CS2311 Computer Programming

LT9: Pointers

Part I

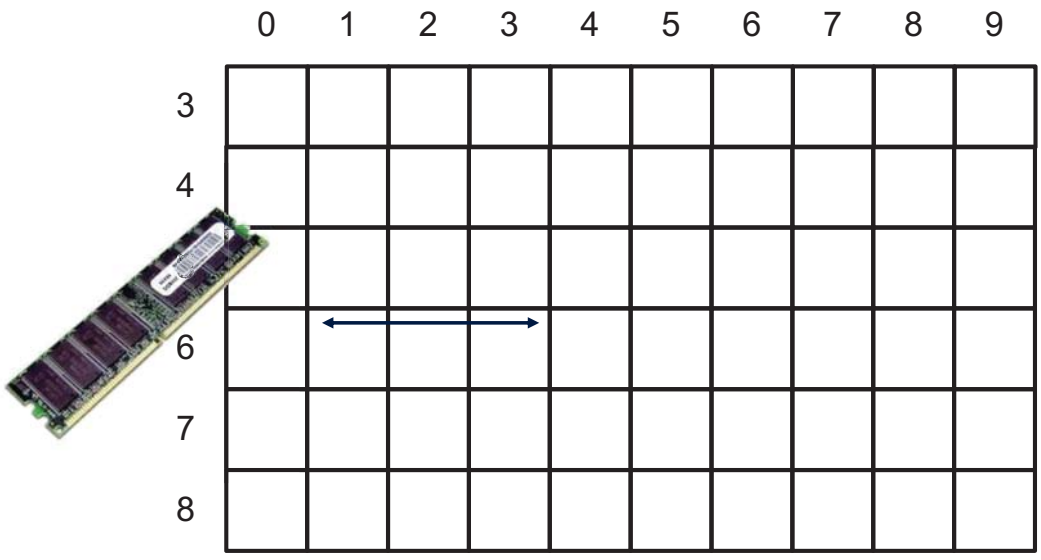
Outline

- Memory and variables
- Pointers and pointer operations
- Call by reference

Memory and variables

- A variable is used to store data that will be accessed by a program on execution.
- Normally, a variable is stored in the main memory
- A variable has four attributes:
  - Value - the content of the variable
  - Identifier - the name of the variable
  - Address - the memory location of the variable
  - Scope - the accessibility of the variable

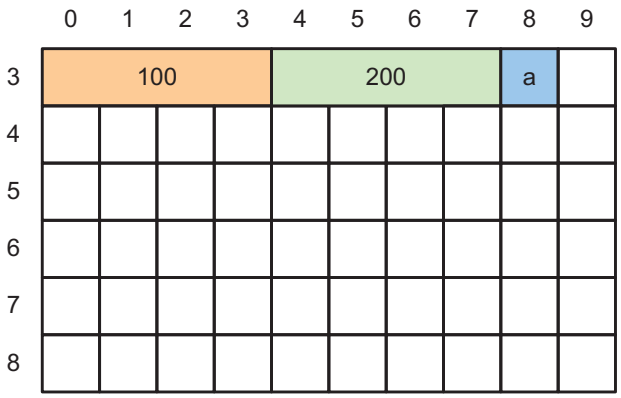
Main Memory



Variable and Memory

```
int main() {
    int x;
    int y;
    char c;
    x = 100;
    y = 200;
    c = 'a';

    return 0;
}
```



Identifier	Value	Address
x	100	30
y	200	34
c	'a'	38

Variable and Memory

- *Most of the time*, the computer allocates adjacent memory locations for variables declared one after the other.
- A variable's address is the **first byte** occupied by the variable.
- Address of a variable depends on the computer, and is usually in hexadecimal (base 16 with values 0-9 and A-F).
  - e.g. `0x00023AF0, 00023AF0`

Pointers

- A **pointer** is a variable which stores the address of another variable, i.e. it points to the variable.

- A pointer, like a regular variable, has a type. Its type is determined by the type of the variable it points to.

