§9 Pointers

ENGG1111

Computer Programming and Applications

Dirk Schnieders

Outline

- Memory Address and Address-of Operator
- Pointer
  - Declaration
  - Compatibility
  - Dereferencing
- Pointers and Structs
- NULL
- Dynamic Variables
- Dangling Pointers
- Pointers and Functions
- Pointers and Arrays
- Dynamically Allocated Arrays

Memory Address

- The main memory is a collection of memory locations (or memory cells)
- Each memory location has a unique address
- Thus, every variable you declare in your C++ program has
  - a value
  - an address

Address	Memory cell
…	
…bb	
…bc	
…bd	
…be	
…bf	
…c0	
…c1	
…c2	
…	

Address-of Operator

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    bool a;
5    int b;
6    double c;
7    cout << &a << endl;
8    cout << &b << en
9    cout << &c << endl;
10 }
```

```
0x7ffcd4d135bb
0x7ffcd4d135bc
0x7ffcd4d135c0
Press any key to continue...
```

Address	Memory cell
…	
…bb	
…bc	
…bd	
…be	
…bf	
…c0	
…c1	
…c2	
…	

Address-of operator & returns the address of a variable

Pointer - Declaration

- A pointer is a variable that stores the address
- A pointer “points” to a variable by telling where the variable is in the main memory
- A pointer can be declared as follows

```
type *variableName;
```

Pointer - Declaration

- Example

```
1   #include <iostream>
2   using namespace std;
3   struct student {
4     string name;
5     double assignmentMark;
6   };
7   int main() {
8     int *a;
9     double *b;
10    student *c;
11  }
```

Pointer - Compatibility

- Type compatibility
  - E.g., an int pointer variable can only store the address of an int variable.

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 5;
5      int *p;
6      p = &x;
7  }
```

p is a pointer variable that stores an int address, we need to assign the address of an int variable to it (i.e., &x)

Pointer - Compatibility

- Type compatibility
  - E.g., an int pointer variable can only store the address of an int variable.

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 5;
5      double *p;
6      p = &x;
7  }
```

gcc-make-run: Compile Error

```
pointers05.cpp: In function ‘int main()’:
pointers05.cpp:6:9: error: cannot convert ‘int*’ to
‘double*’ in assignment
    p = &x;
        ^
```

Pointer - Dereferencing

- To access the memory cell of an address we can use the dereferencing operator *

```
*pointer
```

address

Memory cell of address

Address	Memory cell
…	
…bb	
…bc	
…bd	
…be	
…bf	
…c0	
…c1	
…c2	
…	

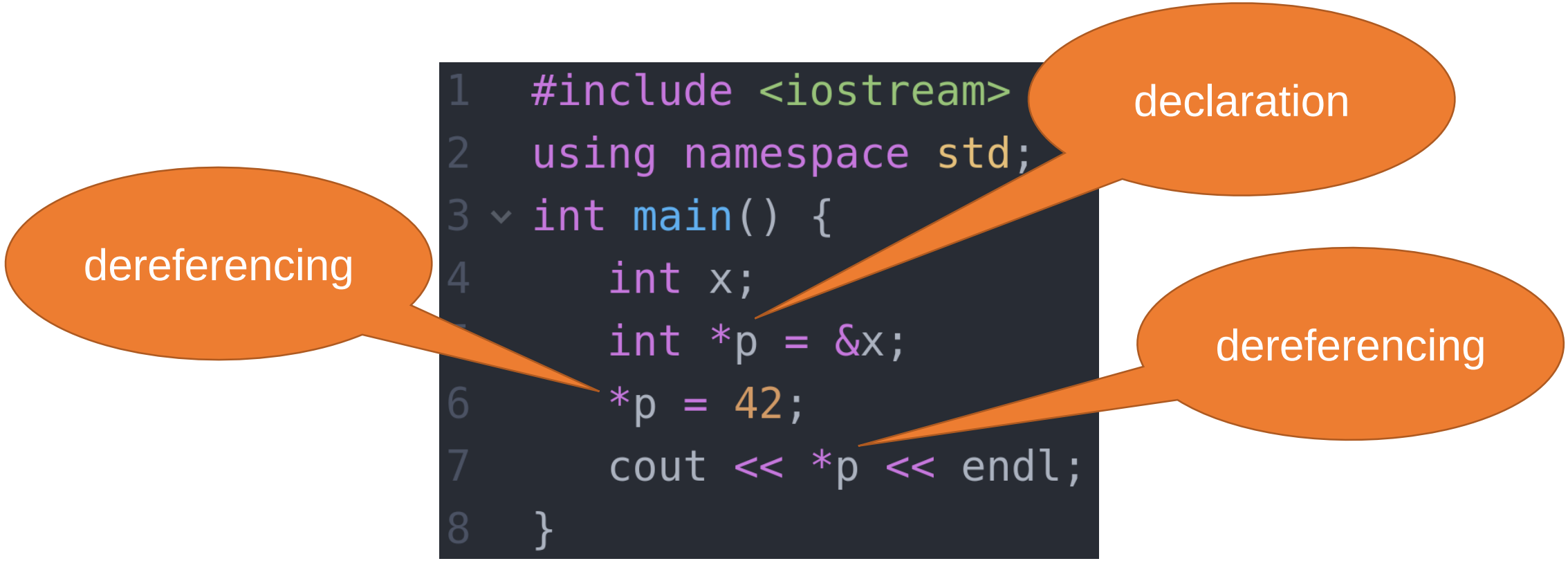
Pointer - Dereferencing

- Example

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 5;
5      int *p;
6      p = &x;
7      cout << *p << endl;
8  }
```

