COMP 4901Q: High Performance Computing (HPC)

Lecture 3: Revisit C/C++ Programming and Common Linux Commands

Instructor: Shaohuai SHI (shaohuais@cse.ust.hk)
Teaching assistants: Mingkai TANG (mtangag@connect.ust.hk)
Yazhou XING (yxingag@connect.ust.hk)

Course website: https://course.cse.ust.hk/comp4901q/

Outline

- C/C++ Programming: Basic Syntax
- Common Linux Commands

High-level Programming Languages in HPC

- Fortran
  - Since 1954 for HPC
  - Good for handling arrays
  - Less flexible in complex and highly dynamic data structure
- **C/C++**
  - Comparable performance with Fortran
  - More flexible
- Java
  - May have performance issues

C/C++ Hello World

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

int main() {
   cout << "Hello World" << endl; // prints Hello World
   return 0;
}
```

Data Types

- Integer types
- Floating-point types
- The **void** type: no value is available

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Type	Storage size	Value range	Precision
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal places
double	8 bytes	2.3E-308 to 1.7E+308	15 decimal places
long double	10 bytes	3.4E-4932 to 1.1E+4932	19 decimal places

Data Types

- **typedef** Declarations
  - typedef type newname;

```
typedef int feet;
feet distance;
```

- Enumerated Types
  - enum enum-name { list of names } var-list;

```
enum color { red, green, blue };
color c;
```

Variables

- **type** identifier**;**
  - "identifier" **is called a variable**
    - Only the following characters may appear in an identifier
      - 0-9, a-z, A-Z, _
    - The first character cannot be a digit (0–9)
    - Keyword (reserved words) are not allowed
    - Case-sensitive
  - **type** must be a valid data type
    - E.g., char, int, float, …

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

reserved words

- A variable is a named memory location
- Declaration, definition, and initialization
  - int a = 10;

```
// Variable declaration:
extern int a, b;
extern int c;
extern float f;
```

```
// Variable definition:
int a, b;
int c;
float f;
```

```
// Variable initialization:
a = 10;
b = 20;
c = a + b;
```

Constants

- Fixed values that the program may not alter
  - Integer
    - 212 // Legal
    - 215u // Legal
    - 0xFeeL // Legal
    - 078 // Illegal: 8 is not an octal digit
    - 032UU // Illegal: cannot repeat a suffix
  - Floating-point
    - 3.14159 // Legal
    - 314159E-5L // Legal
    - 510E // Illegal: incomplete exponent
    - 210f // Illegal: no decimal or exponent
    - .e55 // Illegal: missing integer or fraction
  - Boolean
    - true or false
  - Character
    - in single quotes, e.g., 'x'
  - String
    - are enclosed in double quotes, e.g., "hello, hpc"
    - 85 // decimal
    - 0213 // octal
    - 0x4b // hexadecimal
    - 30 // int
    - 30u // unsigned int
    - 30l // long
    - 30ul // unsigned long
- Defining constants
  - #define preprocessor:
    - #define identifier value

```
#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'
```

  - const keyword
    - const type variable = value;

```
const int  LENGTH = 10;
const int  WIDTH  = 5;
const char NEWLINE = '\n';
```

Operators

- Assignment: =

```
int A = 10;
int B = 20;
int c;
c = A + B;
c = c * A;
```

- Arithmetic operators

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

- Relational operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Operators cont.

- Logical Operators

```
bool A = true;
bool B = false;
```

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

- Bitwise Operators
  - perform bit-by-bit operation

```
int A = 60; //A = 0011 1100
int B = 13; //B = 0000 1101
```

```
A       = 0011 1100
B       = 0000 1101
----------------
A&B     = 0000 1100
A|B     = 0011 1101
A^B     = 0011 0001
~A      = 1100 0011
```

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Operators cont.

- Operators with assignment operator
  - B op= A is equivalent to B = B op A;
  - For example
    - B += A is equivalent to B = B + A;
    - B /= A is equivalent to B = B / A;
    - B |= 2 is equivalent to B = B | 2;
- Operators precedence
  - determines the grouping of terms in an expression

