C++ - introduction

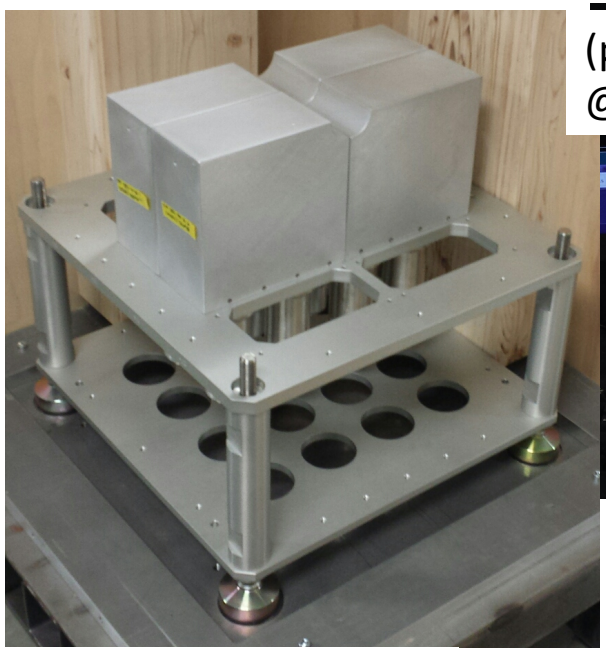
Anna Simon

August 26, 2015

Dr. Anna Simon
Room 221 NSH email: simon.instructor@nd.edu

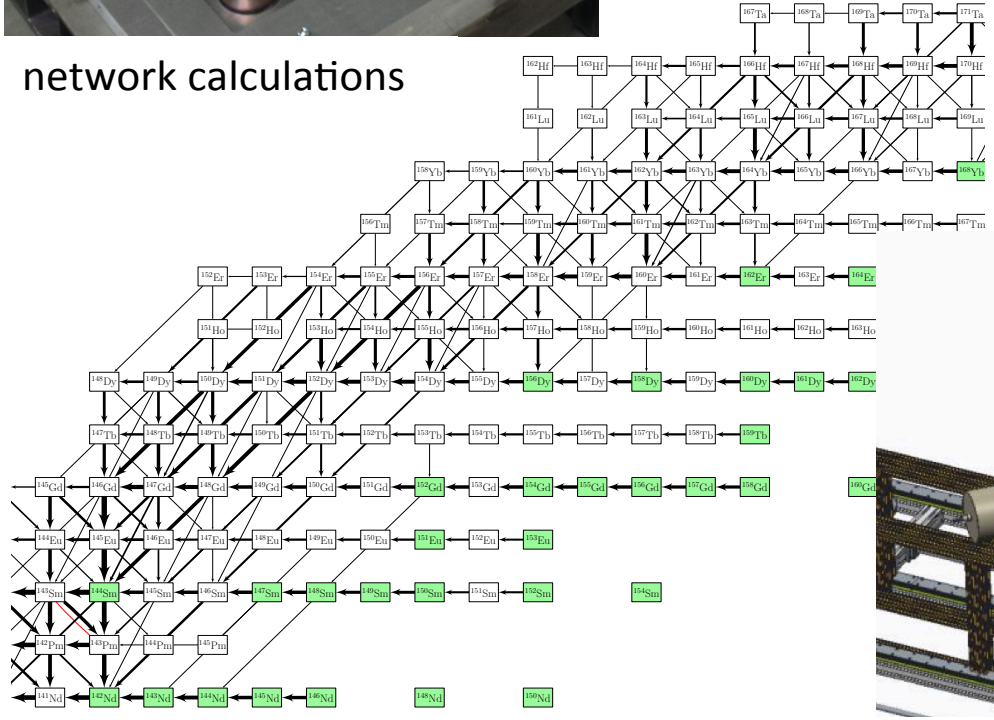
Office hours: please respond to the DOODLE pool (sakai).
Attendance counts towards your final grade!
Computers
CRC (sakai)

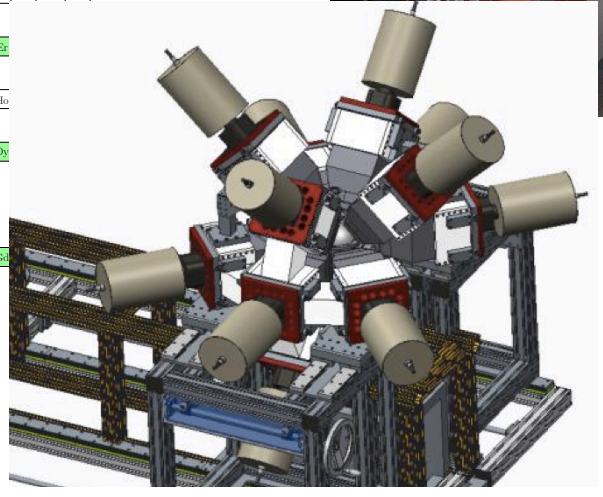
(p,γ) and (α,γ) reactions for p-process @ ND and NSCL



structure and statistical properties of actinides @ TAMU

network calculations





Course plan (subject to change)

