CS31
Introduction to computer science

Fall 2015

Discussion Session 7

November 13, 2015

Notes

- Project 5 due 9:00 PM Tuesday, November 17
- Midterm: either 5:00 to 6:05 in La Kretz 110, 5:15 to 6:20 in Kinsey Pavilion 1220B, 5:30 to 6:35 in Humanities 169, or 5:45 to 6:50 in Humanities A51, Tuesday, November 17)
- Do not forget to sign up
- Topics up to and including C strings

Main Memory

Main Memory

- **Main Memory** consists of a contiguous sequence of bytes (8 bits) that each have a memory address associated with them.

Main Memory

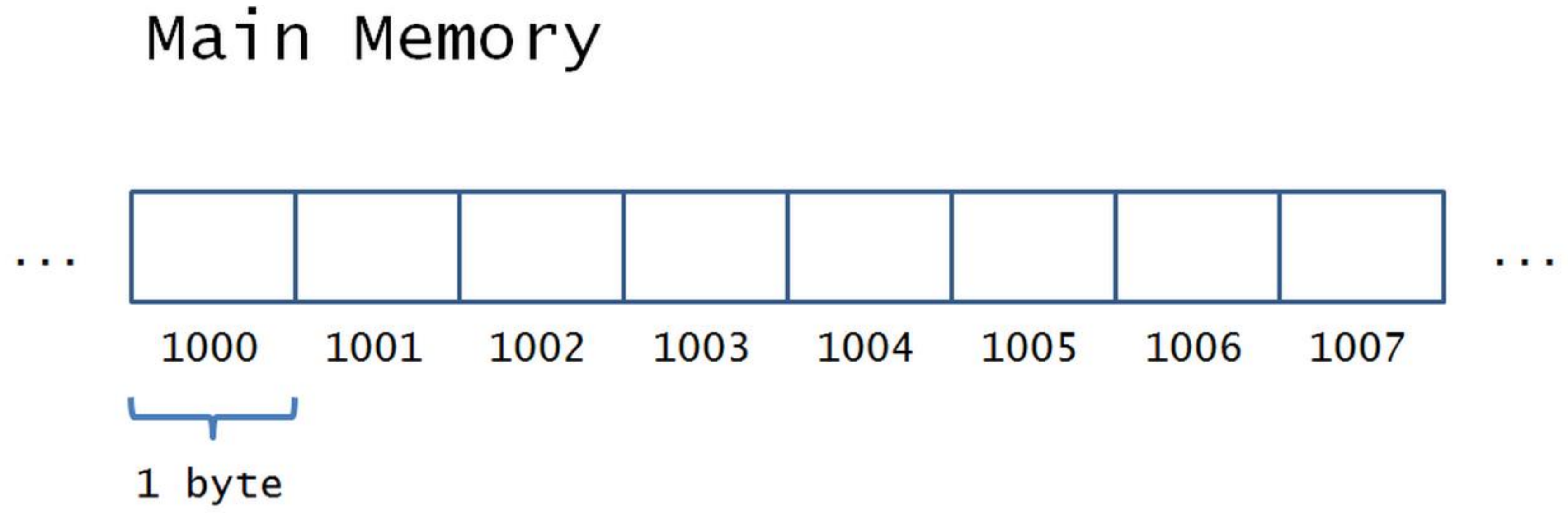
- **Main Memory** consists of a contiguous sequence of bytes (8 bits) that each have a memory address associated with them.

Main Memory

- **Main Memory** consists of a contiguous sequence of bytes (8 bits) that each have a memory address associated with them.
- **Memory Addresses** are just like indexes in an array; they are a sequential numbering of the memory cells.

Main Memory

- **Main Memory** consists of a contiguous sequence of bytes (8 bits) that each have a memory address associated with them.
- **Memory Addresses** are just like indexes in an array; they are a sequential numbering of the memory cells.



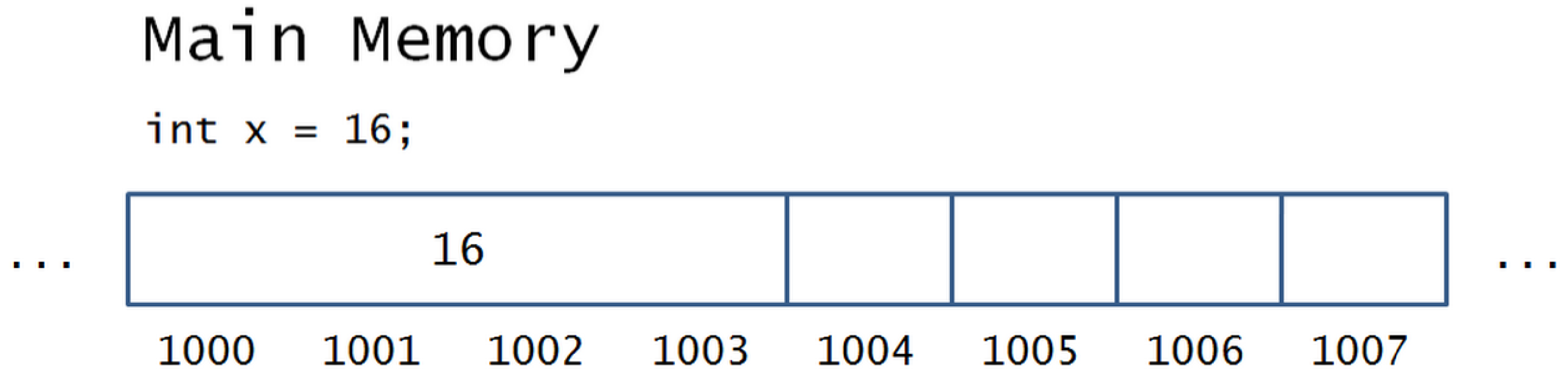
Main Memory - 2

Main Memory - 2

- int x = 16 (on a computer where an int takes up 4 bytes of memory)
    - If the compiler sticks it at memory address 1000, then in memory we have:

Main Memory - 2

- int x = 16 (on a computer where an int takes up 4 bytes of memory)
  - If the compiler sticks it at memory address 1000, then in memory we have:



Main Memory - 3

- char c[] = “cat”;