```
int A = 60;
int B = 13;
int C = 5;
int D = 15;
int E;
E = A + B * C / D;
E = (A + B) * C / D;
E = (A + B) * (C / D);
E = A++;
E = ++A;
```

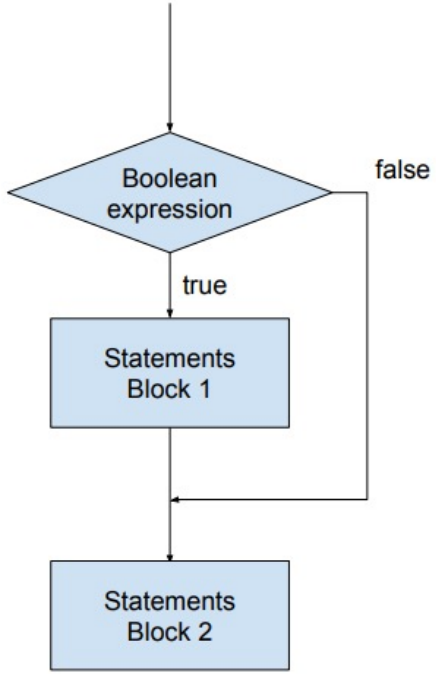
Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Expression and Statement

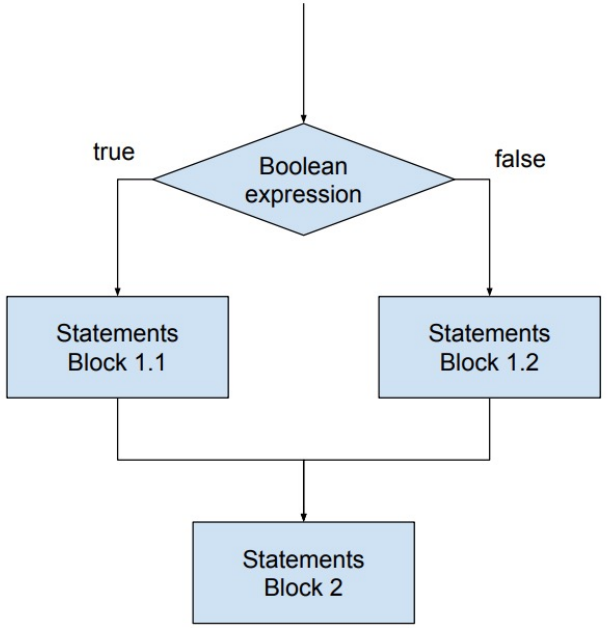
- Expression
  - has a value which is the result of some operation(s) on its(theirs) operands
  - Examples
    - 4
    - x - y
    - !x
- Statement
  - a sentence that acts as a command
  - It does not have a value
  - Ends in a “;”
  - Examples
    - Input statement: cin >> x;
    - Output statement: cout << x;
    - Assignment statement: x = 5;
    - Variable definition: int x;

Flow Control

- **if** statement
  - if (boolean expression) statement;
  - if (boolean expression) { statements; }



- **if-else** statement
  - if (boolean expression) statement;
    else statement;
  - if (boolean expression) {
    statements;
    } else { statements; }



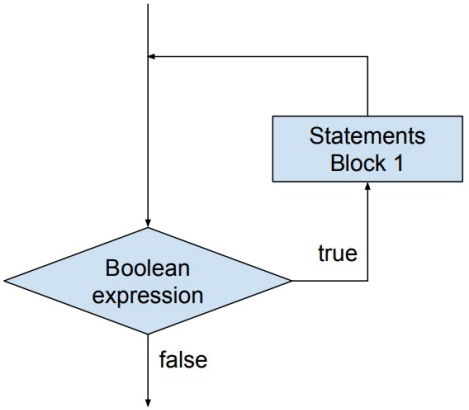
- **if-else-if** statement
  - if (boolean exp1) state;
    else if (boolean exp2) state;
    .
    .
    else state;

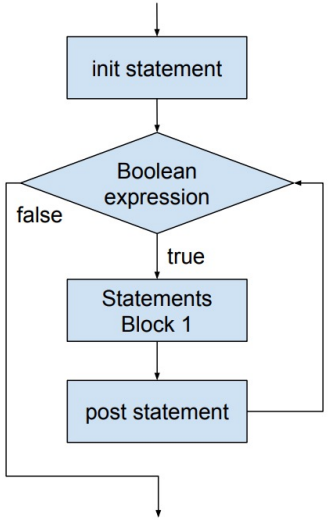
- The **? :** Operator
  - is equivalent to if-else
  - boolean exp1 ? exp2 : exp3;
  - The same as
    - if (exp1) exp2; else exp3;

