Fundamental C++ objects

1. Compiler needs to know in advance how to store different data types

2. Variable name + type, e.g. (price, integer)

3. Types:
   (a) Integers: short, long, signed
   (b) Floating Points: float, double
   (c) Character types

4. Declaration: before usage!
   (a) \{type\} variable name1, variable name2, etc..
   (b) \{type\} variable name1 = value, variable name=value2, etc...
   (c) For example: **double radius, diameter**
// radius.cpp: Calculates radius from circumference
#include <iostream.h>
#define PI 3.1415

int main (void){
   // variable declaration
   double d, r;
   // input circumference
   cout << "Circumference?\n";
   cin >> d;
   // calculate radius
   r = d / (2*PI);
   // output radius
   cout << "Radius = " << r << endl;
   return 0;
}

Declaration:

```cpp
double d, r;
```
Integers

1. Denoted: int
   (a) short int
   (b) int
   (c) long int
   (d) signed int, unsigned int

2. Difference: number of bytes allotted

3. size:
   (a) shortintbytes ≤ intbytes ≤ longintbytes
   (b) Determined by compiler
// test possible values of integer
#include <iostream>
int main (void){
    // declarations
    short int shi;
    int i;
    long int li;
    signed int si;
    unsigned int usi;

    // output byte size:
    cout << "short
int: " << sizeof(shi) << " , int: " << sizeof(i) << " , long
int: " << sizeof(li) << " , signed
int: " << sizeof(si) << " , unsigned
int: " << sizeof(usi) << ndl;

    return 0;
}
// test possible values of integer
#include <iostream>

int main ( void )
{
    // declarations
    unsigned short int usi;
    signed short int ssi;

    cout << "Value for unsigned short int:
";
    cin >> usi;
    cout << "usi value stored:
" << usi << endl;

    cout << "Value for signed short int:
";
    cin >> ssi;
    cout << "ssi value stored:
" << ssi << endl;

    return 0;
}
Characters

1. Denoted: char

   (a) Per definition: 1 byte

   (b) Underlying integer representation:

      i.  ’a’ < ’b’ < ’c’ < ’d’ …

      ii. ’A’ < ’B’ < ’C’ < …

2. Example: char c = ’A’
Bytes in `char` type

```cpp
// outputs number of bytes in char
#include <iostream>

int main ( void ){
    char c;
    cout << "bytes in char: " << sizeof(c) << endl;
    return 0;
}
```
// outputs number of bytes in char
#include <iostream>

int main ( void ){
    char c1, c2;

    c1 = 'a';
    cout << "c1= \n" << c1 << endl;
    c2 = c1 + 256;
    cout << "c2= \n" << c2 << endl;

    return 0;
}
Floating Point

1. Representation of numbers that have integer and fractional part, e.g. 3.14

2. Three types:
   (a) `float`
   (b) `double`
   (c) `long double`

3. Size:
   (a) `float ≤ double ≤ long double`
   (b) Determined by compiler
// outputs number of bytes in floats
#include <iostream>

int main (void){
    float f;
    double d;
    long double ld;

    cout << "bytes in float:\n" << sizeof(f);
    cout << "bytes in double:\n" << sizeof(d);
    cout << "bytes in long double:\n" << sizeof(ld) << endl;

    return 0;
}
Data types

Types:

1. Integer:
   (a) int
   (b) short
   (c) long
   (d) signed, unsigned

2. Real numbers:
   (a) float,
   (b) double
   (c) long double

3. Characters: char

Some notes:

1. Different byte sizes
2. short int\text{bytes} \leq \text{int}\text{bytes} \leq \text{long int}\text{bytes}
3. float \leq \text{double} \leq \text{long double}
4. char: 1 byte
Data Objects

1. Programs deal with data objects, e.g. numbers, text, etc.
2. These can change, either by input or by computation within program
3. We need to store data, and be able to refer to it conveniently
4. Data Objects:
   (a) Have a name
   (b) A type
   (c) and a value (stored in memory location)
5. Name: used to identify specific data objects
6. Type: tells compiler what “species” data object is
7. Value: can be set to anything corresponding to type
8. Data objects: a polite way to say “variable”
A note on variable/object names

**Variable name:**

1. Label or handle on value stored in a memory location
2. Compiler transparently translates name to memory location and back

