Computational Physics

Introduction to C and C++ languages

Korea University
Eunil Won

References

- The C Programming Language by Brian W. Kernighan, Dennis M. Ritchie
- The C++ Programming Language by Bjarne Stroustrup

Getting Started: C

```
#include <stdio.h>
main()
{
  printf("hello ,world\n");
}
```

tells the compiler to include information about the standard input/output library

function

```
/* your first program   */
```

This is for remark: won't affect your program (Compilers neglect this block)

- How to compile this program?

```
eunil$ gcc -o hello hello.c
eunil$ ./hello
hello ,world
eunil$
```

- What is gcc ?

gcc is a a freely available C language compiler (http://gcc.gnu.org/), nicely working under linux/unix OS/osx OS

Getting Started: C

- Variables and Arithmetic Expressions

$$^\circ\mathrm{C} = \frac{5}{9}(^\circ\mathrm{F} - 32)$$ *: conversion between Celsius and Fahrenheit temperatures*

```
#include <stdio.h>

main()
{
  int fahr, celsius;
  int lower, upper, step;

  lower = 0;
  upper = 300;
  step  = 20;

  fahr = lower;
  while (fahr <= upper)
  {
    celsius = 5 * (fahr-32) / 9;
    printf("%d\t%d\n",fahr, celsius);
    fahr = fahr + step;
  }
}
```

Output

```
MA55-93:c eunil$ gcc -o c2f c2f.c
MA55-93:c eunil$ ./c2f
0   -17
20  -6
40  4
60  15
80  26
100 37
120 48
140 60
160 71
180 82
200 93
220 104
240 115
260 126
280 137
300 148
```

Getting Started: C

- Variables and Arithmetic Expressions

A declaration announces the properties of variables; it consists of a type name and a list of variables such as

```
int fahr, celsius;
int lower, upper, step;
```

C provides several other basic data types besides `int` :

`int`	integer
`float`	floating number
`char`	character - a single type
`short`	short integer
`long`	long integer
`double`	double-precision floating point

Arithmetic Expressions

```
while (fahr <= upper)
{
  celsius = 5 * (fahr-32) / 9;
```

: statement about the condition

: multiplication, subtraction, division

Getting Started: C

- Variables and Arithmetic Expressions (2nd version)

$$^\circ\mathrm{C} = \frac{5}{9}(^\circ\mathrm{F} - 32)$$ *: conversion between Celsius and Fahrenheit temperatures*

```
#include <stdio.h>

main()
{
  float fahr, celsius;
  int lower, upper, step;

  lower = 0;
  upper = 300;
  step  = 20;

  fahr = lower;
  while (fahr <= upper)
  {
    celsius = (5.0/9.0) * (fahr-32.0);
    printf("%3.0f %6.1f\n",fahr, celsius);
    fahr = fahr + step;
  }
}
```

Output

```
MA55-93:c eunil$ gcc -o c2ff c2ff.c
MA55-93:c eunil$ ./c2ff
  0  -17.8
 20   -6.7
 40    4.4
 60   15.6
 80   26.7
100   37.8
120   48.9
140   60.0
160   71.1
180   82.2
200   93.3
220  104.4
240  115.6
260  126.7
280  137.8
300  148.9
```

Getting Started: C

- The For Statement

```
#include <stdio.h>

main()
{
  int fahr;

  for (fahr = 0; fahr <= 300; fahr = fahr + 20)
    printf("%3d %6.1f\n",fahr, (5.0/9.0)*(fahr-32));
}
```

: gives you the same results from the previous one

- The Symbolic Constants

```
#include <stdio.h>

#define LOWER 0
#define UPPER 300
#define STEP  20

main()
{
  int fahr;

  for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
    printf("%3d %6.1f\n",fahr, (5.0/9.0)*(fahr-32));
}
```

Array (one dimensional)

- One Dimensional Array

```
#include <stdio.h>

main()
{
  int i = 0;
  int fahr, fahr_[16];
  float celsius_[16];

  for (fahr = 0; fahr <= 300; fahr = fahr + 20)
  {
    fahr_[i] = fahr;
    celsius_[i] = (5.0/9.0)*(fahr_[i]-32);
    printf("%2d %3d %6.1f\n",i, fahr_[i], celsius_[i]);
    i++;
  }
}
```