Variable type	`int`	`float`	`double`	`char`
Pointer type	int*	float*	double*	char*

* and & operators

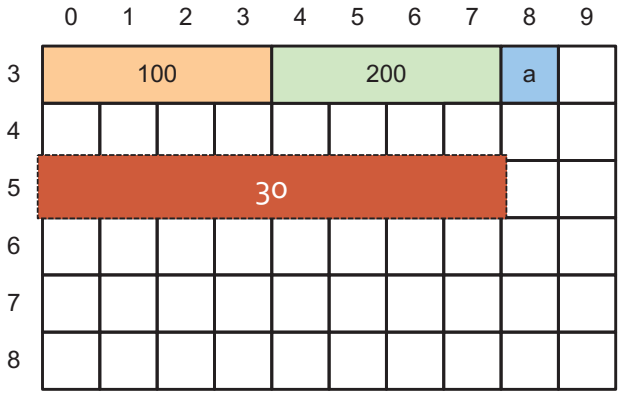
- To declare a pointer, use "*" before an identifier name or after the type:
  - **char *cPtr;** // a character pointer
  - **int *nPtr;** // an integer pointer
  - **float * fp;** // this is also okay, a floating point pointer

- To retrieve the address of a variable, use the "&" operator (referencing):
  - **int x;**
  - **nPtr = &x;** // &x returns the address of x;

- To access the variable a pointer pointing to, use "*" operator (dereferencing)
  - ***nPtr = 10;**
  - **y = *nPtr;**

Pointers

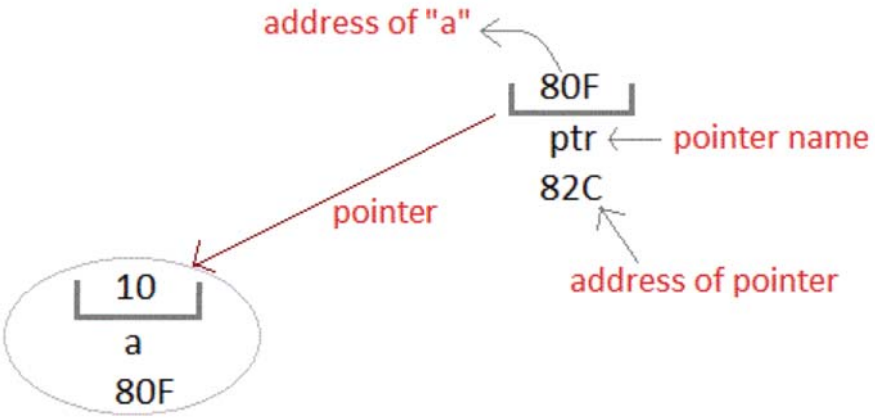
```
int main() {
    int x;
    int y;
    char c;
    int *p;
    x = 100;
    y = 200;
    c = 'a';
    p = &x;
    return 0;
}
```



Identifier	Value	Address
x	100	30
y	200	34
c	'a'	38

Demo

Pointers



Example

```
int x,y;                //x and y are integer variables
void main() {
    int *p1,*p2;        /*p1 and p2 are pointers of integer type */
    x = 10;
    y = 12;
    p1 = &x;            /* p1 stores the address of  variable x */
    p2 = &y;            /* p2 stores the address of variable y */
    *p1 = 5;            /* p1 value unchanged but x is updated to 5 */
    *p2 = *p1+10;       /* what are the values of p2 and y? */

    return 0;
}
```

Common operations

- Set a pointer **p1** point to a variable **x**

  p1 = &x;
- Set a pointer **p1** point to the variable pointed by another pointer **p2**

  p1 = p2;
- Update the value of variable pointed by a pointer

  ***p = 10;**
- Retrieve the value of variable pointed by a pointer

  int x = *p;

More examples

```
x = 3;
y = 5;
p1 = &x;
p2 = &y;
y = *p1 - *p2;
p1 = p2;
y = *p1 + *p2;
x = p1 + 1;
```

```
p2 = p1+2;
x = *p2 * *p1;
x = *p2 / *p1;
y = x * *p2;
y += *p1

*p2 += 3;
*p1 *= *p2
```

Guideline

- * operator will give the value of pointing variable
- & operator will give the address of a variable

Applications of pointer

- Call by Reference
- Fast Array Access
  - Will be covered in later class
- Dynamic Memory Allocation
  - Require additional memory space for storing value.
  - Similar to variable declaration but the variable is stored outside the program.

Call by reference

- Pass the address of a variable to a function
- To update a variable provided by a caller as call by value cannot be used to update arguments to function
- Consider the following function

Call by value

```
void f (char c1_in_f) {
    c1_in_f = 'B';     //c1_in_f=66
}

int main() {
    char c1_in_main = 'A';   // c1_in_main =65
    f(c1_in_main);
    cout << c1_in_main;      //print 'A'
    return 0;
}
```

Call by value

- When calling **f()**, the value of **c1 (65 or 'A')** is assigned to **c1_in_f**
- Inside **f()**, the value of **c1_in_f** is changed to 66 or 'B'.
- **c1_in_main** remains unchanged (65 or 'A').
- In call by value, there is no way to modify **c1_in_main** inside **f()**, unless we use the return statement.