**Valid names:**

1. Sequence of lower and uppercase letters and underscores
2. Name can not start with digit
3. Do not start with underscore or double underscore

**Examples:**

1. **Good:** diAmeTer2, radius2_Circle, Counter
2. **Bad:** 2Pac, ?Value?, SS#
More restrictions

**Keywords:**

1. Variable names can not be members of set of C++ keywords
2. Compiler needs to distinguish difference keywords and variables

**Good Practice:**

1. Make your variable names descriptive and distinctive
2. Make them short to avoid typos and confusion
3. t, i, j, usually used as counters
4. Avoid inconsistent capitalization
Fundamental C++ object types

1. Compiler needs to know in advance how to store different data types

2. Variable name + type, e.g. (price, integer)

3. Types:
   (a) Integers: short, long, signed
   (b) Floating Points: float, double
   (c) Character types

4. Declaration: before usage!
   (a) \{type\} variable name1, variable name2, etc..
   (b) \{type\} variable name1 = value, variable name=value2, etc...
   (c) For example: **double radius, diameter**
// radius.cpp: Calculates radius from circumference
#include <iostream.h>
#define PI 3.1415

int main (void) {
    // variable declaration
    double d, r;
    // input circumference
    cout << "Circumference?\n";
    cin >> d;
    // calculate radius
    r = d / (2 * PI);
    // output radius
    cout << "Radius=\n" << r << endl;
    return 0;
}
Integers

1. Denoted: \texttt{int}
   
   (a) \texttt{short int}
   
   (b) \texttt{int}
   
   (c) \texttt{long int}
   
   (d) \texttt{signed int, unsigned int}

2. Difference: number of bytes allotted

3. size:
   
   (a) $\texttt{short int}_{\text{bytes}} \leq \texttt{int}_{\text{bytes}} \leq \texttt{long int}_{\text{bytes}}$
   
   (b) Determined by compiler
/ test possible values of integer
#include <iostream>
int main (void) {
   // declarations
   short int shi;
   int i;
   long int li;
   signed int si;
   unsigned int usi;

   // output byte size:
   cout << "short int: " << sizeof(shi) << ", ";
   cout << "int: " << sizeof(i) << ", " long int: ";
   cout << sizeof(li) << ", " signed int: ";
   cout << sizeof(si) << ", " unsigned int: ";
   cout << sizeof(usi) << endl;

   return 0;
}
// test possible values of integer
#include <iostream>

int main ( void ){
    // declarations
    unsigned short int usi;
    signed short int ssi;

    cout << ”Value for unsigned short int:”;
    cin >> usi;
    cout << ”usi value stored:” ” ” usi << endl;

    cout << ”Value for signed short int:”;
    cin >> ssi;
    cout << ”ssi value stored:” ” ” ssi << endl;

    return 0;
}
Characters

1. Denoted: `char`
   
   (a) Per definition: 1 byte
   
   (b) Underlying integer representation:
       
       i. ‘a’ < ’b’ < ’c’ < ’d’ ···
       
       ii. ’A’ < ’B’ < ’C’ < ···

2. Example: `char c = ’A’`
Bytes in `char` type

```cpp
// outputs number of bytes in char
#include <iostream>

int main ( void ){
    char c;
    cout << "bytes in char: " << sizeof(c) << endl;
    return 0;
}
```


char math

// outputs number of bytes in char
#include <iostream>

int main ( void ) {
    char c1 , c2 ;

    c1 = 'a' ;
    cout << "c1=\"" << c1 << endl;
    c2 = c1+256;
    cout << "c2=\"" << c2 << endl;

    return 0;
}

January 18, 2004

Floating Point

1. Representation of numbers that have integer and fractional part, e.g. 3.14

2. Three types:
   (a) float
   (b) double
   (c) long double

3. Size:
   (a) float \leq double \leq long double
   (b) Determined by compiler
// outputs number of bytes in floats
#include <iostream>

int main (void) {
    float f;
    double d;
    long double ld;

    cout << ”bytes in float:” << sizeof(f);
    cout << ”bytes in double:” << sizeof(d);
    cout << ”bytes in long double:” << sizeof(ld) << endl;

    return 0;
}
Syntax and structure of C++ programs

```
#include<library>
#define PI 3.1415

int main (void){
  double a,b=3;
  variableDeclaration:

  Commands:
  c=b*PI;
  cout << c;

}
```
Data types