Note: the array starts from 0 (not from 1) !

```
MA55-93:c eunil$ ./array
 0   0  -17.8
 1  20   -6.7
 2  40    4.4
 3  60   15.6
 4  80   26.7
 5 100   37.8
 6 120   48.9
 7 140   60.0
 8 160   71.1
 9 180   82.2
10 200   93.3
11 220  104.4
12 240  115.6
13 260  126.7
14 280  137.8
15 300  148.9
```

For array[n], we have array[0], array[1],..., array[n-1]

Function (by value)

- Function, returning values

```
#include <stdio.h>

int power(int , int);

main()
{
  int i;

  for (i=0;i<10;++i)
    printf("%d %d %d \n",i,power(2,i),power(-3,i));
 return 0;
}

int power(int base, int n)
{
  int p;

  for (p=1;n>0;--n)
    p = p * base;
  return p;
}
```

```
eunil$ ./power
0 1 1
1 2 -3
2 4 9
3 8 -27
4 16 81
5 32 -243
6 64 729
7 128 -2187
8 256 6561
9 512 -19683
```

return -type function-name (parameter declarations, if any)
{
declarations
statements
}

Function (by reference)

- Function, with pointers

```
#include <stdio.h>

void swap_v(int , int );
void swap_p(int *, int *);

main()
{
  int x = 10, y = 20;

  printf("before       : x = %d %d \n",x,y);
  swap_v(x,y);
  printf("after swap_v : x = %d %d \n",x,y);
  swap_p(&x,&y);
  printf("after swap_p : x = %d %d \n",x,y);

  return 0;
}

void swap_v(int vx, int vy)
{
  int temp;
  temp = vx;
  vx = vy;
  vy = temp;
}

void swap_p(int *px, int *py)
{
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```

Yes, we are talking about pointers...

```
$ ./swap
before       : x = 10 20
after swap_v : x = 10 20
after swap_p : x = 20 10
```

A pointer is a variable that contains the address of a variable
The unary operator & gives the address of an object

```
int x = 1, y = 2, z[10];

int *ip;          /* ip is a pointer to int */
ip = &x;          /* ip now points to x     */
y = *ip;          /* y is now 1             */
```

Structure

- structure (keyword is struct)

```
#include<stdio.h>

struct point {    /* we declare a structure here */
  int x;
  int y;
};

int main(void)
{
  struct point mypoint;

  mypoint.x = 1;  /* in this way we can assign values to members */
  mypoint.y = 2;

  printf("%d + %d = %d\n",mypoint.x,mypoint.y,mypoint.x+mypoint.y);
}
```

```
$ ./point
1 + 2 = 3
```

Structure can be nested. See below:

```
struct rect {
  struct point pt1;
  struct point pt2;
};
```

End of introduction: C

- There are a lot more to learn in C, but above is a minimum requirement for the our class

- I strongly urge you to exercise yourself with C language until you feel comfortable

Syntax highlighting

- Example with vim editor