Main Memory - 3

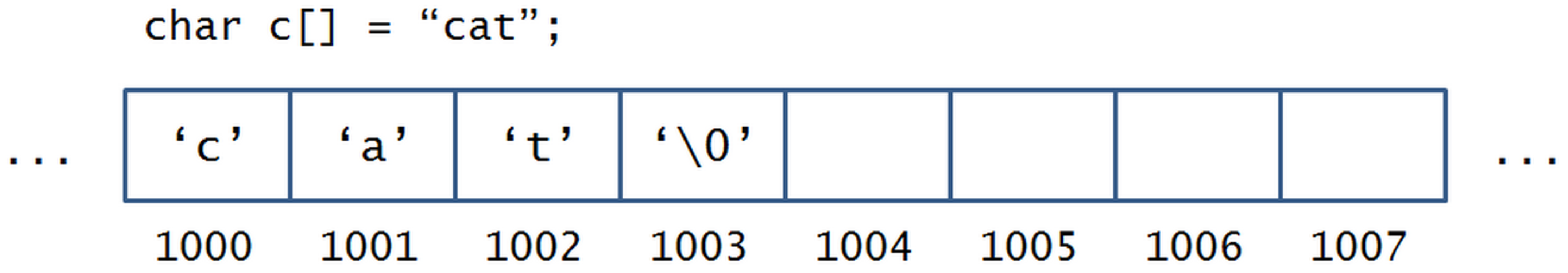
- char c[] = “cat”;
  - We know that each cell holding a character of the cstring "cat" will take up the same amount of space on any given system.

Main Memory - 3

- char c[] = “cat”;
  - We know that each cell holding a character of the cstring "cat" will take up the same amount of space on any given system.
  - Each character in the string “cat” will be located contiguously in memory, so if a char takes up 1 byte on our system, then c will take up 4 bytes of space

Main Memory - 3

- char c[] = “cat”;
  - We know that each cell holding a character of the cstring "cat" will take up the same amount of space on any given system.
  - Each character in the string “cat” will be located contiguously in memory, so if a char takes up 1 byte on our system, then c will take up 4 bytes of space



The & operator

The & operator

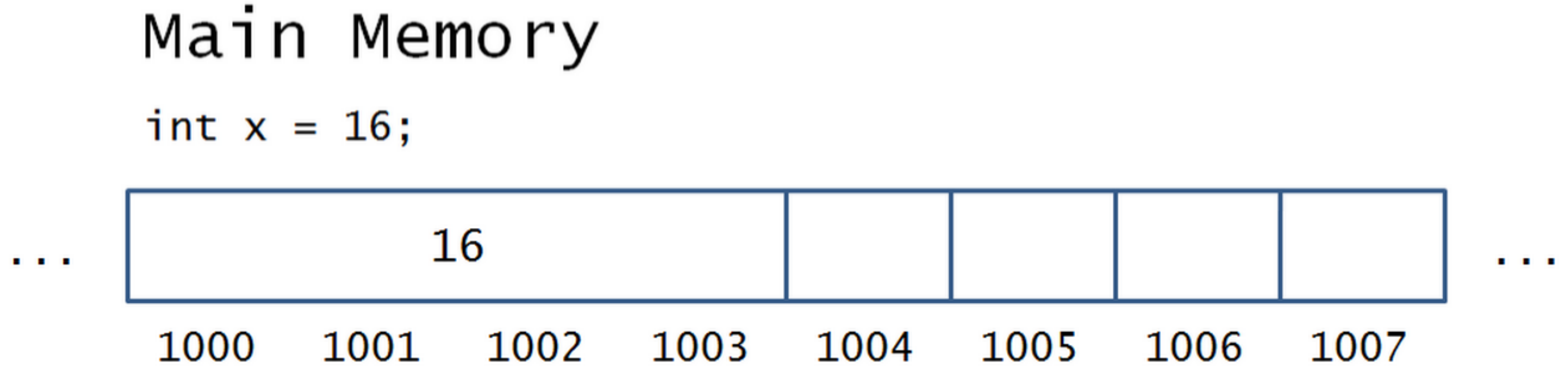
- **The address-of operator (&)**: The address-of operator says "give me the memory address of my rvalue," or the expression to the right of the ampersand.

The & operator

- **The address-of operator (&)**: The address-of operator says "give me the memory address of my rvalue," or the expression to the right of the ampersand.
  - In the case where the item we're requesting spans multiple bytes, the address-of operator returns the memory address of the first byte of that object.

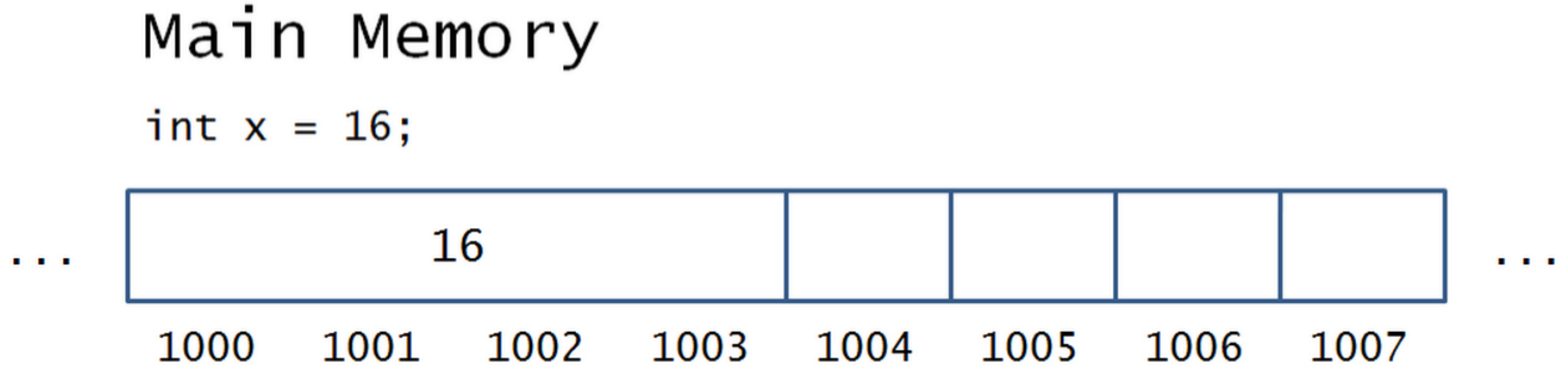
The & operator

- **The address-of operator (&)**: The address-of operator says "give me the memory address of my rvalue," or the expression to the right of the ampersand.
  - In the case where the item we're requesting spans multiple bytes, the address-of operator returns the memory address of the first byte of that object.