Pointer - Dereferencing

- Example



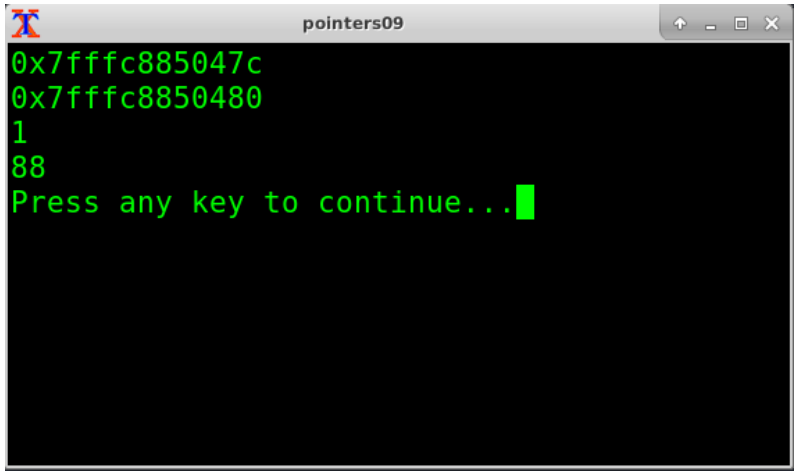
Pointer - Dereferencing

- Example

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x;
5      int *p = &x;
6      *p = 42;
7      cout << x << endl;
8  }
```

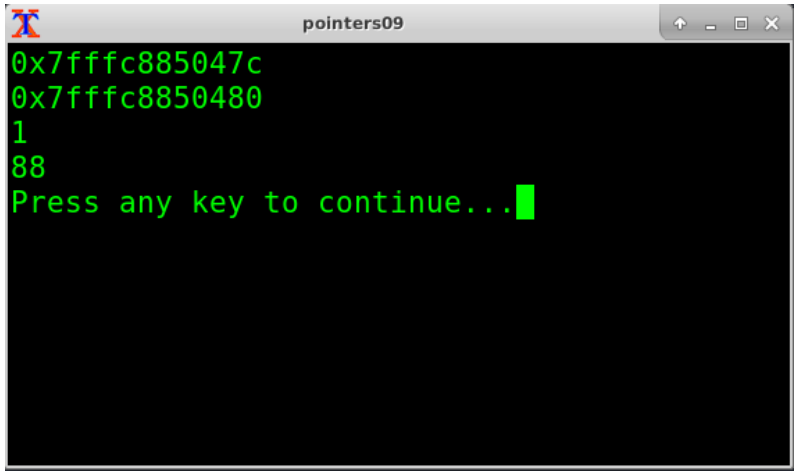
Dereferencing – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 1;
5      int *p;
6      p = &x;
7      cout << p << endl;
8      cout << &p << endl;
9      cout << *p << endl;
10     *p = 44 * 2;
11     cout << x << endl;
12 }
```