```
for (int loop=0;loop&lt;N_FINESSE;loop++)
{
  if (((* (int *) bufptr) & 0xffff0000) != fHDR)
  {
    m_status = ERROR_NOfHDR;
    return m_status;
  }
  bufptr++;
  bufptr++;  // extra spacing btw fHDR and aHDR
  int bcount = (loop + 0xa) << 24;
  if (((  *(int *) bufptr)&0xff000000) == (aHDR|bcount))    // aHDR ?
  {
    bufptr++;
  }
    else
  {
    m_status = ERROR_NOaHDR;
    return m_status;;
  }
  //
  //  loop over aSGL (the tdc edge data)
  //  5 = number of headers and trailers
  //
  int nlength = ndata[loop] - 5;
  printf("before loop %x (length = %x %d)\n",*(int *) bufptr,nlength,nlength);
  for (int nhits=0;nhits&lt;nlength;nhits++)
  {
    bufptr++;
  }
}
```

From http://particle.korea.ac.kr/news/Nov-14-2005.html

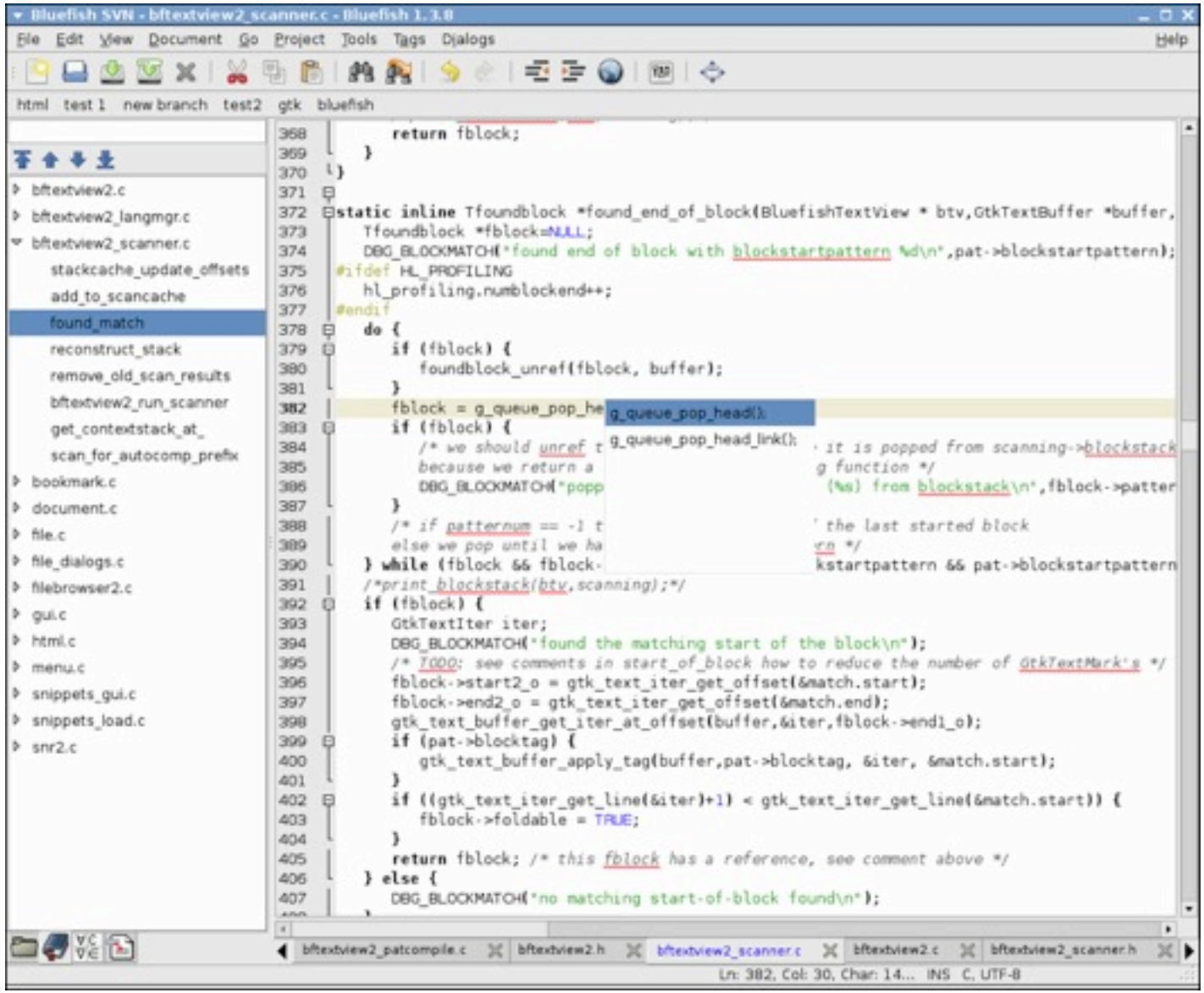
Syntax highlighting will help you to program and guide through source codes

Depending on your own choice of editor, you will have different look

- Editor?
  - vim
  - emacs
  - pico
  - bluefish
  - or your favorite...

Syntax highlighting

- Example with bluefish editor



I use vim - but cannot recommend it to you since it takes longer learning time (vim is one of text editors)

Probably for you all, this window-type editor might be better choice but choice is yours to make

Getting Started: C++

```
#include <stdio.h>
main()
{
  printf("hello ,world\n");     /* your first program   */
}
```

Remember this program?

- First do `$ cp hello.c hello.cc`
- How to compile this program as C++?

Roughly saying, c program is a subset of C++ program (so we can compile the above with C++ compiler)

```
eunil$ g++ -o hello hello.cc
eunil$ ./hello
hello ,world
eunil$
```

Usual convention is to put cc as the file extension for C++ source codes

- What is g++ ?

g++ is a a freely available C++ language compiler (http://gcc.gnu.org/), nicely working under linux/unix OS

Getting Started: C++

- Real C++ version of "Hello world"

```
#include <iostream>
int main()
{
  //
  // c++ version of remark
  //
  std::cout << "Hello, world" << std::endl;
}
```

Inclusion of standard library (STL)

`::` *indicates namespace*

Vim knows c++ syntax highlighting as well.