The & operator

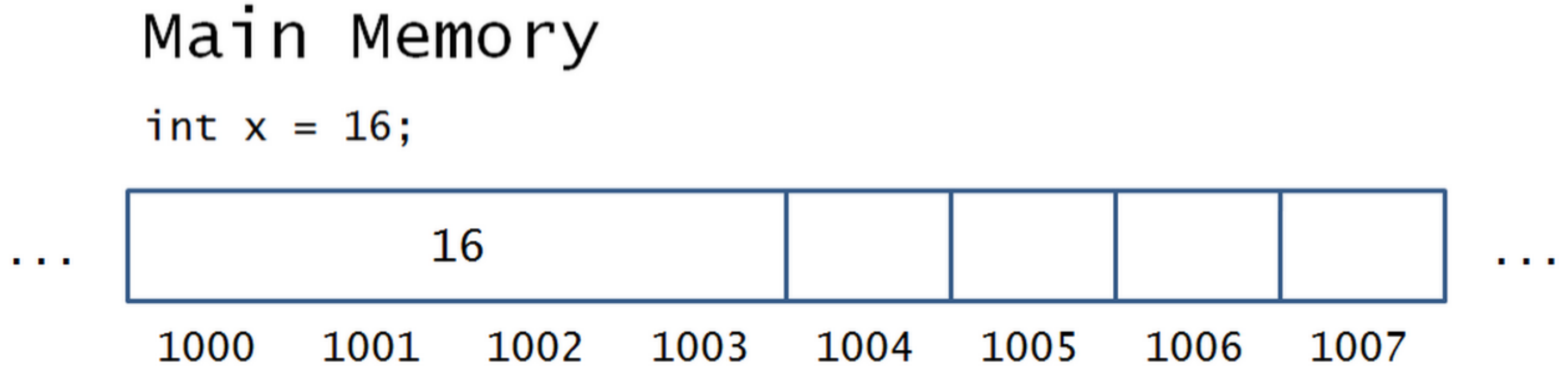
- **The address-of operator (&)**: The address-of operator says "give me the memory address of my rvalue," or the expression to the right of the ampersand.
  - In the case where the item we're requesting spans multiple bytes, the address-of operator returns the memory address of the first byte of that object.



Q: What is the value of &x?

The & operator

- **The address-of operator (&)**: The address-of operator says "give me the memory address of my rvalue," or the expression to the right of the ampersand.
  - In the case where the item we're requesting spans multiple bytes, the address-of operator returns the memory address of the first byte of that object.



Q: What is the value of &x?

A: 1000

What is a pointer?

What is a pointer?

- A variable that stores a reference (memory address) to another variable, or more generally, to some location in memory.

What is a pointer?

- A variable that stores a reference (memory address) to another variable, or more generally, to some location in memory.
  - (size is 4 bytes)

What is a pointer?

- A variable that stores a reference (memory address) to another variable, or more generally, to some location in memory.
  - (size is 4 bytes)
- **Pointers** store memory addresses and are assigned a type corresponding to the type of variable that they point to.

What is a pointer?

- A variable that stores a reference (memory address) to another variable, or more generally, to some location in memory.
  - (size is 4 bytes)
- **Pointers** store memory addresses and are assigned a type corresponding to the type of variable that they point to.
- Declaration syntax:
  - <type>* <name>;
  - ex: int* p; char* s; bool* sharp;

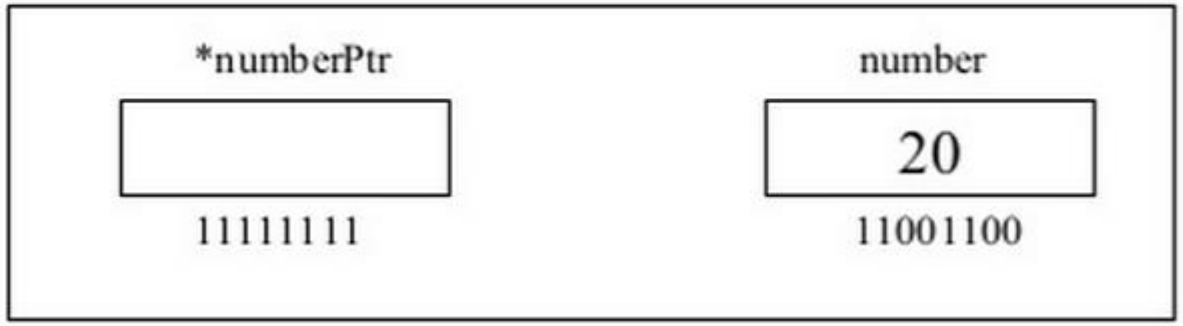
What is a pointer?

- A variable that stores a reference (memory address) to another variable, or more generally, to some location in memory.
  - (size is 4 bytes)
- **Pointers** store memory addresses and are assigned a type corresponding to the type of variable that they point to.
- Declaration syntax:
  - <type>* <name>;
  - ex: int* p; char* s; bool* sharp;
- Initialization:
  - To **initialize** pointers, we give them memory addresses corresponding to locations of variables.
  - Ex: int pointedAt = 16;

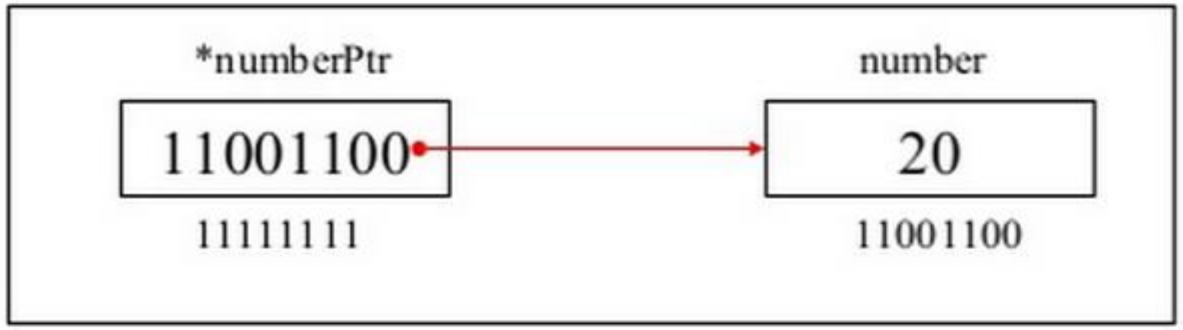
    int* pointer = &pointedAt;

Graphical Representation

```
int *numberPtr, number = 20;
```



```
numberPtr = &number;
```



The Deference Operator

The Deference Operator

- **The dereference operator (*)**: The dereference operator says "Give me the value of whatever's at the address my rvalue is pointing to." NOTE: This means the rvalue must be a pointer, i.e., a memory address!