Types:

1. Integer:
   (a) int
   (b) short
   (c) long
   (d) signed, unsigned

2. Real numbers:
   (a) float,
   (b) double
   (c) long double

3. Characters: char

Some notes:

1. Different byte sizes
2. short int_{bytes} ≤ int_{bytes} ≤ long int_{bytes}
3. float ≤ double ≤ long double
4. char: 1 byte
Constants

Constants:

1. Data objects or variables store changing values
2. This means we need to write down values and assign them to these variables on occasion
3. Need for notation standard

Correspond to type:

1. String and character constants
2. Integer constants
3. Floating point constants
String and character constants

1. Sequence of zero or more characters enclosed in double quotes

2. Example: “Hello World!”

3. Size:
   (a) char: 1 byte
   (b) every string terminates with NULL char (o)
   (c) Length in Memory: number of chars + 1

![Hexadecimal representation of "Hello World!"

Jan 18, 2004

Escape characters

1. Purpose:
   (a) Escape mechanism: to include characters that do not have “visible” representation
   (b) To avoid confusion with language syntax (e.g. “ close string)

2. Non-exhaustive list:
   (a) New line: \n
   (b) Horizontal tab: \t

   (c) backslash: \\n
   (d) single quote: \’

   (e) double quote: \”
Example:

```cpp
// program to test string literals
#include <iostream>

int main (void){

    cout << "1\t2\t3\n";
    cout << "5\t6\t7\n";
    cout << "8\t3\t9\n";
    return 0;
}
```
Char constant

Representation:

1. Single character
2. Enclosed by single ‘
3. Example: ’A’, ’;’, ’\n’

// program to test char constant
#include<iostream>

int main ( void ){
    char c ;
    c = ’A’;
    cout << c << endl;
    return 0;
}

January 18, 2004

Integer Constant

Denotes integer number:

1. Example: 23 or 45 or 78 or 1001

2. First must be non-zero for base 10:

3. Different bases
   (a) octal (base 8): start 0, e.g. 012 (10)
   (b) hexadecimal: start 0xFF (255)
Integer Constant Example

// program to test char constant
#include <iostream>

int main ( void ){
    int i, o, hx;
    i = 23;
    o = 031;
    hx = 0xFF;
    cout "dec = " i " endl;
    cout "oct = " o " endl;
    cout "hexadecimal = " hx " endl;
    return 0;
}
Floating Point Constants

Two methods to write out floating-point constants:

1. Digits.Digits\{f,F,L,l\}
   (a) Example: 2.34F, 3.1415L, 29.00l, .45f
   (b) double is default, f,F = float, l,L = long double

2. Scientific Notation: Digits.Digits\{f,F,L,l\}
   (a) Example: 234E-2, 31415E-4, 29.001E0
   (b) Default is double, suffix f,F = double, l,L = long double
Floating Point Example

// test floating point constants
#include <iostream>

int main ( void ){
    float f1, f2;
    double d;
    long double ld;

    f1 = 2.34f; f2 = 34E-2F;
    cout << "f1=" << f1 << "\t" << "f2=" << f2 << "\n";

    d = 3141592E-6;
    cout << "d=" << d << "\n";

    ld = 5.311020L;
    cout << "ld=" << ld << "\n";

    return 0;
}
Floating Point

1. Representation of numbers that have integer and fractional part, e.g. 3.14

2. Three types:
   (a) `float`
   (b) `double`
   (c) `long double`

3. Size:
   (a) `float ≤ double ≤ long double`
   (b) Determined by compiler
// outputs number of bytes in floats
#include <iostream>

int main ( void ) {
    float f;
    double d;
    long double ld;

    cout << "bytes in float: " << sizeof(f);
    cout << "bytes in double: " << sizeof(d);
    cout << "bytes in long double: " << sizeof(ld) << endl;

    return 0;
}
Syntax and structure of C++ programs

#include<library>
#define PI 3.1415

Preprocessor
Directives

int main (void){

Variable Declaration:
double a,b=3;

Commands:
c=b*PI;
cout << c;

}