```
#include <iostream>
int main()
{
  //
  // c++ version of remark
  //
  std::cout << "Hello, world" << std::endl;
}
```

Namespace

- Namespaces allow to group entities like classes, objects and functions under a name. This way the global scope can be divided in "sub-scopes", each one with its own name.

The format of namespace is

```
namespace identifier
{
  entities
}
```

The format of namespace is

```
namespace myNamespace
{
  int a, b;
}
```

Then we can do for example,

```
myNamespace::a
myNamespace::b
```

```
#include <iostream>
using namespace std;

namespace first
{
  int var = 5;
}

namespace second
{
  double var = 3.1416;
}

int main () {
  cout << first::var << endl;
  cout << second::var << endl;
  return 0;
}
```

```
5
3.1416
```

Class - main idea in C++

- Class is a extended concept of `struct` from C

Classes are declared using the keyword `class`*:*

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
  } object_names;
```

For example,

```
class CRectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area (void);
  } rect;
```

What is "public" keyword?
Members can be functions !

```
#include <iostream>
using namespace std;

class CRectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area () {return (x*y);}
};

void CRectangle::set_values (int a,
int b) {
  x = a;
  y = b;
}

int main () {
  CRectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

```
area: 12
```

Class - main idea in C++

- constructors and destructors of a class

  - constructor : a special function which is automatically called whenever a new object of this class is created. This constructor function must have the same name as the class, and cannot have any return type; not even void

  - destructor : is automatically called when an object is destroyed

Question: why do we need to have constructors and destructors in C++?

```
#include <iostream>
using namespace std;

class CRectangle {
    int *width, *height;
  public:
    CRectangle (int,int);
    ~CRectangle ();
    int area () {return (*width * *height);}
};

CRectangle::CRectangle (int a, int b) {
  width = new int;
  height = new int;
  *width = a;
  *height = b;
}

CRectangle::~CRectangle () {
  delete width;
  delete height;
}

int main () {
  CRectangle rect (3,4), rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

```
rect area: 12
rectb area: 30
```

Class - main idea in C++

- Overloading constructors

  - a constructor can also be overloaded with more than one function that have the same name but different types or number of parameters

Note: usual functions can also be overloaded

```
#include <iostream>
using namespace std;

class CRectangle {
    int width, height;
  public:
    CRectangle ();
    CRectangle (int,int);
    int area (void) {return (width*height);}
};

CRectangle::CRectangle () {
  width = 5;
  height = 5;
}

CRectangle::CRectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  CRectangle rect (3,4);
  CRectangle rectb;
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

```
rect area: 12
rectb area: 25
```

Class - main idea in C++

- Pointers to classes

  - It is perfectly valid to create pointers that point to classes. to access directly to a member of an object pointed by a pointer we use the arrow operator (->) of indirection.

Careful. It is easy to get lost with pointers

```
a area: 2
*b area: 12
*c area: 2
d[0] area: 30
d[1] area: 56
```

```
#include <iostream>
using namespace std;

class CRectangle {
    int width, height;
  public:
    void set_values (int, int);
    int area (void) {return (width * height);}
};