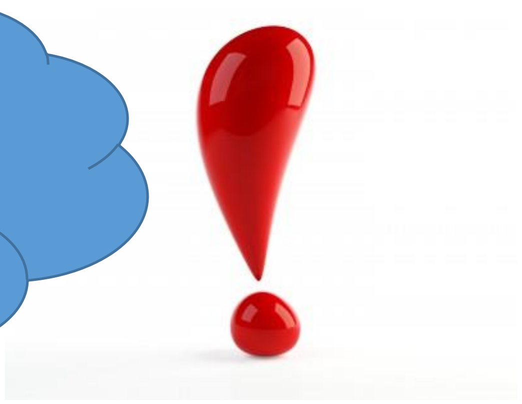
The Deference Operator

- **The dereference operator (*)**: The dereference operator says "Give me the value of whatever's at the address my rvalue is pointing to." NOTE: This means the rvalue must be a pointer, i.e., a memory address!
- So remember:

The Deference Operator

- **The dereference operator (*)**: The dereference operator says "Give me the value of whatever's at the address my rvalue is pointing to." NOTE: This means the rvalue must be a pointer, i.e., a memory address!
- So remember:

Ampersand: Address
Star: Value



Example

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

int main(){
    int pointedAt = -10;
    int* p = &pointedAt;
    cout << "Te value of what p points at : " << *p
    << endl;
    cout << "The address of what p points at : " << p
    << endl;
    cout << "The address of where p is stored : " <<
    &p << endl;

}
```

Example

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

int main()
{
   int x;
   int* p = &x;
   cout << *p << endl;

}
```

Example

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

int main()
{
   int x;
   int* p = &x;
   cout << *p << endl;

}
```

Q: What is the output of this program?

Example

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

int main()
{
   int x;
   int* p = &x;
   cout << *p << endl;

}
```

Q: What is the output of this program?

A: some junk value because x is not initialized

Example

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

int main(){
  int x;
  int* p = x;
  cout << *p << endl;

}
```

Example

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

int main(){
  int x;
  int* p = x;
  cout << *p << endl;

}
```

Q: What is the output of this program?

Example

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

int main(){
   int x;
   int* p = x;
   cout << *p << endl;

}
```

Q: What is the output of this program?

A: Compile error! Cannot convert from type int to int*

Example

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

int main(){

    int i = 2;
    double d = 2.2;
    int* p = &i;
    cout << *p << endl;
    p = &d;
    cout << *p << endl;

}
```

Example

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

int main(){

    int i = 2;
    double d = 2.2;
    int* p = &i;
    cout << *p << endl;
    p = &d;
    cout << *p << endl;

}
```

Q: What is the output of this program?

Example

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

int main(){

    int i = 2;
    double d = 2.2;
    int* p = &i;
    cout << *p << endl;
    p = &d;
    cout << *p << endl;

}
```

Q: What is the output of this program?

A: Compile error! Cannot convert from type double* to int*

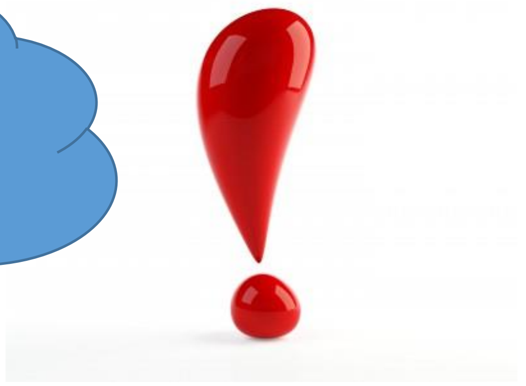
Example

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

int main(){

    int i = 2;
    double d = 2.2;
    int* p = &i;
    cout << *p << endl;
    p = &d;
    cout << *p << endl;

}
```

Q: What is the output of this program?

A: Compile error! Cannot convert from type double* to int*

Be aware that pointers should point to objects of their own type, or you'll likely get an error!



Reference and Dereference operators

- Reference and dereference operators could be considered inverses of one another, i.e., they "undo" each other.

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

int main(){

   int x = -10;
   int* p = &*&x;
   cout << *p << endl;
}
```

Reference and Dereference operators

- Reference and dereference operators could be considered inverses of one another, i.e., they "undo" each other.

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

int main(){

   int x = -10;
   int* p = &*&x;
   cout << *p << endl;
}
```

Q: What is the output of this program?

Reference and Dereference operators

- Reference and dereference operators could be considered inverses of one another, i.e., they "undo" each other.

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

int main(){

   int x = -10;
   int* p = &*&x;
   cout << *p << endl;
}
```

Q: What is the output of this program?

A: -10

Reference and Dereference operators

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

int main(){

   int x = -10;
   int* p = &x;
   cout << *(&(*p)) << endl;
}
```

Reference and Dereference operators

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

int main(){

   int x = -10;
   int* p = &x;
   cout << *(&(*p)) << endl;
}
```

Q: What is the output of this program?

Reference and Dereference operators

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

int main(){

   int x = -10;
   int* p = &x;
   cout << *(&(*p)) << endl;
}
```

Q: What is the output of this program?

A: -10

Assigning to pointers

Assigning to pointers

- Whenever we want to assign a value to the memory location that a pointer is pointing to, we use the following syntax:
  - *ptr = value;

Assigning to pointers

- Whenever we want to assign a value to the memory location that a pointer is pointing to, we use the following syntax:
  - *ptr = value;
- It's also perfectly legal to assign pointers to each other (i.e., store the same memory address in two pointers), so long as the types agree:

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

int main() {
int i = 0;
double d = 3.3;
bool b;

int* iPtr = &i;
double* dPtr = &*&d;
bool* bPtr = &b;

*iPtr = 20;

*&*dPtr = 3.33;

b = (*iPtr % 2 == 0) || (*dPtr > 4);

cout << i << endl;
cout << *iPtr << endl;
cout << d << endl;
cout << *dPtr << endl;
cout << b << endl;
cout << *bPtr << endl;
}
```

An integer pointer that points to the memory address at which integer i is located.

An double pointer that points to the memory address at which double d is located.

An bool pointer that points to the memory address at which bool b is located.