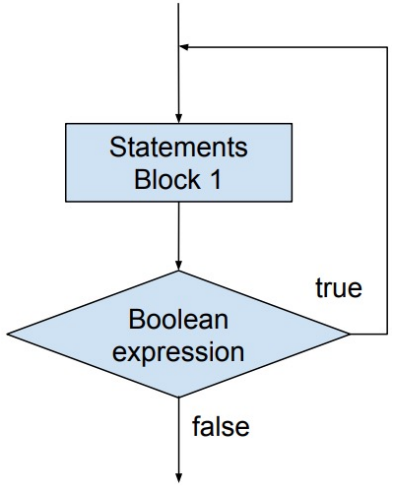
Loops

- Loop statements
  - while (boolean expression) { statements; }
  - for(init statement; boolean expression; post statement) { statements; }
  - do { statements; } while (boolean expression);

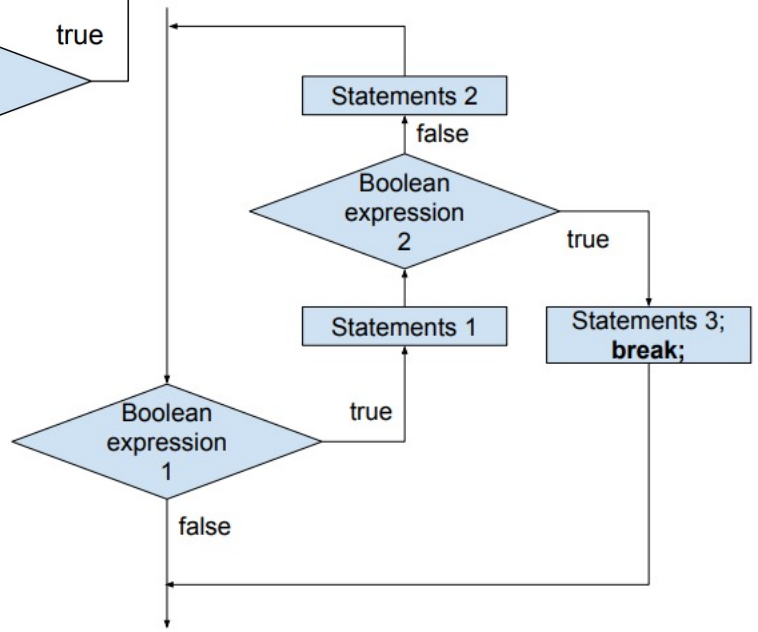
No	Loop Type & Description
1	while loopRepeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loopExecute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loopLike a 'while' statement, except that it tests the condition at the end of the loop body.

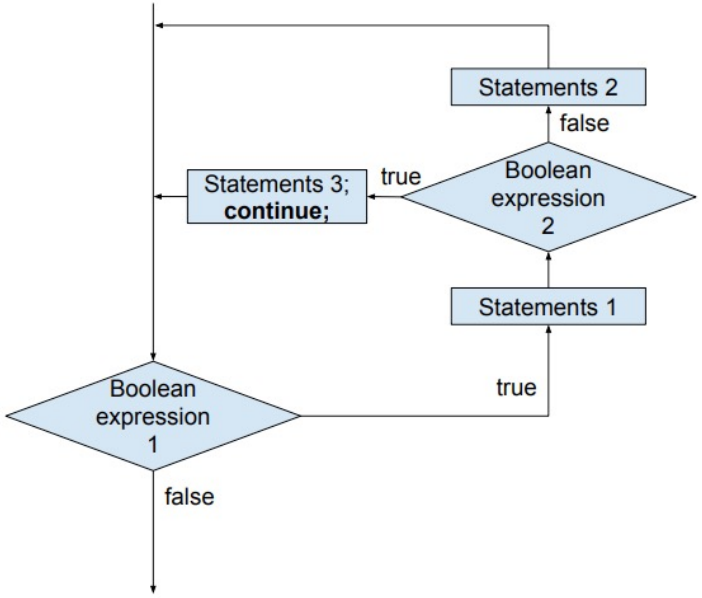






- Loop control statements
  - **break**
    - Exit the loop immediately
  - **continue**
    - Go to the next iteration's boolean expression.





Functions

- Function declaration
  - tells the compiler about a function's name, return type, and parameters.
  - return_type function_name(formal parameter list);
- Function definition
  - provides the actual body of the function
  - return_type function_name(formal parameter list) {

    body of the function

    }
- Function call
  - function_name(actual parameter list);
- **void** as the return type of a function
  - return statement can be avoided.
- **void** as the formal parameter list of a function
  - no arguments are needed when calling a function