Part I: C++ (until the fall break)

- Introduction
- Variables
- Operators
- Mathematical functions and operators
- Control flow
- Arrays, C-strings, Pointers and References
- Functions
- Input and output
- Strings
- Compiling a program using Makefile
- Passing arguments into main()
- Introduction to object oriented programming

Part II (second half of the semester)

- Introduction
- Data structure: trees
- Creating a class for analysis of the tree (MyClass)
- Histograms
- Canvas, pads
- Functions, fitting histograms, extracting fit parameters
- Other useful classes (TMath, TSpectrum, ...)
- Producing publication-worthy plots: TStyle, TLegend, TGraph, TStack
- Incorporating ROOT libraries into a C++ program

Structure of a program

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      char name[20]; /*this will be a
           name, max 20 characters
           long*/
8      cout << "What is your name?" <<
           endl;
9      cin >> name;
10     cout << "Hello "<< name << endl;
11     return 0;
12 }
```

- headers:
  <> look for file in C++ libraries,
  "" - look for files in local folders
- main() - this is where your program lives
- statements:

```
1 int x, y;   //declaration
2 x = 5;      //assignment
3 y = x + 3; //assignment and
      expression
4 cout << x; //output statement
```

basic I/O operators

`cin >> x;` prompts for user input and passes the value to x

`cout << "x=" << x;` outputs 'x=' and the value of x to screen

Compiling a simple program

```
g++ -o exec_name file_name.cpp
```

Usually `exec_name` and `file_name` are the same.

File extensions

Recognized as C++ by gcc and g++:

Unix:	.C (upper case), .cc, .cxx
GNU C++:	.C, .cc, .cpp, .c++, .cxx
Also used:	.CPP, .cp

Recognized as C: .c (lower case)

Comments within the code

```
#include <iostream>

int main()
{
    using namespace std;
    char name[20]; //this will be a
        name, max 20 characters
        long
    cout << "What is your name? <<
        endl;
    cin >> name;
    cout << "Hello "<< name << endl;
    return 0;
}
```

Use comments to explain:

what - at library, function, program level
how - inside library, function, program
why - at the statement level

```
//this is a hello world example
#include <iostream>

using namespace std;

//beginning of the program
int main()
{
        char name[20]; //this will be
            a name, max 20
            characters long
    // use cout and endl from
        iostream library output the
        text to screen
    cout << "What is your name?" <<
        endl;
    // read the user provided string
        of characters into 'name'
    cin >> name;
    // Welcome the user!
    cout << "Hello " << name << endl
        ;
    // everything went well, so
        return zero
    return 0;
}
```

Data types

Fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. They can mainly be classified into:

- Character types: They can represent a single character, such as 'A' or '$'. The most basic type is char, which is a one-byte character. Other types are also provided for wider characters.
- Numerical integer types: They can store a whole number value, such as 7 or 1024. They exist in a variety of sizes, and can either be signed or unsigned, depending on whether they support negative values or not.
- Floating-point types: They can represent real values, such as 3.14 or 0.01, with different levels of precision, depending on which of the three floating-point types is used.
- Boolean type: The boolean type, known in C++ as bool, can only represent one of two states, true or false.

Data types

?

What are bit and byte?

Group	Type names	Size/precision
Character types	char	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	short	Not smaller than char. At least 16 bits.
	int	Not smaller than short. At least 16 bits.
	long	Not smaller than int. At least 32 bits.
	long long	Not smaller than long. At least 64 bits.
Integer types (unsigned)	unsigned short unsigned int unsigned long unsigned long long	(same size as their signed counterparts)
Floating-point types	float	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	bool	
Void type	void	no storage

Data types

Example:
Size of the `int` and number of represantable values

Size	# of unique represantable values	
8-bit	256	$= 2^8$
16-bit	65 536	$= 2^{16}$
32-bit	4 294 967 296	$= 2^{32}$ (4 billion)
64-bit	18 446 744 073 309 551 616	$= 2^{64}$ (18 billion billion)

Question

What is the difference in the range represented by 16-bit `int` and 16-bit `unsigned int`?

How to check the size of a type?

`sizeof( type )` returns the size of the type in bytes.
`sizeof( variable )` returns the size of the variable in bytes.

Literals and typed constant expressions

Literal constants are numbers inserted into the code, e.g. 5, 1.3, 8e-4.

Suffixes

```
1 unsigned int nValue = 5u; // unsigned constant
2 long nValue2 = 5L; // long constant
3 float fValue = 5.0f; // float constant
```

Sometimes variables with a fixed value are needed throughout the code, e.g. electron mass, unit charge, etc. It is wise to make sure one cannot accidentally change their value when running the program. For this case typed constant expressions are used. Additionally this makes the code easier ot read.

Examples

```
1 const double pi = 3.1415926;
2 const char tab = '\t';
```

Question

Will a code with the following fragment compile?