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

int main() {
int i = 0;
double d = 3.3;
bool b;

int* iPtr = &i;
double* dPtr = &*&d;
bool* bPtr = &b;

*iPtr = 20;

*&*dPtr = 3.33;

b = (*iPtr % 2 == 0) || (*dPtr > 4);

cout << i << endl;
cout << *iPtr << endl;
cout << d << endl;
cout << *dPtr << endl;
cout << b << endl;
cout << *bPtr << endl;
}
```

An integer pointer that points to the memory address at which integer i is located.

An double pointer that points to the memory address at which double d is located.

An bool pointer that points to the memory address at which bool b is located.

Q: What is the output of this program?

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

int main() {
int i = 0;
double d = 3.3;
bool b;

int* iPtr = &i;
double* dPtr = &*&d;
bool* bPtr = &b;

*iPtr = 20;

*&*dPtr = 3.33;

b = (*iPtr % 2 == 0) || (*dPtr > 4);

cout << i << endl;
cout << *iPtr << endl;
cout << d << endl;
cout << *dPtr << endl;
cout << b << endl;
cout << *bPtr << endl;
}
```

An integer pointer that points to the memory address at which integer i is located.

An double pointer that points to the memory address at which double d is located.

An bool pointer that points to the memory address at which bool b is located.

Q: What is the output of this program?

A:
```
20
20
3.33
3.33
1
1
```

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

int main() {
int pointedAt = 1;
int* pointy = &pointedAt;

int* ditto = pointy;

cout << *pointy << endl;
cout << *ditto << endl;


cout << pointy << endl;
cout << ditto << endl;
}
```

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

int main() {
int pointedAt = 1;
int* pointy = &pointedAt;

int* ditto = pointy;

cout << *pointy << endl;
cout << *ditto << endl;


cout << pointy << endl;
cout << ditto << endl;
}
```

Q: Will these 2 be equal?

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

int main() {
int pointedAt = 1;
int* pointy = &pointedAt;

int* ditto = pointy;

cout << *pointy << endl;
cout << *ditto << endl;


cout << pointy << endl;
cout << ditto << endl;
}
```

Q: Will these 2 be equal?

A: Yes!

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

int main() {
int pointedAt = 1;
int* pointy = &pointedAt;

int* ditto = pointy;

cout << *pointy << endl;
cout << *ditto << endl;


cout << pointy << endl;
cout << ditto << endl;
}
```

Q: Will these 2 be equal?

A: Yes!

Q: Will these 2 be equal?

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

int main() {
int pointedAt = 1;
int* pointy = &pointedAt;

int* ditto = pointy;

cout << *pointy << endl;
cout << *ditto << endl;


cout << pointy << endl;
cout << ditto << endl;
}
```

Q: Will these 2 be equal?

A: Yes!

Q: Will these 2 be equal?

A: Yes!

Uninitialized pointers

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

int main(){

   int *p;
   *p = 5;
   cout << *p << endl;
}
```

Uninitialized pointers

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

int main(){

   int *p;
   *p = 5;
   cout << *p << endl;
}
```

Uninitialized pointers can lead to undefined behavior or illegal memory accesses when they haven't been assigned somewhere first.

What can we do?

- We use ***the null pointer***
- It takes value NULL (or 0) that is defined in <cstddef>
    - #include<cstddef>
- NULL is a pointer literal that indicates a pointer that isn't pointing anywhere. We use it for safety (to not dereference an undefined pointer) and clarity.
- Does not mean address 0 in memory
- Guaranteed to be distinct from any actual memory address

Example

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

int main(){
   int i = 5;
   int *p = NULL;
   if (p == NULL)
   {
      p = &i;
   }
   else{
      cout << "It is not null" << endl;
   }
   cout << *p << endl;
}
```

Example

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

int main(){
   int i = 5;
   int *p = NULL;
   if (p == NULL)
   {
      p = &i;
   }
   else{
      cout << "It is not null" << endl;
   }
   cout << *p << endl;
}
```

Q: What is the output?

Example

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

int main(){
   int i = 5;
   int *p = NULL;
   if (p == NULL)
   {
      p = &i;
   }
   else{
      cout << "It is not null" << endl;
   }
   cout << *p << endl;
}
```

Q: What is the output?

A: 5

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

int main() {
   int i[][3] = {
   { 1, 2, 3 },
   { 4, 5, 6 }
   };

   int*  pointy = &i[1][1];
   int*  copyPointy = pointy;

   *pointy = 100;
   pointy = &i[0][2];

   cout << *pointy << endl;
   cout << *copyPointy << endl;
}
```

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

int main() {
   int i[][3] = {
   { 1, 2, 3 },
   { 4, 5, 6 }
   };

   int*  pointy = &i[1][1];
   int*  copyPointy = pointy;

   *pointy = 100;
   pointy = &i[0][2];

   cout << *pointy << endl;
   cout << *copyPointy << endl;
}
```

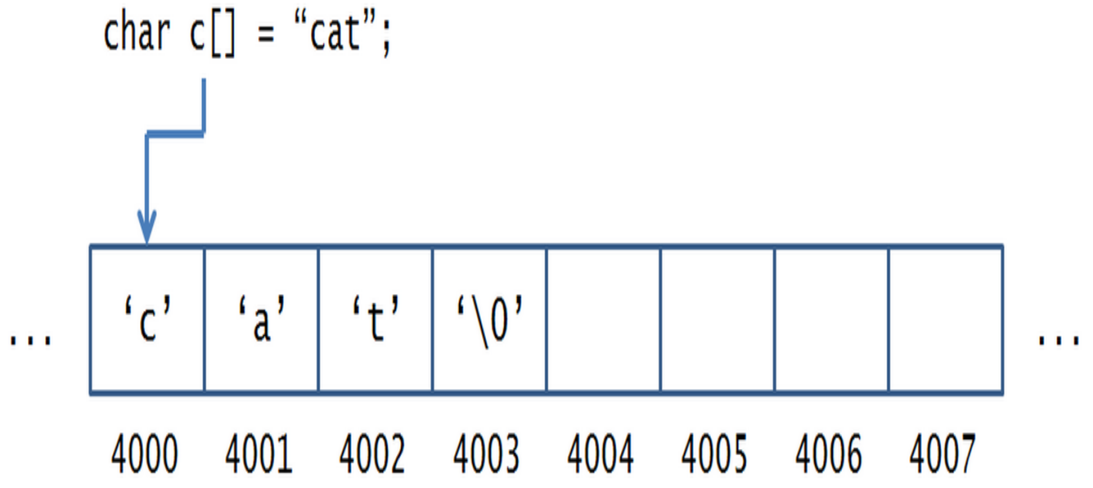
Q: What is the output of this program?