```
int max(int num1, int num2);

int max(int num1, int num2) {
    int result;
    if (num1 > num2) result = num1; else result = num2;
    return result;
}

int main () {
    int a = 10;
    int b = 12;
    int larger;
    larger = max(a, b);
    return 0;
}
```

Function Arguments

- When calling a function, there are three ways that arguments can be passed to a function
  - Call by value
    - The value should be copied
    - Changes inside the function to the parameter have no effect on the argument
  - Call by reference
    - A reference variable is an alias of anther variable
    - Copy the reference of argument instead of copying values
    - Changes made to the parameter affect the argument.
  - Call by pointer
    - Copy the address of an argument into the formal parameter
    - The address is used to access the actual argument
    - Changes made to the parameter affect the argument.

```
int max1(int num1, int num2) {
    if (num1 > num2) num1 = num1; else num1 = num2;
    return num1;
}

int max2(int& num1, int& num2) {
    if (num1 > num2) num1 = num1; else num1 = num2;
    return num1;
}

int max3(int* num1, int* num2) {
    if (*num1 > *num2) *num1 = *num1; else *num1 = *num2;
    return *num1;
}


int main () {
    int a = 10, b = 12;
    max1(a, b);
    max2(a, b);
    max2(&a, &b);
    return 0;
}
```

Arrays

- Definition of a 1D array
  - **type arrayName [arraySize];**
  - arraySize should be a positive constant or constant expression
  - Examples
    - double balance[10];
    - double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
    - double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0}; //omit the size of the array
  - Successive elements are stored in contiguous memory
- Access
  - Use subscript operator: [] with an array index
  - The index is from 0, 1, 2, …, N-1 for an N-element array
- Multi-dimensional array
  - **type name[size1][size2]…[sizeN];**

```
int a[3][4] = { {0, 1, 2, 3} , /* initializers for row indexed by 0 */
                {4, 5, 6, 7} , /* initializers for row indexed by 1 */
                {8, 9, 10, 11} /* initializers for row indexed by 2 */ };
```

String

- The C-Style Character String
  - For a string of length N, add ’\0’ as the (N + 1)th element of its char array
  - char arrays
    - char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  - double quotes
    - char greeting[] = "Hello";

- The String Class in C++
  - #include <string>
  - string str1 = "Hello";

Pointers

- Every variable is a memory location and every memory location has its address
- The ampersand (&) operator denotes an address in memory
- A **pointer** is a variable whose value is the address of another variable
  - **type *var-name;**
- Pointer to Pointer
  - **type **var-name;**


```
int sum = 255;
short int age = -1;
double average = 4.45015e-308;
int *ptrSum = &sum;
```

Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00	sum	int (4 bytes)	000000FF(255_{10})
90000001	00			
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF(-1_{10})
90000005	FF			
90000006	1F	averge	double (8 bytes)	1FFFFFFFFFFFFFFF ($4.45015E-308_{10}$)
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal

Source: https://astrocoke.tistory.com/32

C++ Classes and Objects

- Class
  - define a blueprint for a data type

Sr.No	Concept & Description
1	Class Member Functions: A member function of a class is a function that has its definition or its prototype within the class definition like any other variable.
2	Class Access Modifiers: A class member can be defined as public, private or protected. By default members would be assumed as private.
3	Constructor & Destructor: A class constructor is a special function in a class that is called when a new object of the class is created. A destructor is also a special function which is called when created object is deleted.

