Scientific Computing (Phys 2109/Ast 3100H)
I. Scientfic Software Development

SciNet HPC Consortium

University of Toronto

Winter 2013

SciNet compute • calcul CANADA

Part I

Introduction to Software Development

Lecture 1

C++ Introduction

Language

SciNet compute • calcul CANADA

Why C++?

Advantages

- High performance
- Low-level programming
- Ubiquitous and standardized
- Useful libraries
- Modular design

Disadvantages

- Labour intensive, error prone
- High-level programming up to you
- Non-interactive
- Things like graphics can be hard
- Beware of performance pitfalls

For Python, advantages and disadvantages almost reversed.

No free lunch:

You can program poorly and inefficiently in any language.

C++ Introduction

- C was designed for (unix) system programming.
- C has a very small base.
- Most functionality is in (standard) libraries.
- C++ is almost a superset of C.

C++ Introduction

Example (Basic C++ program)

```
// hw.cc - prints 'Hello world.'  - RvZ/Jan 14, 2013
#include <iostream>
// include this to define input/output routines
int main() // called first
{ // braces delimit a code block
   std::cout << "Hello world."// print to standard out
             << std::endl;    // end of line
   // semicolon after statement
   return 0;
   // return value to shell
}
```

```
$ g++ -o hw hw.cc [-g -O2]
$ ./hw
Hello world.
$ echo $?
0
```

C++ Intro: Variable definition

type name [= *value*];

Here, *type* may be a

* floating point type:
  `float`, `double`, `long double`, `std::complex<float>`, ...
* integer type:
  [`unsigned`] `short`, `int`, `long`, `long long`
* character or string of characters:
  `char`, `char*`, `std::string`
* boolean:
  `bool`
* array, pointer
* class, structure, enumerated type, union

Non-initialized variables are not 0, but have random values!

C++ Intro: Functions

Function declaration (prototype)

```
returntype name(argument-spec);
```

argument-spec = comma separated list of variable definitions (may be empty)

e.g.: `int main(int argc, char ** argv);`

Function definition

```
returntype name(argument-spec) {
    statements
    return expression-of-type-returntype;
}
```

Function call

```
var=name(argument-list);
f(name(argument-list));
name(argument-list);
```

argument-list = comma separated list of values

C++ Intro: Namespaces

- Variables and function, as well as variable types, have names.
- In larger projects, name clashes can occur.
- Solution: put all functions, types, ... in a namespace:

```
namespace nsname {
...
}
```

- Effectively prefixes all of `...` with *nsname::*
  Example:
  `std::cout << "Hello, world" << std::endl;`
- Many standard functions/types are in namespace std.
- To omit the prefix, do using namespace *nsname*
- Can selectively omit prefix, e.g., `using std::vector`

C++ Intro: Pass by value or by reference

Passing function arguments (default)

```
// passval.cc
void inc(int i) {
   i = i+1;
}
int main() {
   int j = 10;
   inc(j);
   return j;
}
```

```
$ g++ -o passval passval.cc
$./passval
$ echo $?
10
$
```

C++ Intro: Pass by value or by reference

Passing function arguments by reference

```
// passref.cc
void inc(int &i){
   i = i+1;
}
int main() {
   int j = 10;
   inc(j);
   return j;
}
```

```
$ g++ -o passref passref.cc
$ ./passref
$ echo $?
11
$
```

C++ operators

Arithmetic

`a+b`	Add a and b
`a-b`	Subtract a and b
`a*b`	Multiply a and b
`a/b`	Divide a and b
`a%b`	Remainder of a over b

Logic

`a==b`	a equals b
`a!=b`	a does not equal b
`!a`	a is not true (also: `not a`)
`a&&b`	both a and b true (also: `a and b`)
`a||b`	either a or b is true (also: `a or b`)

Assignment

`a=b`	Assign an expression b to the variable b
`a+=b`	Add b to a (result stored in a)
`a-=b`	Subtract b from a (result stored in a)
`a*=b`	Multiply a with b (result stored in a)
`a/=b`	Divide a by b (result stored in a)
`a++`	Increase value of a by one
`a--`	Decrease value of a by one

C++ operators

Gotcha: Bad precendence

Relying on operator precedence is error-prone and makes code harder to read and thus maintain
(except for +, -, *, / and maybe %).

C++ Intro: Loops

```
for (initialization; condition; increment) {
    statements
}
```

```
while (condition) {
    statements
}
```

You can use `break` to exit the loop.

