

Basic and Library Types

For : COP 3330.
Object oriented Programming (Using C++)
<http://www.compgeom.com/~piyush/teach/3330>

Piyush Kumar

Basic C++ Types

- integer data-types :
 - char, short, int, long, unsigned char, unsigned short,...
- floating point data-types :
 - float, double, long double
- logical data-type : bool
 - bool constants : true, false
- character data-type : char and wchar_t
 - Char constants in single quotes : 'a'
- text data-type :
 - string
 - string constants in double quotes : "Hello world"

Basic C++ Types

Recommended assignments: 2.4/2.7/2.8/2.9

- void
 - Return type for a function with no return value
 - void pointers
- unsigned and signed integers
 - unsigned char x = -1; ?
 - unsigned short int x = -1; ?

Wrap around effect.
Machine dependent.

Basic Arithmetic types

Type	Low	High	Digits of Precision	Bytes
char	-128	127	-	1
short	-32768	32767	-	2
int	-2147483648	2147483647	-	4
long	-2147483648	2147483647	-	4
float	3.4×10^{-38}	3.4×10^{38}	7	4
double	1.7×10^{-308}	1.7×10^{308}	15	8
long double	3.4×10^{-4932}	3.4×10^{4932}	19	10

Variable Naming

- Read about hungarian naming:
 - http://en.wikipedia.org/wiki/Hungarian_notation
 - <http://www.idleloop.com/hungarian/hungarian.html>
 - http://www.totse.com/en/technology/computer_technology/hungnote.html
- Use them in your next assignment onwards.

Hungarian naming

- Three parts
 - Base Type
 - One or more prefixes
 - Qualifier

Hungarian Base Types

- Specifies the type of the variable being named
- Not for predefined types
- Example:
 - wn = Window
 - scr = Screen
 - fon = Font
 - pa = Paragraph
- Example:
 - WN wnMain=NULL;
 - FONT fonUserSelected = TIMES_NEW_ROMAN;

Prefixes

- Go in front of the base type
- Somewhat standard list:
 - a = Array
 - c = Count
 - d = Difference
 - e = Element of an array
 - g = Global variable
 - h = Handle
 - i = index to array
 - m = Module-level variable
 - p(np, lp) = Pointer (near or long)
- Describes how the variable will be used.
- Examples
 - Array of windows: awnDialogs
 - Handle to a window: hwnMyWindow
 - Number of fonts: cfon

Qualifiers

- The rest of the variable name in case you were not using Hungarian naming.

More Examples

- bBusy : [boolean](#)
- cApples : count of items
- dwLightYears : double [word](#)
- fBusy : [boolean](#) (flag)
- iSize : [integer](#)
- fpPrice : [floating-point](#)
- dbPi : [double](#)
- pFoo : [pointer](#)
- rgStudents : array, or range
- szLastName : zero-terminated string
- u32Identifier : unsigned 32-bit [integer](#)
- m_nWheels : member of a class, integer

Variables

- What you should know?
 - Machine level representation of built in types.
 - Difference between : 'a' and "a"
 - Type of every entity in our program should be known to the compiler.

Variables

- Declaration
 - extern int i;
- Definition
 - int i;
 - Is also a declaration

↙
Type specifier

Variable Initialization

o Direct initialization

- float funity (1.0); → Can be more efficient for general classes.
- std::string all_nines(10,'9'); // #include <string>
- Happens when a variable is created and gives the variable its initial value.

o Copy initialization

- float funity = 1.0;
- Assignment involves replacing the current value of the variable with a new one.

More on this when we do "class" in detail.

Coding Standards

- o Always initialize your variables
- o Define variables where they are used.
- o Pick names carefully.
- o Be careful about scopes of variables
 - Global Scope
 - Local Scope
 - Statement scope

Const Qualifier

- o for (int Index = 0; Index < 512; ++Index)
- o const int ibufSize = 512;
- o ibufSize = 0; // error: attempt to write to const object
- o const int i, j = 0; ← What is wrong?
- o Const objects are local to a file by default.
- o Prefer const to #define
 - Using const instead of #define allows much better type checking and reduces name space pollution.

Const qualifier

o Const member functions

- Specify which member functions can be invoked on const objects.

```

Class Rational { ...
public:
    ...
    const int numerator(void);
    ...
    const Rational operator+(const Rational& rhs) const;
}

const Rational operator+(const Rational& lhs,
                        const Rational& rhs);

```

Const qualifier

o Const member functions

- Specify which member functions can be invoked on const objects.

```

Class Rational { ...
public:
    ...
    const int numerator(void);
    ...
    const Rational operator+(const Rational& rhs) const;
}

```

Rational a,b,c; a = b+c;
But does not allow (a+b) = c;

Const qualifier

o Const member functions

- Specify which member functions can be invoked on const objects.

```

const Rational operator+(const Rational& lhs,
                        const Rational& rhs);

```

When operator+ is a member function,
a = b+c is allowed, but what if you want to do
a = c + 2;
a = 2 + c;
Mixed mode operations are the reason why operators
are usually defined outside the class.