Dereferencing – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 1;
5      int *p;
6      p = &x;
7      cout << p << endl;
8      cout << &p << endl;
9      cout << *p << endl;
10     *p = 44 * 2;
11     cout << x << endl;
12 }
```



Address	Memory cell
…	
…7b	
…7c	
…7d	
…7e	
…7f	
…80	
…81	
…82	
…	

Dereferencing – Step by Step

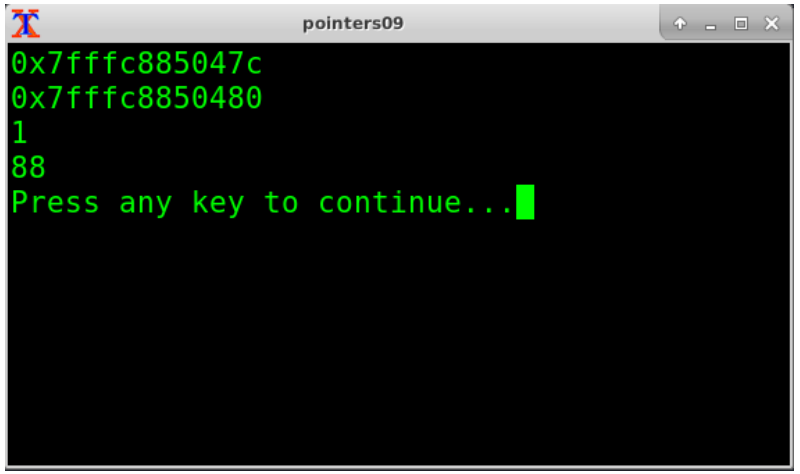
```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 1;
5      int *p;
6      p = &x;
7      cout << p << endl;
8      cout << &p << endl;
9      cout << *p << endl;
10     *p = 44 * 2;
11     cout << x << endl;
12 }
```

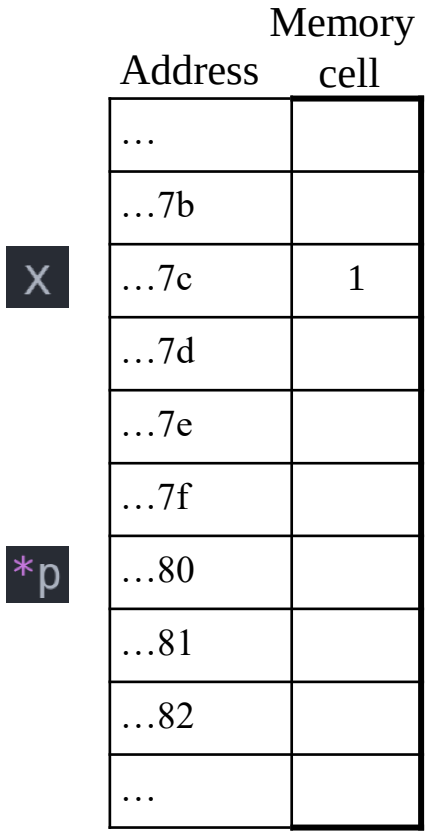
x

Address	Memory cell
…	
…7b	
…7c	1
…7d	
…7e	
…7f	
…80	
…81	
…82	
…	

Dereferencing – Step by Step

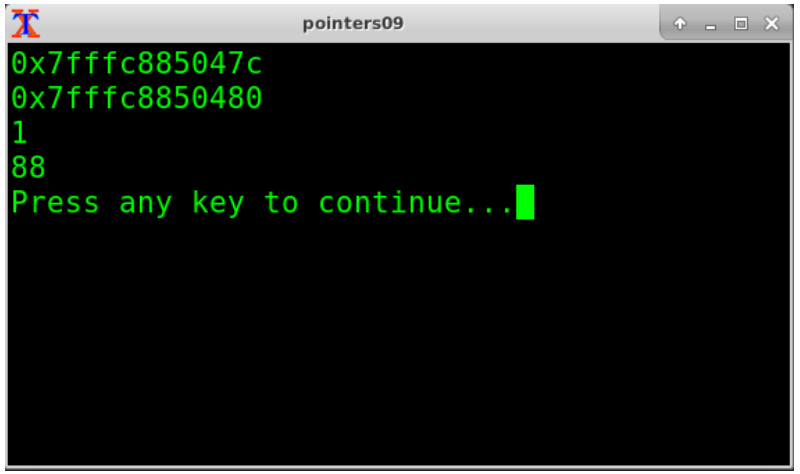
```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 1;
5      int *p;
6      p = &x;
7      cout << p << endl;
8      cout << &p << endl;
9      cout << *p << endl;
10     *p = 44 * 2;
11     cout << x << endl;
12 }
```





Dereferencing – Step by Step

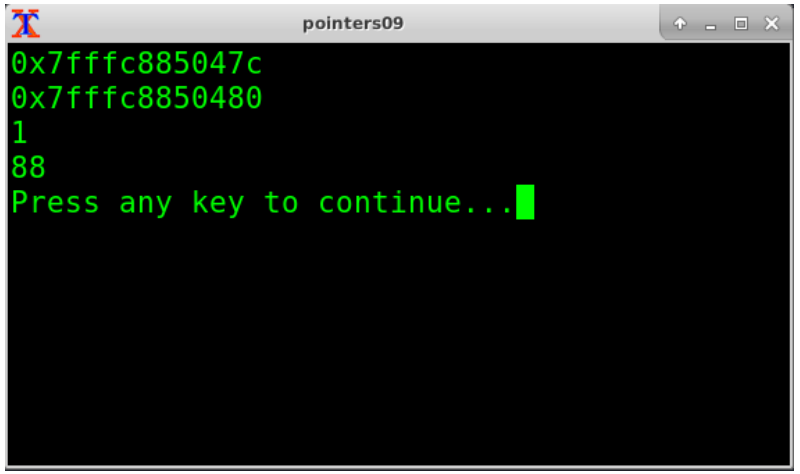
```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 1;
5      int *p;
6      p = &x;
7      cout << p << endl;
8      cout << &p << endl;
9      cout << *p << endl;
10     *p = 44 * 2;
11     cout << x << endl;
12 }
```



	Address	Memory cell
	…	
	…7b	
x	…7c	1
	…7d	
	…7e	
	…7f	
*p	…80	….7c
	…81	
	…82	
	…	

