.Y;
  return(temp);
}
```

```
double poly::area()
{
  double pa=0.;
  double tvec[2][2];
  ppoint v1, v2;
  for (int i=2;i<this->ncur;++i){
    v1=this->parr[i-1]-this->parr[0];
    v2=this->parr[i ]-this->parr[0];
    pa=pa+(v1.X*v2.Y-v1.Y*v2.X)*0.5;
  }
  return pa;
}
```

C++ polygon poly::append

- poly::append hides a couple of details
 - Appending
 - Growing the storage process

```
void poly::append(ppoint pa){  
    int nc=this->ncur;  
    int nm=this->nnodes_max;  
    if ( nc == nm ) {  
        int nmnew=nm+this->nblk;  
        ppoint *parrnew = new ppoint[nmnew];  
        for (int i=0;i<nc;++i){  
            parrnew[i]=this->parr[i];  
        }  
        delete this->parr;  
        this->parr=parrnew;  
    }  
    this->parr[nc]=pa;  
    ++this->ncur;  
}
```

C++ polyarea example

- Look at and try “Lec18_poly_area.cc” (use c++/g++ compiler).

TAHMac[1583] cc Lec18_poly_area.cc

Undefined symbols for architecture arm64:

"std::terminate()", referenced from:

__clang_call_terminate in Lec18_poly_area-32e891.o

"operator delete[](void*)", referenced from:

poly::poly() in Lec18_poly_area-32e891.o

poly::append(ppoint) in Lec18_poly_area-32e891.o

"operator delete(void*)", referenced from:

poly::append(ppoint) in Lec18_poly_area-32e891.o

poly::append(ppoint) in Lec18_poly_area-32e891.o

"operator new[](unsigned long)", referenced from:

poly::poly() in Lec18_poly_area-32e891.o

poly::append(ppoint) in Lec18_poly_area-32e891.o

"__cxa_begin_catch", referenced from:

__clang_call_terminate in Lec18_poly_area-32e891.o

"__gxx_personality_v0", referenced from:

/private/var/folders/t8/r6ksvxyc8xj78r0059bkhsm00000gr/T/Lec18_poly_area-32e891.o

ld: symbol(s) not found for architecture arm64

clang: error: linker command failed with exit code 1 (use -v to see invocation)

Correct compiler.

TAHMac[1581] c++ Lec18_poly_area.cc

TAHMac[1582] a.out

(0.000000,0.000000)

(2.000000,0.000000)

(2.000000,2.000000)

(0.000000,2.000000)

(0.000000,0.000000)

Poly area = 4.000000

Summary

- Looked at pointers and structures
- Next class; inheritance and classes in Python.

MIT OpenCourseWare

<https://ocw.mit.edu>

12.010 Computational Methods of Scientific Programming, Fall 2024

For more information about citing these materials or our Terms of Use, visit <https://ocw.mit.edu/terms>.