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

int main() {
   int i[][3] = {
   { 1, 2, 3 },
   { 4, 5, 6 }
   };

   int*  pointy = &i[1][1];
   int*  copyPointy = pointy;

   *pointy = 100;
   pointy = &i[0][2];

   cout << *pointy << endl;
   cout << *copyPointy << endl;
}
```

Q: What is the output of this program?

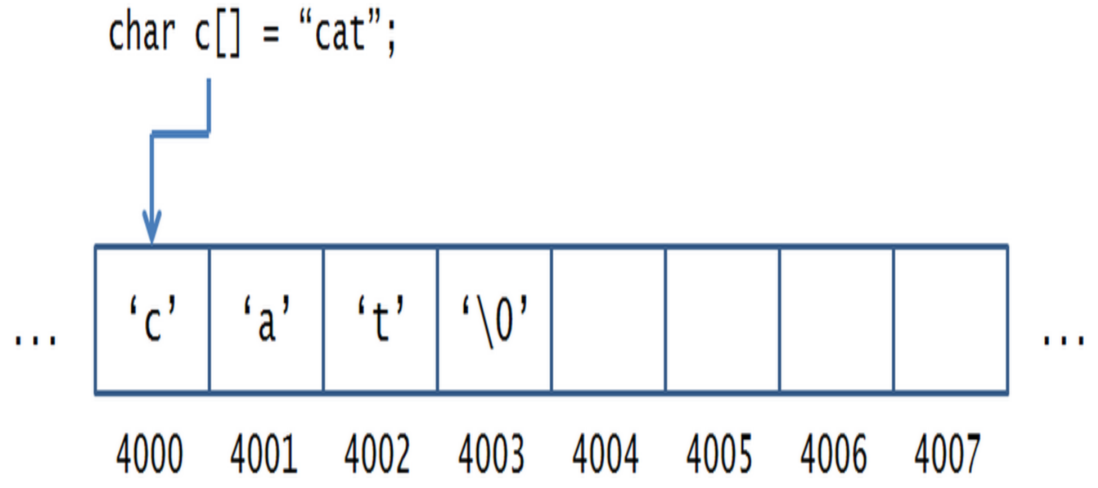
A: 3
   100

Pointers and Arrays - 1

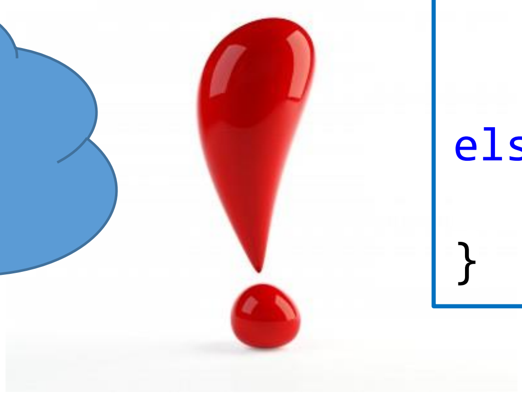


```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

int main() {

char c[] = "cat";

if (c == &c[0])
    cout << "Yes they point to the first
    element" << endl;
else
        cout << "Hmm" << endl;
}
```

Pointers and Arrays - 1

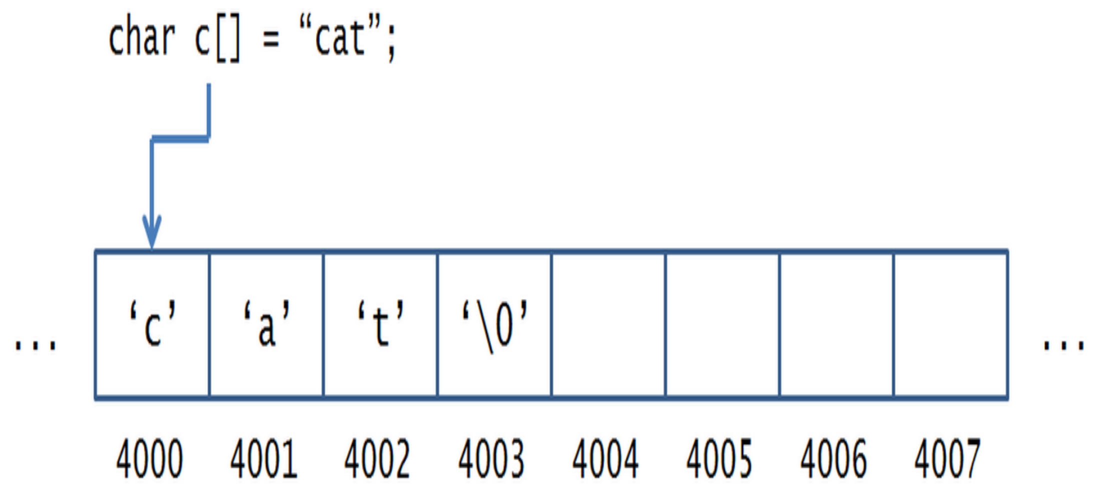


An array name refers to the address of the first element of the array

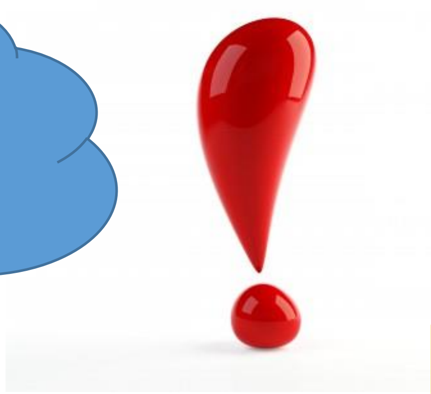


```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

int main() {

char c[] = "cat";

if (c == &c[0])
    cout << "Yes they point to the first
    element" << endl;
else
        cout << "Hmm" << endl;
}
```

Pointers and Arrays - 1



An array name refers to the address of the first element of the array



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

int main() {

char c[] = "cat";

if (c == &c[0])
    cout << "Yes they point to the first
    element" << endl;
else
        cout << "Hmm" << endl;
}
```