Call by reference

```
void f(char *c1_ptr) {
    *c1_ptr = 'B';
}

int main() {
    char c1_in_main = 'A'; // c1_in_main =65
    f(&c1_in_main);
    cout << c1_in_main;    // print 'B'
    return 0;
}
```

When **f()** is called, the following operation is performed

c1_ptr = **&c1_in_main**;

First, the value **&c1_in_main** (which is 3A8E) is retrieved

Call by reference

```
void f (char *c1_ptr) {
    *c1_ptr = 'B';
}

void main() {
    char c1_in_main = 'A';     // c1_in_main =65
    f(&c1_in_main);
    cout << c1_in_main;        // print 'B'
}
```

points to

Variable	Variable type	Memory location	Content
c1_in_main	char	3A8E	65
c1_ptr	char pointer	4000	3A8E

Assign location 3A8E to c1_ptr

Location of c1_in_main (location 3A8E) is assigned to c1_ptr1
c1_ptr = **&c1_in_main**;

Call by reference

```
void f (char *c1_ptr) {
    *c1_ptr = 'B';
}

void main() {
    char c1_in_main='A'; // c1_in_main = 65
    f(&c1_in_main);
    cout << c1_in_main; // print 'B'
}
```

update

c1_ptr points to location 3A8E (that is the variable c1_in_main). *c1_ptr refers to the variable pointed by c1_ptr, i.e. the variable stored at 3A8E

Variable	Variable type	Memory location	Content
c1_in_main	char	3A8E	**66**
c1_ptr	char pointer	4000	3A8E

c1_ptr = 'B'; //error
Reason: c1_ptr1 stores a location so it cannot store a char
(or the ASCII code of a char)

Note the different meaning of *

The type of c1_ptr is **char*** (pointer to star)

dereference a pointer

```
void f (char *c1_ptr) {
    *c1_ptr = 'B';
}

void main() {
    char c1_in_main = 'A';   // c1_in_main =65
    f(&c1_in_main);
    cout << c1_in_main;      // print 'B'
}
```


Exercise: what are the errors?

```
int x = 3;
char c = 'a';
char *ptr;
ptr = &x;
ptr = c;
ptr = &c;
```

Answer

```
int x = 3;
char c = 'a';
char *ptr;
ptr = &x;      // error: ptr can only points to a char, but not int
ptr = c;       // error: cannot assign a char to a pointer.
               // A pointer can only store a location
ptr = &c;      // correct
```


Exercise: What is the output?

```
int num = 100;
int *ptr1;

ptr1 = &num;
*ptr1 = 40;
cout << num;
```

Answer

```
int num = 100;
int *ptr1;

ptr1 = &num;
*ptr1 = 40;
cout << num; // print 40
```


Call by value and call by reference

- In call by value, only a single value can be returned using a **return statement**
- In call by reference, the argument(s) can be a pointer which may reference or points to the variable(s) in the caller function
  - More than one variables can be updated, achieving the effect of returning multiple values.

Call-by-Value

```
int add(int m, int n) {
    int c;
    c = m + n;
    m = 10;
    return c;
}

void main() {
    int a = 3, b = 5, c;
    c = add(a,b);
    cout << a << " " << c << endl;
}
```

Pass value as parameter.

Call-by-Reference

```
int add(int *m, int *n) {
    int c;
    c = *m + *n;
    *m = 10;
    return c;
}

void main() {
    int a = 3, b = 5, c;
    c = add(&a, &b);
    cout << a << " "<< c << endl;
}
```

Pass address as parameter.

Call by Reference - Guideline

- Add the '*' sign to the function parameters that store the variable call by reference.

  E.g.

```
void cal(int *x, int y) {
    //x is call by reference
    //y is call by value
    ......
    .......*x;
}
```

- Add the '&' sign to the variable when it needs to be call by reference.