```
1 const double pi = 3.1415926;
2 pi = 7;
```

Typed constant expressions

Compare the two example expressions:

```
energy =
((-10.82461)*1000.0) -
((1.0/106.)*(((106.+3)*telescope.
    energy) -
((106.-1.)*Ta) -
(2.0*sqrt(1.*3*34.566*1000.0*
    telescope.energy)*costheta[
    derid])));
```

```
const double MassProj = 1.;
const double MassProd = 106.;
const double MassEjec = 3.;
const double EneProj = 34.566;
const double QValue = -10.82461;
const double MeV2keV = 1000.0;

energy =
(QValue*MeV2keV)-
((1.0/MassProd)*(((MassProd+
    MassEjec)*telescope.energy)-
((MassProd-MassProj)*EneProj*MeV2
    keV)-
(2.0*sqrt(MassProj*MassEjec*EneProj
    *MeV2keV*telescope.energy)*
    costheta[derid])));
```

The definitions of all the constants can be placed in a separate header file, so that the user doesn't have to search through the code to find and modify them.

Escape codes

Name	Symbol	Meaning
Alert	\a	Makes an alert, such as a beep
Backspace	\b	Moves the cursor back one space
Formfeed	\f	Moves the cursor to next logical page
Newline	\n	Moves cursor to next line
Carriage return	\r	Moves cursor to beginning of line
Horizontal tab	\t	Prints a horizontal tab
Vertical tab	\v	Prints a vertical tab
Single quote	\'	Prints a single quote
Double quote	\"	Prints a double quote
Backslash	\\	Prints a backslash
Question mark	\?	Prints a question mark
Octal/hex number	\(number)	Translates into char represented by octal/hex number

Examples

```
1 '\n'
2 '\t'
3 "Left \t Right"
4 "one\ntwo\nthree"
```

Question

What is the difference between single and double quotes used in the examples?

Functions

In C++, statements are typically grouped into units called functions. A **function** is a collection of statements that executes sequentially.

- each C++ program has a main() function,
- functions are written to perform specific tasks, e.g. CalculateGrade(), GetAverageEnergy(),

Definition of each function requires:

- type (here: void)
- name (DoPrint)
- statements enclosed in curly brackets

```
1  #include <iostream>
2  using namespace std;
3
4  // Declaration
5  void DoPrint()
6  {
7      cout << "In DoPrint()" << endl;
8  }
9
10 int main()
11 {
12     cout << "Starting main()" <<
           endl;
13     DoPrint(); // Function call to
           DoPrint()
14     cout << "Ending main()" << endl
           ;
15     return 0;
16 }
```

Functions: implementation

The compiler reads the code from top to bottom, line by line. A function called within main() has to be **declared** before main().

Implementation before main()

```
1  #include <iostream>
2  using namespace std;
3
4  // Declaration and implementation
5  void DoPrint()
6  {
7      cout << "In DoPrint()" << endl;
8  }
9
10 int main()
11 {
12     cout << "Starting main()" <<
           endl;
13     DoPrint(); // Function call to
           DoPrint()
14     cout << "Ending main()" << endl
           ;
15     return 0;
16 }
```

Forward declaration

```
1  #include <iostream>
2  using namespace std;
3
4  void DoPrint(); //Function
       prototype
5
6  int main()
7  {
8      cout << "Starting main()" <<
           endl;
9      DoPrint(); // Function call to
           DoPrint()
10     cout << "Ending main()" << endl
           ;
11     return 0;
12 }
13
14 void DoPrint()  // Implementation
15 {
16     cout << "In DoPrint()" << endl;
17 }
```

Functions: return values

To obtain a function that returns a value, change the return type of the function in the function's declaration.

void the function does not return a value.

int the function returns an integer value to the caller. That value can be used within a program.

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

void ReturnNothing() // void:  does not return a value
{
    // This function does not return a value
}

int Return5() // int: returns an integer value
{
    return 5;
}

int main()
{
    cout << Return5(); // prints 5
    cout << Return5() + 2; // prints 7
    return 0;
}
```

Functions: parameters

Parameters are used to allow the caller to pass information to a function.

```
1  #include <iostream>
2  using namespace std;
3
4  // calculates the area of a rectangle of given length and width
5  int GetArea(int length, int width)
6  {
7      return length * width;
8  }
9
10 int main()
11 {
12     int x,y;
13     x = 5;
14     y = 3;
15     cout << GetArea(x,y) << endl;
16     return 0;
17 }
```