Output: Yes the point to the first element

Pointers and Arrays - 2

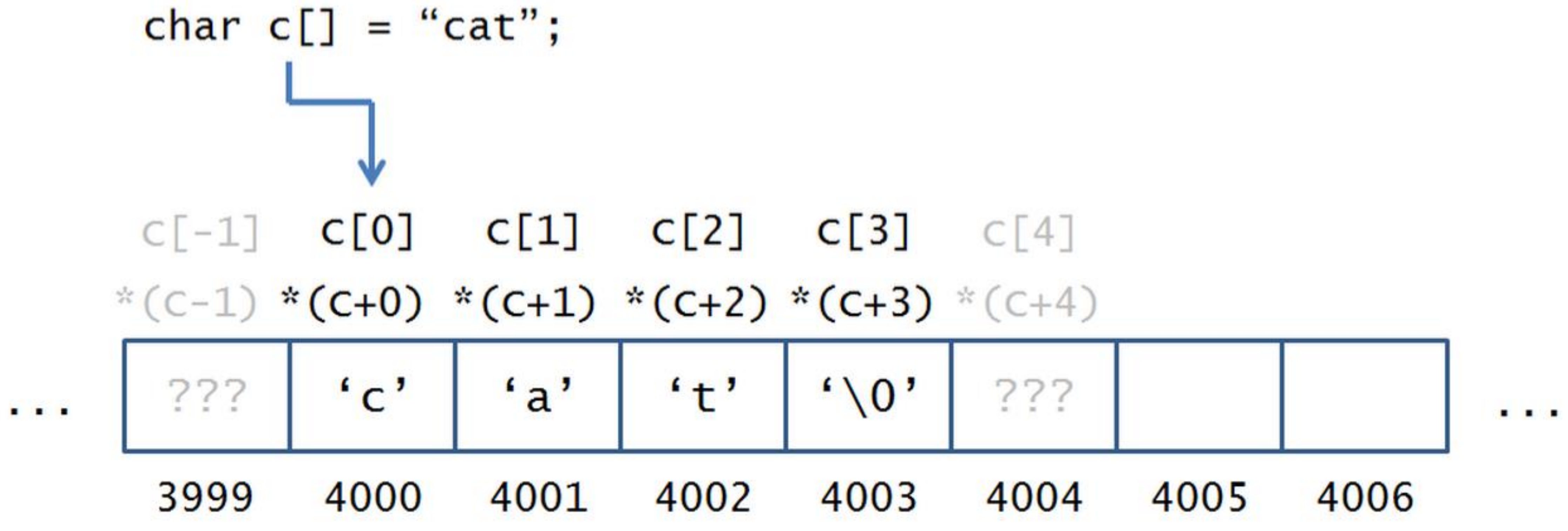
Pointers and Arrays - 2

- The bracket notation of pointers can be thought of as "dereference with offset," meaning that, if ptr is a pointer:
  - ptr[offset] is equivalent to saying *(ptr + offset)
  - i.e. give me the value at offset locations away from memory address ptr

Pointers and Arrays - 2

- The bracket notation of pointers can be thought of as "dereference with offset," meaning that, if ptr is a pointer:
  - ptr[offset] is equivalent to saying *(ptr + offset)
  - i.e. give me the value at offset locations away from memory address ptr
- A **pointer offset** is a number indicating how many x bytes away the next or previous value is, depending on the type of the pointer.

  For example, if an int is treated as 4 bytes, and I want to access the int directly following the int at address 1000 (as in an int array starting at address 1000), then an int pointer offset of 1 tells me to look 4 bytes later, at memory address 1004. Similarly, an int pointer offset of -2 tells me to look 8 bytes before, at memory address 992, etc.

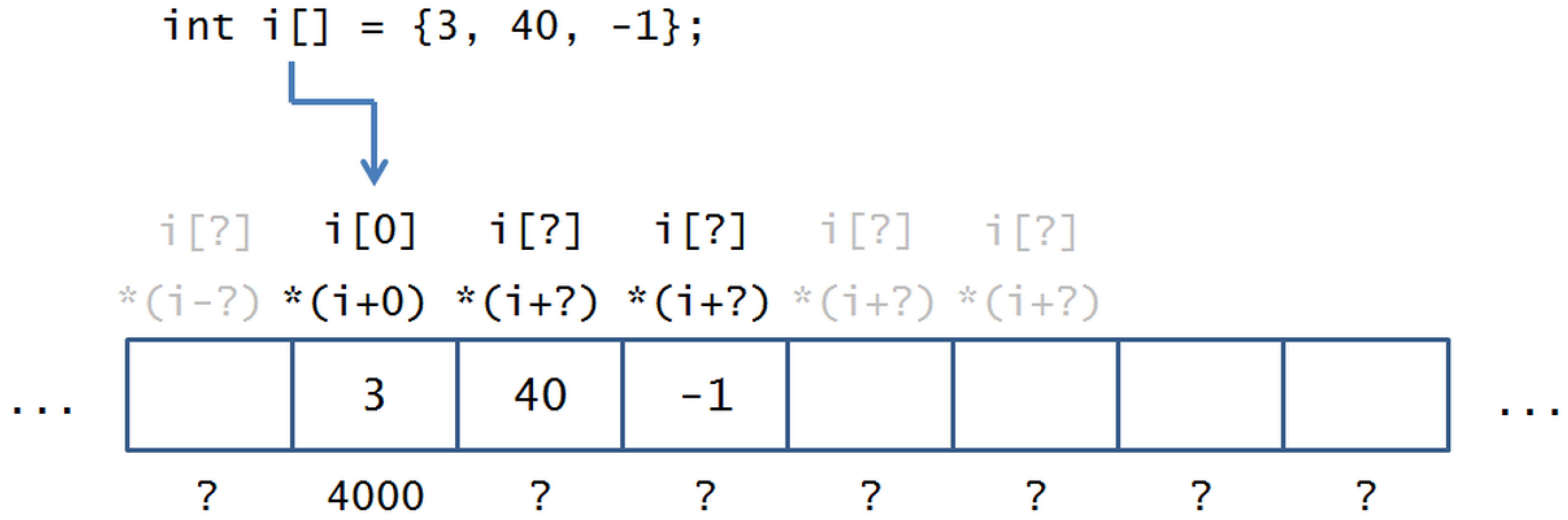

char c[] = “cat”;
c[-1] c[0] c[1] c[2] c[3] c[4]
*(c-1) *(c+0) *(c+1) *(c+2) *(c+3) *(c+4)
... ??? ‘c’ ‘a’ ‘t’ ‘\0’ ??? ...
3999 4000 4001 4002 4003 4004 4005 4006

•Notice how c is an array of chars, and so c is a char pointer, meaning that its offsets assume the next char is 1 byte away.