```
class Box {
  public:
    double length;        // Length of a box
    double breadth;       // Breadth of a box
    double height;        // Height of a box
    Box() { length = 1; } // Constructor
    ~Box() {}             // Destructor
    double getVolume(void) {
      return length * breadth * height;
    }
};
```

Files and Streams

- **iostream** standard library
  - **cin**: standard input
  - **cout**: standard output
- **fstream**
  - **ofstream**: output file stream
  - **ifstream**: input file stream
  - **fstream**: represents the file stream, and has the capabilities of both ofstream and ifstream
- Open/close a File
  - void open(const char *filename, ios::openmode mode);

Sr.No	Mode Flag & Description
1	**ios::app:** Append mode. All output to that file to be appended to the end.
2	**ios::ate:** Open a file for output and move the read/write control to the end of the file.
3	**ios::in:** Open a file for reading.
4	**ios::out:** Open a file for writing.
5	**ios::trunc:** If the file already exists, its contents will be truncated before opening the file.

  - void close();

C++ Multithreading

- Multithreading for multitasking that allows your computer to run two or more programs concurrently
- Usage
  - **#include <pthread.h>**
  - **pthread_create (thread, attr, start_routine, arg)**
    - **thread:** an opaque, unique identifier for the new thread
    - **attr**: an opaque attribute object that may be used to set thread attributes
    - **start_routine**: the C++ routine that the thread will execute once it is created
    - **arg**: Aa single argument that may be passed to start_routine
  - **pthread_exit (status)**

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
using namespace std;
#define NUM_THREADS 5
struct thread_data {
  int  thread_id;
  char *message;
};
void *PrintHello(void *threadarg) {
  struct thread_data *my_data;
  my_data = (struct thread_data *) threadarg;
  cout << "Thread ID : " << my_data->thread_id ;
  cout << " Message : " << my_data->message << endl;
  pthread_exit(NULL);
}

int main () {
  pthread_t threads[NUM_THREADS];
  struct thread_data td[NUM_THREADS];
  int rc, i;
  for( i = 0; i < NUM_THREADS; i++ ) {
    cout <<"main() : creating thread, " << i << endl;
    td[i].thread_id = i;
    td[i].message = "This is message";
    rc = pthread_create(&threads[i], NULL, PrintHello, (void *)&td[i]);
    if (rc) {
      cout << "Error:unable to create thread," << rc << endl;
      exit(-1);
    }
  }
  pthread_exit(NULL);
}
```

Common Linux Commands

cmd	Description
ping	Check the connectivity status to a server
ssh	Login a remote server
scp	Copy files from/to a server

cmd	Description
pwd	Path of the current directory
cd	Go to a directory $cd folder; $cd - $cd $cd ..
ls	View the contents of a directory

cmd	Description
cat	List the contents of a file
cp	Copy files $cp file.txt file-copy.txt; $cd -r folder folder-copy;
mv	Move files
mkdir	Create a new directory $mkdir new-directory $mkdir -p d1/d2/d3
rm	Remove files $rm file.txt $rm -r d1
touch	Create a blank new file
find	Searches for files and directories $find . -name notes.txt $find / -type d -name notes
grep	search through all the text in a given file $grep blue notepad.txt