void CRectangle::set_values (int a, int b) {
  width = a;
  height = b;
}

int main () {
  CRectangle a, *b, *c;
  CRectangle * d = new CRectangle[2];
  b= new CRectangle;
  c= &a;
  a.set_values (1,2);
  b->set_values (3,4);
  d->set_values (5,6);
  d[1].set_values (7,8);
  cout << "a area: " << a.area() << endl;
  cout << "*b area: " << b->area() << endl;
  cout << "*c area: " << c->area() << endl;
  cout << "d[0] area: " << d[0].area() << endl;
  cout << "d[1] area: " << d[1].area() << endl;
  delete[] d;
  delete b;
  return 0;
}
```

Class - main idea in C++

- Topics that I have not touched but important

  - Operator overloading
  - The keyword "this"
  - Static members
  - Friendship
  - inheritance
  - template
  - polymorphism
  - ...

- You do not need to know all these to survive in this class.
- If (you think) your intelligence is challenged, you can prove your ability of course.

Standard Library (STL)

- Standard library (STL): The standard C++ library is a collection of functions, constants, classes, objects and templates that extends the C++ language providing basic functionality to perform several tasks, like classes to interact with the operating system, data containers, manipulators to operate with them and algorithms commonly needed.

- Real C++ version of "Hello world"

```
#include <iostream>  : including input/output stream from STL
int main()
{
  //
  // c++ version of remark
  //
  std::cout << "Hello, world" << std::endl;
}
```

`cout` - object of class ostream that represents the standard output stream

`endl` - object of class ostream that Insert newline and flush

Standard Library (STL)

- Container class : let's take a look at "vector" class as an example

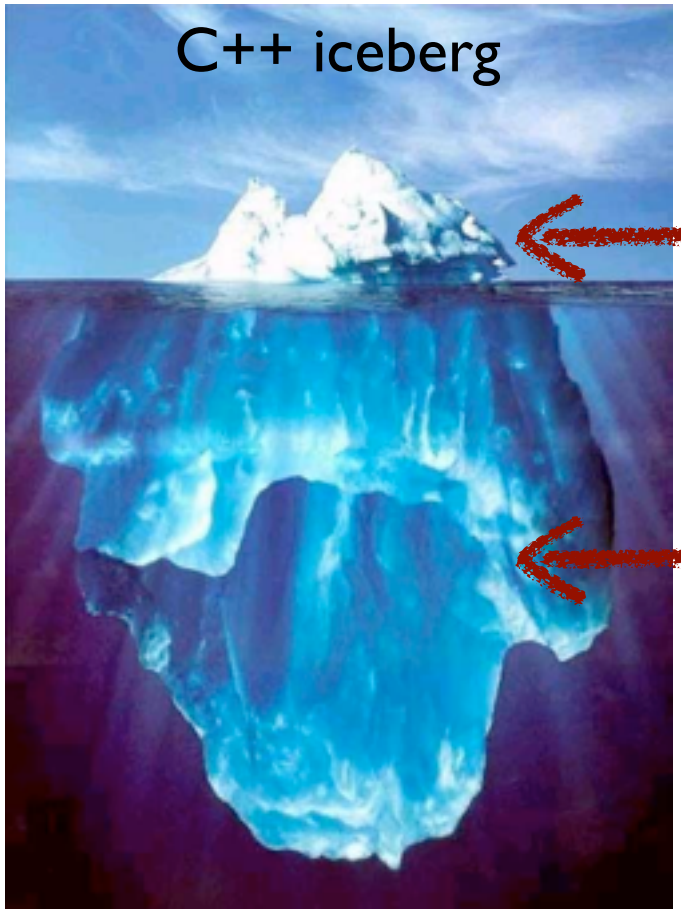
```
#include <iostream>
#include <vector>

using namespace std;
int main()
{
    vector <int> example;          //Vector to store integers
    example.push_back(3);          //Add 3 onto the vector
    example.push_back(10);         //Add 10 to the end
    example.push_back(33);         //Add 33 to the end
    for(int x=0; x<example.size(); x++)
    {
        cout<<example[x]<<" ";     //Should output: 3 10 33
    }
    if(!example.empty())           //Checks if empty
        example.clear();           //Clears vector
    vector <int> another_vector;   //Creates another vector to store integers
    another_vector.push_back(10); //Adds to end of vector
    example.push_back(10);         //Same
    if(example==another_vector)    //To show testing equality
    {
        example.push_back(20);
    }
    for(int y=0; y<example.size(); y++)
    {
        cout<<example[y]<<" ";     //Should output 10 20
    }
    return 0;
}
```

```
eunil$ ./vector
3 10 33 10 20 eunil$
```

End of C(++) class

- I only gave you very short introduction to C(++) language
  - What I covered is a "minimum" for our study in this semester
- There are of course tons of things you have not heard



This is what we covered

There are lots of C(++) techniques that we skipped