Dereferencing – Step by Step

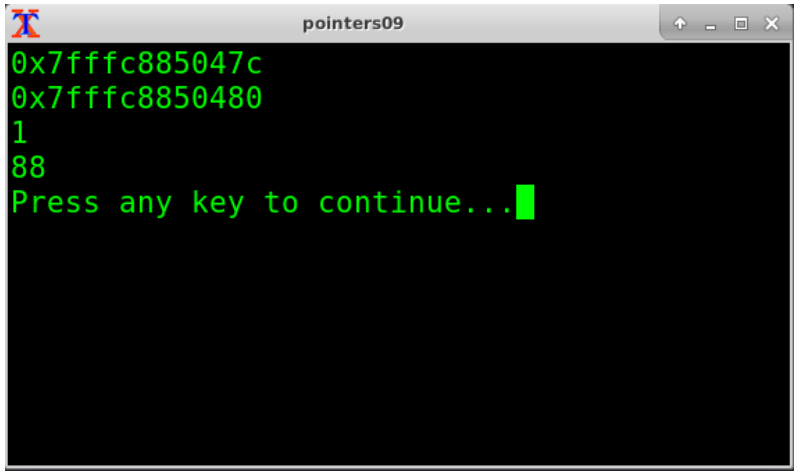
```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 1;
5      int *p;
6      p = &x;
7      cout << p << endl;
8      cout << &p << endl;
9      cout << *p << endl;
10     *p = 44 * 2;
11     cout << x << endl;
12 }
```



	Address	Memory cell
	…	
	…7b	
x	…7c	1
	…7d	
	…7e	
	…7f	
*p	…80	….7c
	…81	
	…82	
	…	

Dereferencing – Step by Step

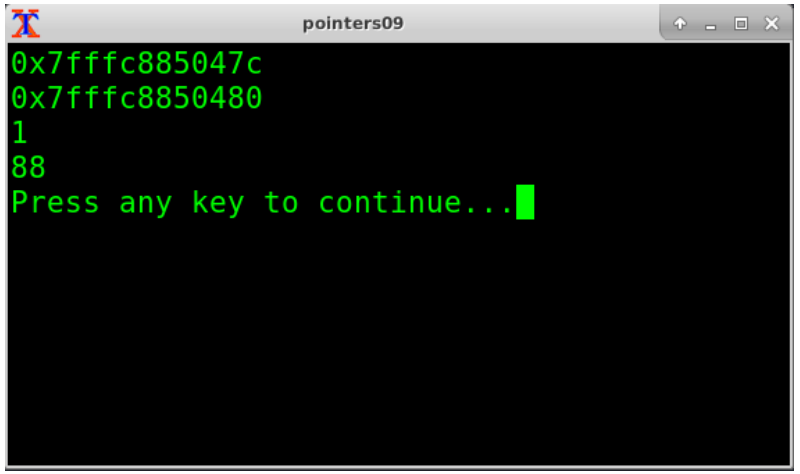
```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 1;
5      int *p;
6      p = &x;
7      cout << p << endl;
8      cout << &p << endl;
9      cout << *p << endl;
10     *p = 44 * 2;
11     cout << x << endl;
12 }
```



	Address	Memory cell
	…	
	…7b	
x	…7c	1
	…7d	
	…7e	
	…7f	
*p	…80	….7c
	…81	
	…82	
	…	

Dereferencing – Step by Step

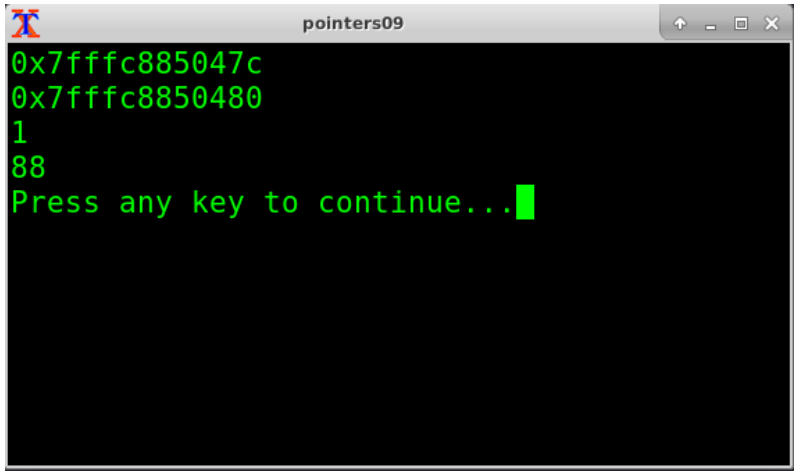
```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       int x = 1;
5       int *p;
6       p = &x;
7       cout << p << endl;
8       cout << &p << endl;
9       cout << *p << endl;
10      *p = 44 * 2;
11      cout << x << endl;
12  }
```



	Address	Memory cell
	…	
	…7b	
x	…7c	1
	…7d	
	…7e	
	…7f	
*p	…80	….7c
	…81	
	…82	
	…	

Dereferencing – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x = 1;
5      int *p;
6      p = &x;
7      cout << p << endl;
8      cout << &p << endl;
9      cout << *p << endl;
10     *p = 44 * 2;
11     cout << x << endl;
12 }
```