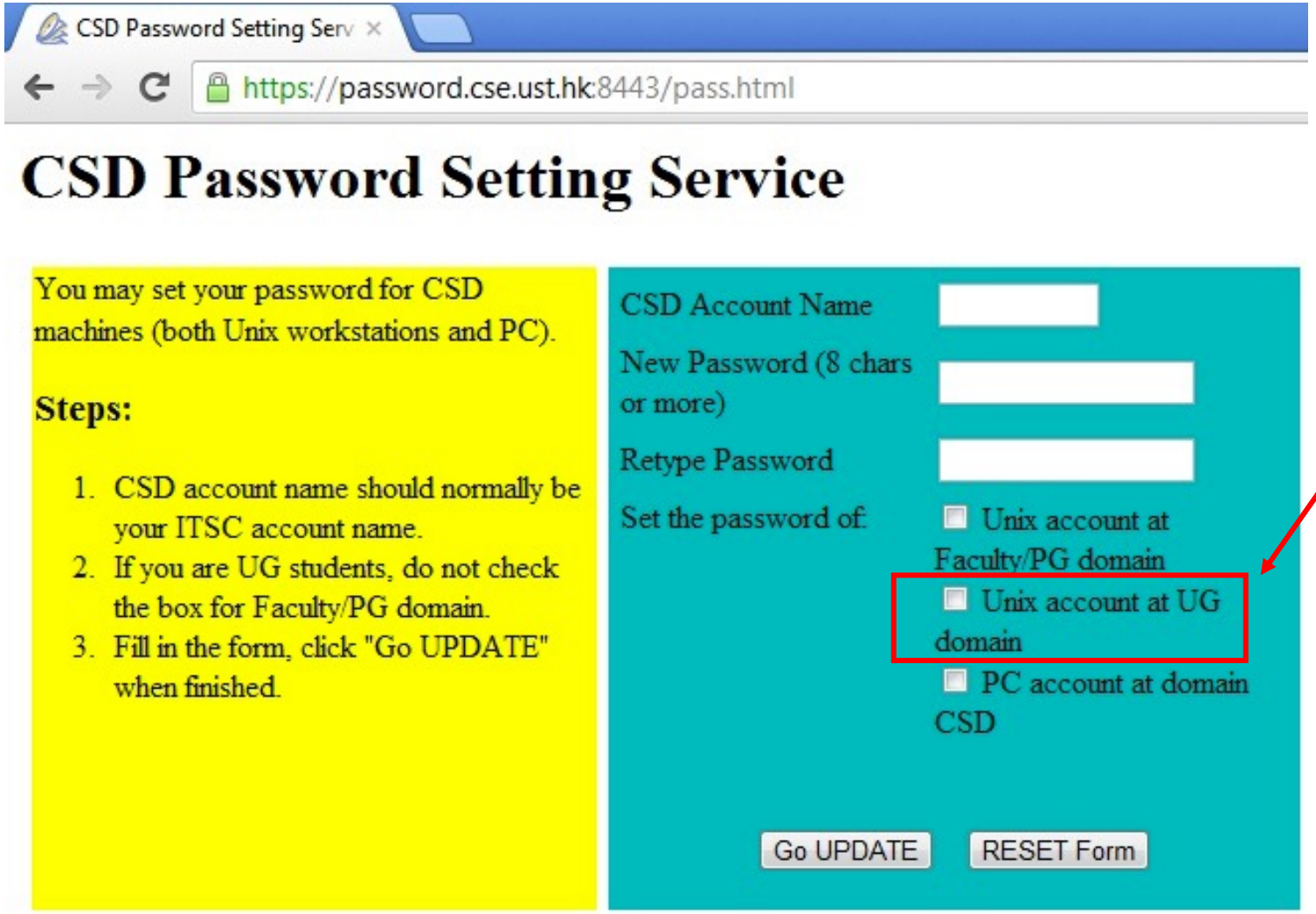
cmd	Description
kill	Terminate a process
wget	Download files
top	Display a list of running processes
lscpu	Check CPU information
df	Check disk information $df -h
ifconfig	Display network configuration
free	Display system memory $free -h
vi	Default editor of Unix/Linux systems

Ubuntu is easy to use!

http://mally.stanford.edu/~sr/computing/basic-unix.html

Access Our GPU Cluster

Follow the guide: https://cssystem.cse.ust.hk/UGuides/activation.html



Remember to check this!

Summary

- **C/C++ programming basics**
- **Common Linux commands**

Practice makes perfect!