  E.g.

```
cal(&result, factor);
```

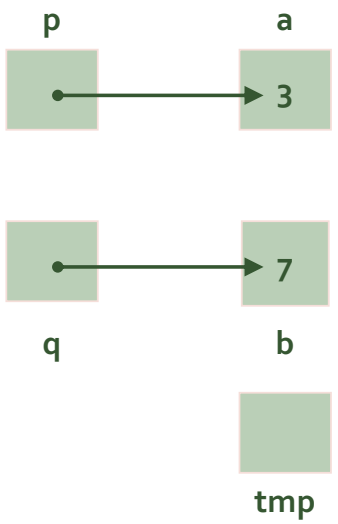
Example: swapping values

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

void swap(int *p, int *q) {
  int   tmp;

  tmp = *p;         /* tmp = 3 */
  *p = *q;          /* *p = 7 */
  *q = tmp;         /* *q = 3 */
}

int main(void) {
  int   a = 3, b = 7;
  swap(&a, &b);
  cout << a <<" "<< b << endl;  /* 7  3 is printed */
  return 0;
}
```



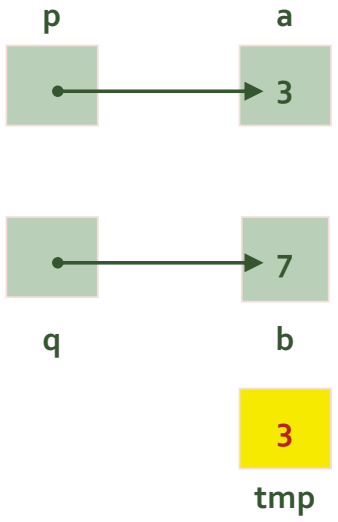
Example: swapping values

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

void swap(int *p, int *q) {
  int   tmp;

  tmp = *p;         /* tmp = 3 */
  *p = *q;          /* *p = 7 */
  *q = tmp;         /* *q = 3 */
}

int main(void) {
  int   a = 3, b = 7;
  swap(&a, &b);
  cout << a <<" "<< b << endl;  /* 7  3 is printed */
  return 0;
}
```



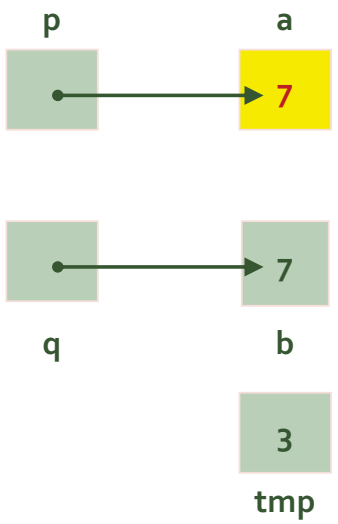
Example: swapping values

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

void swap(int *p, int *q) {
  int   tmp;

  tmp = *p;         /* tmp = 3 */
  *p = *q;          /* *p = 7 */
  *q = tmp;         /* *q = 3 */
}

int main(void) {
  int   a = 3, b = 7;
  swap(&a, &b);
  cout << a <<" "<< b << endl;  /* 7  3 is printed */
  return 0;
}
```



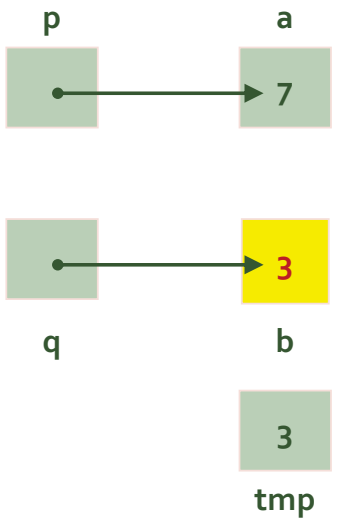
Example: swapping values

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

void swap(int *p, int *q) {
  int   tmp;

  tmp = *p;         /* tmp = 3 */
  *p = *q;          /* *p = 7 */
  *q = tmp;         /* *q = 3 */
}

int main(void) {
  int   a = 3, b = 7;
  swap(&a, &b);
  cout << a <<" "<< b << endl;  /* 7  3 is printed */
  return 0;
}
```



Summary

- Pointer is a special variable used to store the memory address (location) of another variable.
- Pointer is typed, and its type is determined by the variable it pointing to.
- * operator has two meaning
  - For declaration, e.g int *p1, char *pc;
  - For dereference, e.g. x=*p1, *pc='b';
- & operator return the address of any variable

Summary

- Call by reference: pass the address of a variable to a function such that its content can be updated within the function.
- For a function returns one value only, we can use a return statement.
- For a function returns more than one value, one must use call by reference.