	Address	Memory cell
	…	
	…7b	
x	…7c	88
	…7d	
	…7e	
	…7f	
*p	…80	….7c
	…81	
	…82	
	…	

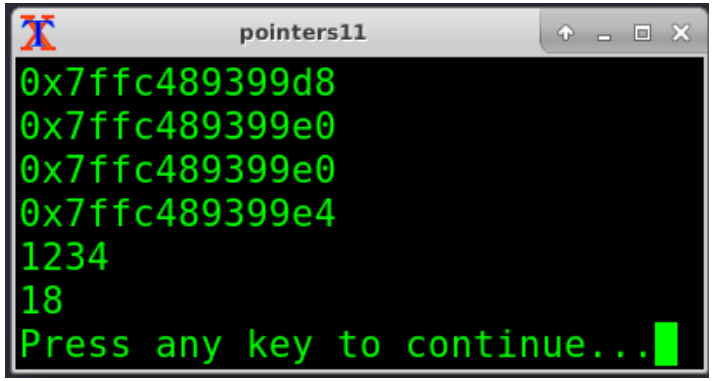
Task

- What is the output of the following program?

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int i = 25, *p1;
5      p1 = &i;
6      char c = 'h', *p2;
7      p2 = &c;
8      cout << *p1 << endl;
9      *p2 = 'a';
10     cout << c << endl;
11 }
```

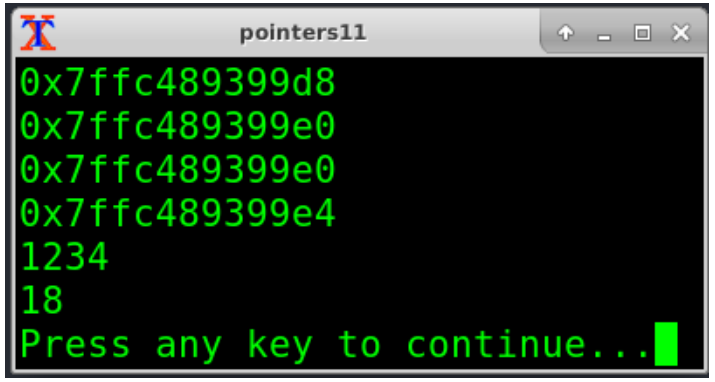
Pointers and Struct

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```



Pointers and Struct – Step by Step

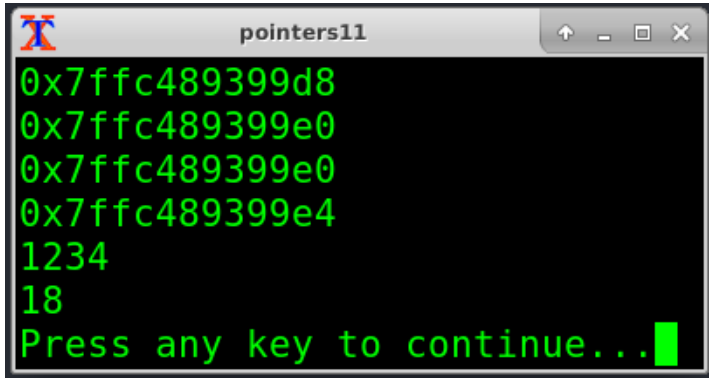
```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