References

- What is a reference?
 - An alternate name for an object.
 - Frequently used for pass by reference.

```
void swap(int& i, int& j) {
    int tmp = i;
    i = j;
    j = tmp;
}

int main() {
    int x, y;
    ...
    swap(x,y);
    ...
}
```

Here i and j are aliases for main's x and y respectively. In other words, i is x — not a pointer to x, nor a copy of x, but x itself. Anything you do to i gets done to x, and vice versa.

References

- A reference is an alias.
 - `int ival = 1024; int &refVal = ival;`
 - `refVal += 2; // increases ival.`
- Errors:
 - `int &refVal2 = 10; // a reference must be initialized.`
 - `int &refVal3;`

Const references

- A reference that refers to a **const** object.
 - `const int ival = 1024;`
 - `const int& refval = ival;`
 - `const double& pi = 3.14; // only legal for const references`
- Errors:
 - `int &ref2 = ival; // Non-const reference to a const object.`

Const references

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

int main(void){
    int dval = 100;
    const int& ref = dval;
    ref++;
    return 0;
}
```

What is wrong?

Typedef names

Typedef lets us define a synonym for a type.

```
typedef double wages;
typedef int exam_score;
typedef wages salary;
typedef int Pounds, Shillings, Pennies, Dollars, Cents;
```

Variable definitions:

```
wages wage1, wage2;
exam_score person1, person2;
```

```
typedef struct { int scruples; int drams; int grains; } WEIGHT;
```

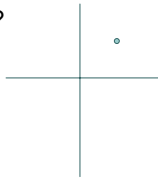
```
typedef class {
public:
    int scruples; int drams; int grains;
} WEIGHT;
```

Back to the C++ Class

- The C++ Class
 - Similar to a struct
 - Defines what objects of a class are (what attributes describe them)
 - Usually contains functionality in addition to attributes
- The Object
 - An *instance* of a class
 - A variable declared to be of a "class type"
- **class member**: a component of a class; may be either a data item or a function

Class example

- o Write a class to represent a two dimensional point.
- o What data attributes do we need?
- o What operations?



2D Point class

- o Data
 - Position
- o Operations
 - Set Position
 - Get Position
 - Distance from another point

Your first class.

```
#ifndef POINT2D_HPP
#define POINT2D_HPP

class point2D{
    int _x,_y;

public:
    void setX(const int val);
    void setY(const int val);
    int getX();
    int getY();
    double double_distance(point2D& p);
};

typedef Point2D Cell_tower;
typedef Point2D Cell_phone;

// Implementation goes here...

#endif
```

Include guards
Private
Interface

Using the point2D class

```
Point2D p,q;
p.setX(10); p.setY(10);
q.setX(0); q.setY(10);
```

```
Cout << "Distance between p and q is "
<< p.distance(q) << endl;
```

Implementing the point2D class.

```
#ifndef POINT2D_HPP
#define POINT2D_HPP

class point2D{
    int _x,_y;

public:
    void setX(const int val) {
        _x = val;
    }

    // ... Rest of the interface/implementation...
};

typedef Point2D Cell_tower;
typedef Point2D Cell_phone;

// Implementation goes here...

#endif
```

Implementing the point2D class.

```
#ifndef POINT2D_HPP
#define POINT2D_HPP

class point2D{
    int _x,_y;

public:
    void setX(const int val) ;

    // ... Rest of the interface/implementation...
};

typedef point2D Cell_tower;
typedef point2D Cell_phone;

// Implementation goes here...
void point2D::setX(const int val){
    _x = val;
}

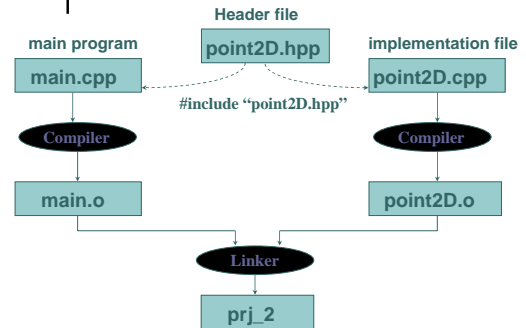
#endif
```

Can be also implemented in point2D.cpp

Headers

- Always use #include guards
- Do not define variables or functions in headers except
 - Class definitions
 - Const objects
 - Inline functions
- Difference between
 - #include <myfile.h>
 - #include "myfile.h"

Header files



Avoiding Multiple Inclusion of Header Files

- often several program files use the same header file containing typedef statements, constants, or class type declarations—but, it is a compile-time error to define the same identifier twice
- this preprocessor directive syntax is used to avoid the compilation error that would otherwise occur from multiple uses of #include for the same header file

```
#ifndef Preprocessor_Identifier
#define Preprocessor_Identifier
. . .
. . . h
. . .
#endif
```

Using the MyTimer class

- #include "timer.hpp"

```
MyTimer time_elapsed;
// your code goes here...
cout << time_elapsed;
```

A Problem with #define.

- #define max(a,b) ((a) > (b) ? (a) : (b))
- int a = 10, b = 20;
- max(a,b)
- max(++a,b)
- max(++a,b+10)