CISC 1110 (CIS 1.5)
Introduction to Programming Using C++

Spring 2012

Instructor : K. Auyeung

Email Address: kenny@sci.brooklyn.cuny.edu

Course Page: http://www.sci.brooklyn.cuny.edu/~kenny/cisc1110

Class Hours: MW 2:40 – 4:45PM 4428N

Agenda

- Brief Review
- Data types
- Binary & Hexadecimal Conversion
- Arithmetic Operations
- Increment & Decrement Operators
- Assignment Operators

What will be displayed?

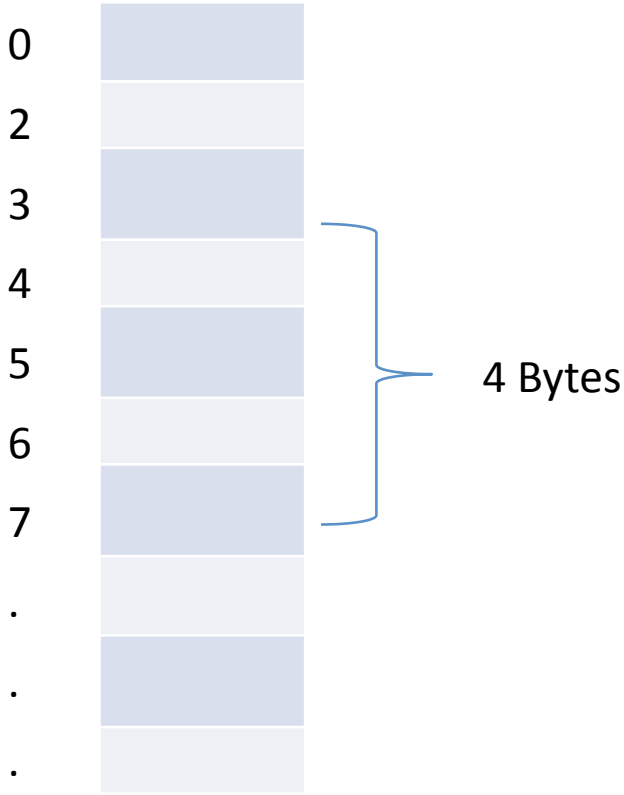
```
#include <iostream>

using namespace std;

int main()
{
        int number, sqnumber;

        number = 3;

        sqnumber = number * number;

        cout << number << " "
             << sqnumber
             << sqnumber * sqnumber
             << endl;

        return 0;
}
```

Storing Data

Program Code

```
int age = 21;
```

Computer Memory



Variables

- Variables have:
  - Name
  - Type
  - Value

- Variable/Identifier Names
  - Letters, either lowercase or uppercase
  - The digits 0 through 9
  - The underscore '_'

Variables Continued…

- Variable/Identifier Names
  - Should be Mnemonic

```
age
birthDate
carColor
size
firstName
angle
```

  - Are case sensitive
    - i.e. `booktitle` is not the same as `BOOKTITLE`

Name	Description	Size*	Range*
char	Character or small integer	1 byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer	2 bytes signed	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int	Long integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value, It can take one of two values: true or false	1 byte	
float	Floating point number	4 bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number	8 bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number	8 bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character	2 *or* 4 bytes	1 wide character

• Depends on the system the program is compiled on

Source: http://www.cplusplus.com/doc/tutorial/variables/

Data Type Modifiers

- Modifies the amount of memory allocated to a data type
    - short
    - long
    - signed
    - unsigned

Sample Code

```
#include<iostream>

using namespace std;

int main() {

    signed int balance = 450;
    int price = 1500;
    int negativeNumber = -120;

    bool oweMoney = true;


    balance = balance - price;
    cout << endl << endl << "Balance " << balance << endl << endl;

    cout << endl << endl << "My Negative Number " << negativeNumber << endl << endl;

    return 0;
}
```

Binary Numbers

- Base 2
- 1 and 0
- ASCII Codes
    - a = 97 (decimal)

OR

    - 0 1 1 0 0 0 0 1

Converting from Decimal to Binary

128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	0	0	0	0	1

$(1*2^6) + (1*2^5) + (1*2^0) = 64 + 32 + 1 = 97$

- $105_{10} = ?_2$
- $11100011_2 = ?_{10}$
- $214_{10} = ?_2$

Hexadecimal

- Base 16
- Numbers 0 – 9 and A – F
- More compact than binary, each hexadecimal digit is equivalent to 4 bits
- $97_{10} = 0\ 1\ 1\ 0\ 0\ 0\ 0\ 1_2 = 61_{16}$

Converting from/to Hexadecimal

16	1
16^1	16^0

6	1

$(6*16^1) + (1*16^0) = 96 + 1 = 97$

- $105_{10} = ?_{16}$
- $3F_{16} = ?_{10}$
- $214_{10} = ?_{16}$

Short Cut

$84_{10} =$

$84 \ / \ 2$	$=$	42	rem 0
$42 \ / \ 2$	$=$	21	rem 0
$21 \ / \ 2$	$=$	10	rem 1
$10 \ / \ 2$	$=$	5	rem 0
$5 \ / \ 2$	$=$	2	rem 1
$2 \ / \ 2$	$=$	1	rem 0
$1 \ / \ 2$	$=$	0	rem 1

$\Rightarrow 1010100_2$

Signed Numbers in Binary

- Two's Compliment System

128	64	32	16	8	4	2	1
$-(2^7)$	2^6	2^5	2^4	2^3	2^2	2^1	2^0

  - Maximum: $2^7 - 1 = 127$
  - Minimum: $-(2^7) = -128$

Converting Signed Decimal to Binary

$-114_{10} = ?_2$

1) $114_{10} = 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0_2$

2) Invert each digit (NOT) : $1\ 0\ 0\ 0\ 1\ 1\ 0\ 1_2$

3) Add 1 : 1 0 0 0 1 1 1 0

128	64	32	16	8	4	2	1
$-(2^7)$	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	0	1	1	1	0

(1*-128) + (1*8) + (1*4) + (1*2) = -128 + 8 + 4 + 2 = -114

Arithmetic Operations

+	unary plus
−	unary minus
+	addition
−	subtraction
*	multiplication
/	division
%	modulo

example:

```
int x, y;
x = -5;
y = x * 7;
y = y + 3;
x = x * -2;
y = x / 19;
```

what are x and y equal to?

modulo means "remainder after integer division"

Increment & Decrement Operator

- increment operator: ++
  meaning: add one and assign
  example: `i++;`
  is the same as: `i = i + 1;`

- decrement operator: --
  meaning: subtract one and assign
  example: `i--;`
  is the same as: `i = i - 1;`

Assignment Operators

operator	*meaning*	*example*
+=	add and assign	`i += 3;` is the same as: `i = i + 3;`
-=	subtract and assign	`i -= 3;` is the same as: `i = i - 3;`
*=	multiply and assign	`i *= 3;` is the same as: `i = i * 3;`
/=	divide and assign	`i /= 3;` is the same as: `i = i / 3;`
%=	modulo and assign	`i %= 3;` is the same as: `i = i % 3;`