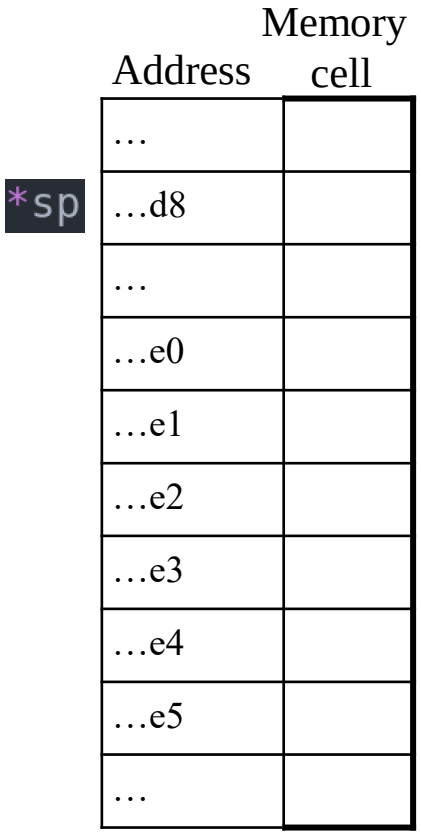


Address	Memory cell
…	
…d8	
…	
…e0	
…e1	
…e2	
…e3	
…e4	
…e5	
…	

Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```





Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

```
pointers11
0x7ffc489399d8
0x7ffc489399e0
0x7ffc489399e0
0x7ffc489399e4
1234
18
Press any key to continue...
```

	Address	Memory cell
	…	
*sp	…d8	
	…	
s UID	…e0	1234
	…e1	
	…e2	
	…e3	
age	…e4	90
	…e5	
	…	

Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

```
pointers11
0x7ffc489399d8
0x7ffc489399e0
0x7ffc489399e0
0x7ffc489399e4
1234
18
Press any key to continue...
```

	Address	Memory cell
	…	
*sp	…d8	…e0
	…	
s UID	…e0	1234
	…e1	
	…e2	
	…e3	
age	…e4	90
	…e5	
	…	

Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

```
pointers11
0x7ffc489399d8
0x7ffc489399e0
0x7ffc489399e0
0x7ffc489399e4
1234
18
Press any key to continue...
```

		Address	Memory cell
		…	
*sp		…d8	…e0
		…	
s	UID	…e0	1234
		…e1	
		…e2	
		…e3	
	age	…e4	90
		…e5	
		…	

Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

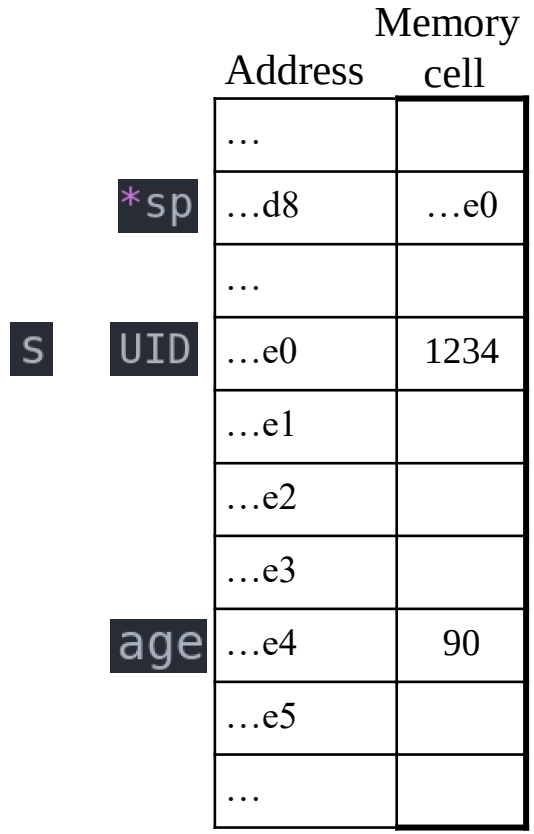
```
pointers11
0x7ffc489399d8
0x7ffc489399e0
0x7ffc489399e0
0x7ffc489399e4
1234
18
Press any key to continue...
```

		Address	Memory cell
		…	
*sp		…d8	…e0
		…	
s	UID	…e0	1234
		…e1	
		…e2	
		…e3	
	age	…e4	90
		…e5	
		…	

Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

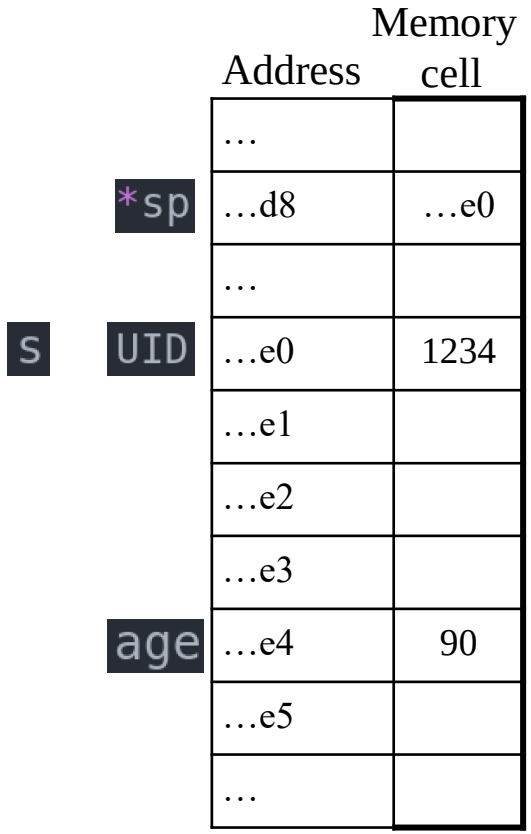
```
pointers11
0x7ffc489399d8
0x7ffc489399e0
0x7ffc489399e0
0x7ffc489399e4
1234
18
Press any key to continue...
```



Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

```
pointers11
0x7ffc489399d8
0x7ffc489399e0
0x7ffc489399e0
0x7ffc489399e4
1234
18
Press any key to continue...
```



Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

```
pointers11
0x7ffc489399d8
0x7ffc489399e0
0x7ffc489399e0
0x7ffc489399e4
1234
18
Press any key to continue...
```

		Address	Memory cell
		…	
	*sp	…d8	…e0
		…	
s	UID	…e0	1234
		…e1	
		…e2	
		…e3	
	age	…e4	90
		…e5	
		…	

Pointers and Struct – Step by Step

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp;
9    Student s = {1234, 90};
10   sp = &s;
11   cout << &sp << endl;
12   cout << &s << endl;
13   cout << &(s.UID) << endl;
14   cout << &(s.age) << endl;
15   cout << (*sp).UID << endl;
16   (*sp).age = 18;
17   cout << s.age << endl;
18 }
```

```
pointers11
0x7ffc489399d8
0x7ffc489399e0
0x7ffc489399e0
0x7ffc489399e4
1234
18
Press any key to continue...
```

		Address	Memory cell
		…	
	*sp	…d8	…e0
		…	
s	UID	…e0	1234
		…e1	
		…e2	
		…e3	
	age	…e4	18
		…e5	
		…	

Pointers and Struct – Member Access

- A member can be accessed through a pointer by either
  - the dereferencing operator * together with the member operator .
  - or the arrow operator ->

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student s = {1234, 90};
9    Student *sp = &s;
10   cout << (*sp).UID << endl;
11   cout << sp->UID << endl;
12 }
```

Member Access - Example

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    string a = "HKU";
5    string *s = &a;
6    cout << a.length() << endl;
7    cout << s->length() << endl;
8    cout << (*s).length() << endl;
9  }
```

Task

- What is the output of the following program?

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    string name = "Peter Chan";
5    string home = "Hong Kong";
6    string *s = &name;
7    cout << (*s).length() << endl;
8    (*s)[3] = 'c';
9    cout << (*s).substr(0, 5) << endl;
10   s = &home;
11   cout << (*s).length() << endl;
12   (*s)[3] = 'c';
13   cout << (*s).substr(0, 4) << endl;
14 }
```

NULL

- A pointer can be initialize as NULL
  - It stores no address value
  - This indicates that it is not pointing to any valid memory address

NULL - Example

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp = NULL;
9    if (sp == NULL)
10     cout << "NULL" << endl;
11   else {
12     cout << sp->UID << endl;
13   }
14 }
```

NULL

- Never dereference a pointer that could be NULL!

```
1  #include <iostream>
2  using namespace std;
3  struct Student {
4    int UID;
5    int age;
6  };
7  int main() {
8    Student *sp = NULL;
9    cout << sp->UID << endl;
10 }
```

Why use pointers ?

Dynamic Variables

- So far our program variables have been **static** in the sense that
  - the number of variables is fixed and known before execution of the program
  - points at which variables are created and destroyed are known
  - they are destroyed when they fall out of scope
- In contrast, we can create variables **dynamically**
  - the programmer controls when they are created and destroyed
  - no names are needed for these variables, we access them through pointers

Dynamic Variables

- Dynamic variables are created using the new operator
- The new operator returns a pointer to the variable
- We use the pointer to access and change the value of the variable
- A dynamic variable can be destroyed using the delete operator
  - memory allocated will be freed up

Dynamic Variables - Example

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int *p;
5    string *s;
6    p = new int;
7    s = new string;
8    *p = 8;
9    *s = "dragon";
10   cout << *p << " "<< *s << endl;
11   delete p;
12   delete s;
13 }
```

Task

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int *p1, *p2;
5    p1 = new int;
6    *p1 = 42;
7    p2 = p1;
8    cout << "*p1 = " << *p1 << endl;
9    cout << "*p2 = " << *p2 << endl;
10   *p2 = 53;
11   cout << "*p1 = " << *p1 << endl;
12   cout << "*p2 = " << *p2 << endl;
13   p1 = new int;
14   *p1 = 88;
15   cout << "*p1 = " << *p1 << endl;
16   cout << "*p2 = " << *p2 << endl;
17 }
```

Dangling Pointers

- When a dynamic variable pointed to by a pointer variable is destroyed using the delete operator
  - the value of this pointer variable becomes invalid
  - the values of any other pointer variables pointing to the same deleted dynamic variable also becomes invalid
- These invalid pointers are called dangling pointers
- If the dereference operator * is applied to a dangling pointer, the result is unpredictable and usually disastrous

Dangling Pointers - Example

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4     int *i = new int;
5     *i = 42;
6     int *ii = i;
7     delete i;
8     i = NULL;
9     cout << *ii << endl;
10  }
```