C++ Intro: Loops

Example

```
// count.cc
#include <iostream>
int main() {
   for (int i=1; i<=10; i++)
      std::cout << i << " ";
      // look, no braces!
   std::cout << std::endl;
}
```

```
$ g++ -o count count.cc -O2
$ ./count
1 2 3 4 5 5 6 7 8 9 10
$
```

C++ Intro: Pointers

- Pointers are essentially memory addresses of variables.
- For each type of variable *type*, there is a pointer type *type** that can hold an address of such a variable.
- Useful in arrays, linked lists, binary trees, ...
- Null pointer, denoted by 0, points to nowhere.

Definition:

```
type *name;
```

Assignment ("address-of"):

```
name = &variable-of-type;
```

Deferencing ("content-at"):

```
variable-of-type = *name;
```

C++ Intro: Pointers

Example

```
// ptrex.cc
#include <iostream>
int main() {
   int a = 7, b = 5;
   int *ptr = &a;
   a = 13;
   b = *ptr;
   std::cout<< "b=" << b << std::endl;
}
```

```
$ g++ -o ptrex ptrex.cc -O2
$ ./ptrex
b=13
$
```

C++ Intro: Automatic arrays

type *name*`[`*number*`];`

- *name* is equivalent to a pointer to the first element.
- Usage: *name*`[i]`. Equivalent to `*(`*name*`+i)`.
  This is really a just a different way to dereference a pointer.
- C/C++ arrays are zero-based.

C++ Intro: Automatic arrays

Example

```
// autoarr.cc
#include <iostream>
int main() {
   int a[6]={2,3,4,6,8,2};
   int sum=0;
   for (int i=0;i<6;i++)
      sum += a[i];
   std::cout << sum << std::endl;
}
```

B A D !!

(in general)

```
$ gcc -o autoarr autoarr.cc -O2
$ ./autoarr
25
$
```

Gotcha:

- C standard only says at least one array of at least 65535 bytes.
- In practice, limit is set by compiler and stack size.

SciNet compute • calcul CANADA

C++ Intro: Dynamically allocated array

A dynamically allocated arrays is defined as a pointer to memory:

```
type *name;
```

Allocated using the keyword `new` :

```
name = new type [number];
```

Deallocated by a function call:

```
delete [] name;
```

- Usage of these arrays is the same as for automatic arrays.
- Can access all available memory.
- Can control when memory is gived back.
- Unfortunately, no multi-dimensional version in the standard.

Improved version

Example

```
// dynarr.cc
#include <iostream>
int main() {
   int *a = new int [6];
   for (int i=0;i<6;i++){
      a[i]=i+2;
      if (i>=3)
         a[i] = 2*i*(15-2*i)-48; // is that really better?
   }
   int sum=0;
   for (int i=0;i<6;i++)
      sum += a[i];
   std::cout << sum << std::endl;
   delete [] a;
}
```

```
$ gcc -o dynarr dynarr.cc -O2
$ ./dynarr
25
$
```

C++ Intro: Dynamically allocated arrays

Example

```
// dyna.cc
#include <iostream>
void printarr(int n, int *a)
{
   for (int i=0;i<n;i++)
      std::cout << a[i] << " ";
   std::cout << std::endl;
}
int main()
{
   int n = 100;
   int *b = new int [n];
   for (int i=0;i<n;i++)
      b[i]=i*i;
   printarr(n,b);
   delete [] b;
   return 0;
}
```

```
$ g++ -o dyna dyna.cc -O2
$ ./dyna
0 1 4 9 16 25 36 49 64 81
100 121 144 169 196 225
256 289 324 361 400 441 484
529 576 625 676 729 784 841
900 961 1024 1089 1156 1225
1296 1369 1444 1521 1600
1681 1764 1849 1936 2025
2116 2209 2304 2401 2500
2601 2704 2809 2916 3025
3136 3249 3364 3481 3600
3721 3844 3969 4096 4225
4356 4489 4624 4761 4900
5041 5184 5329 5476 5625
5776 5929 6084 6241 6400
6561 6724 6889 7056 7225
7396 7569 7744 7921 8100
8281 8464 8649 8836 9025
9216 9409 9604 9801
```

C++ Intro: Conditionals

```
if (condition) {
   statements
} else if (other condition) {
   statements
} else {
   statements
}
```

Example

```
int main(){
   int n = 20;
   int *b = new int [n];
   if (b == 0)
      return 1; //error
   else {
      for (int i=0;i<n;i++)
         b[i] = i*i;
      printarr(n,b);
      delete [] b;
   }
}
```

```
$ g++ -o ifm ifm.cc -O2
$ ./ifm
0 1 4 9 16 25 36 49 64 81 100
121 144 169 196 225 256 289
324 361
$
```

To be continued in lecture 2 . . .

SciNet
compute • calcul
CANADA