Pointers and Functions

- Pointers can be used as function parameters
- Passing an address to the function will have the same effect as pass by reference

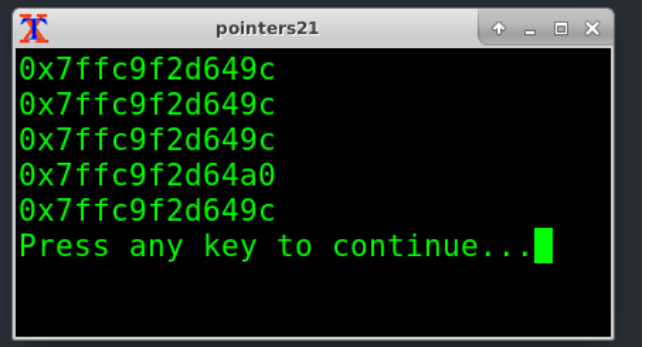
Pointers and Functions - Example

```
1  #include <iostream>
2  using namespace std;
3  void swap (int *x, int *y) {
4    int tmp = *x;
5    *x = *y;
6    *y = tmp;
7  }
8  int main() {
9    int a = 2, b = 5;
10   cout << a << " " << b << endl;
11   swap(&a, &b);
12   cout << a << " " << b << endl;
13 }
```

Pointers and Arrays

- When a pointer is referring to an array, it is storing the address of the 1st slot of the array

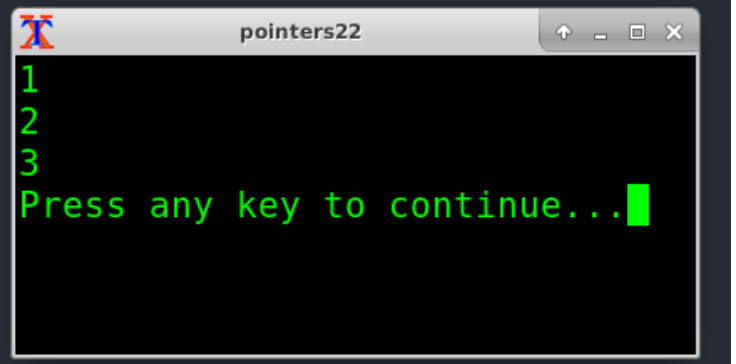
```
1   #include <iostream>
2   using namespace std;
3   int main() {
4     int a[] = {1, 2, 3};
5     cout << a << endl;
6     cout << &a << endl;
7     cout << &(a[0]) << endl;
8     cout << &(a[1]) << endl;
9     int *p = a;
10    cout << p << endl;
11  }
```



Pointers and Arrays

- We can use the increment / decrement operator on pointer variable to go to the next / previous slot of the array

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int a[] = {1, 2, 3};
5    int *p = a;
6    cout << *p << endl;
7    cout << *(p+1) << endl;
8    cout << *(p+2) << endl;
9  }
```



Dynamically Allocated Arrays

- The new operator can also be used to produce a new nameless array variable
- Unlike regular arrays, the size of a dynamically allocated array does not need to be a constant
  - It can be determined during program execution
- A dynamically allocated array can be destroyed using the delete operator to free up the memory allocated to it

Dynamically Allocated Arrays - Example

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int n;
5    cin >> n;
6    int *p = new int[n];
7    for (int i=0;i<n;i++)
8      cin >> p[i];
9    for (int i=0;i<n;i++)
10     cout << p[i] << " ";
11   delete[] p;
12 }
```

Task

- Write a function that takes an integer array and its size
- The function will return a pointer to the largest element in the array

Task

- What unfortunate misinterpretation can occur with the following declaration?

```
int* int_ptr1, int_ptr2;
```

Task

- What is the output of the following program?

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int i1, i2;
5    int *p1 = &i1, *p2 = &i2;
6    *p1 = 10;
7    *p2 = 20;
8    cout << *p1 << " " << *p2 << endl;
9    p1 = p2;
10   cout << *p1 << " " << *p2 << endl;
11   *p1 = 30;
12   cout << *p1 << " " << *p2 << endl;
13 }
```

Task

- What is the output of the following program?

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int i1, i2;
5    int *p1 = &i1, *p2 = &i2;
6    *p1 = 10;
7    *p2 = 20;
8    cout << *p1 << " " << *p2 << endl;
9    *p1 = *p2;
10   cout << *p1 << " " << *p2 << endl;
11   *p1 = 30;
12   cout << *p1 << " " << *p2 << endl;
13 }
```