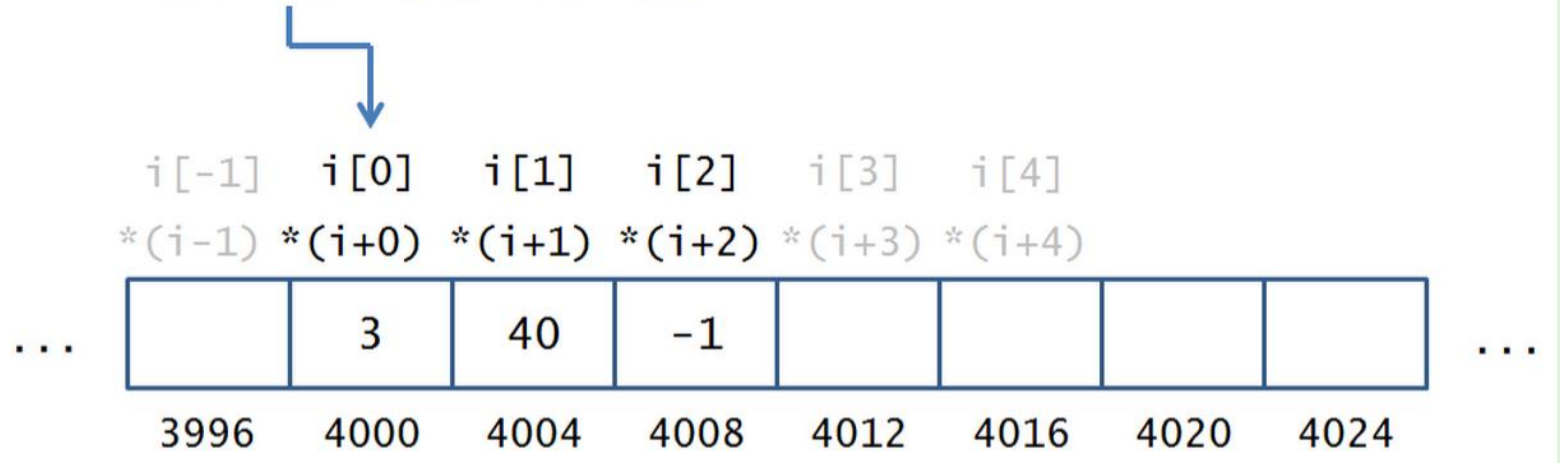
•The bracket c[i] notation is equivalent to the offset *(c + i) notation.

•Although we could generate a pointer to c[4] by saying &c[4], it would be unsafe to assign anything to c[4] (overflow).

If I had an array of ints where ints were treated as 4 bytes a piece, then fill in the blanks below:

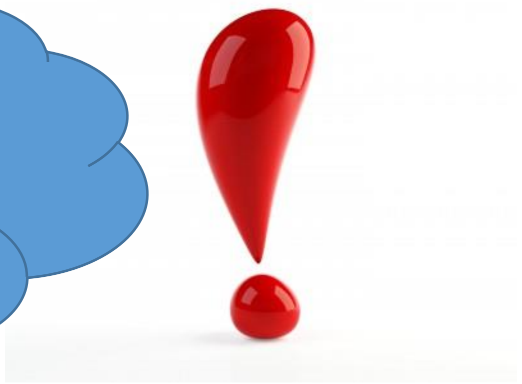


```
int i[] = {3, 40, -1};
```



i[-1] i[0] i[1] i[2] i[3] i[4]
*(i-1) *(i+0) *(i+1) *(i+2) *(i+3) *(i+4)
... 3 40 -1 ...
3996 4000 4004 4008 4012 4016 4020 4024

Every time I use an offset with a pointer, it automatically scales to how large its type is in memory.



Example

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

int main(){

char c[] = "cat";
   int len = strlen(cat);
   for (int i = 0; i<len; i++)
   {
      cout << c[0] << endl;
      c++;
   }
}
```

Example

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

int main(){

char c[] = "cat";
   int len = strlen(cat);
   for (int i = 0; i<len; i++)
   {
      cout << c[0] << endl;
      c++;
   }
}
```

Q: What is the output of this program?

Example

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

int main(){

char c[] = "cat";
   int len = strlen(cat);
   for (int i = 0; i<len; i++)
   {
      cout << c[0] << endl;
      c++;
   }
}
```

Q: What is the output of this program?

A: Compile error! Cannot change where c points to.

Example

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

int main(){

char c[] = "cat";
   int len = strlen(cat);
   for (int i = 0; i<len; i++)
   {
      cout << c[0] << endl;
      c++;
   }
}
```

Q: What is the output of this program?

A: Compile error! Cannot change where c points to.

Q: How can we do it?

Example

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

int main(){
   char c[] = "cat";
   char* helper = c;
   int len = strlen(c);
   for (int i = 0; i<len; i++)
   {
      cout << helper[0] << endl;
      helper++;
   }
}
```

Another way of doing that!

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

int main(){
   char c[] = "cat";
   for (char* helper = c; *helper != '\0'; helper++)
   {
          cout << helper[0] << endl;
   }
}
```

Another way of doing that!

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

int main(){
   char c[] = "cat";
   for (char* helper = c; *helper != '\0'; helper = &helper[1])
      {
           cout << helper[0] << endl;
      }
}
```

Examples

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

int main(){
   char c[] = "cat";
   for (char* helper = c; *helper !=
   '\0'; helper++)
   {
      cout << helper << endl;
   }
}
```

Examples

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

int main(){
   char c[] = "cat";
   for (char* helper = c; *helper !=
   '\0'; helper++)
   {
      cout << helper << endl;
   }
}
```

Q: What is the output of this program?

Examples

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

int main(){
   char c[] = "cat";
   for (char* helper = c; *helper !=
   '\0'; helper++)
   {
      cout << helper << endl;
   }
}
```

Q: What is the output of this program?

A:
```
cat
at
t
```

Pointer Arithmetic and Operators

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

int main() {

    double d[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    double* ptr = d;

    cout << *ptr << endl;
    cout << *(ptr + 1) << endl;
    cout << ptr[2] << endl;

    ptr += 2;
    cout << ptr[2] << endl;
}
```

Pointer Arithmetic and Operators

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

int main() {

    double d[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    double* ptr = d;

    cout << *ptr << endl;
    cout << *(ptr + 1) << endl;
    cout << ptr[2] << endl;

    ptr += 2;
    cout << ptr[2] << endl;
}
```

Q: What is the output of this program?

Pointer Arithmetic and Operators

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

int main() {

    double d[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    double* ptr = d;

    cout << *ptr << endl;
    cout << *(ptr + 1) << endl;
    cout << ptr[2] << endl;

    ptr += 2;
    cout << ptr[2] << endl;
}
```

Q: What is the output of this program?

A:
```
1.1
2.2
3.3
5.5
```

Pointer Arithmetic and Operators - 2

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

int main() {

   double d[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
   double* ptr = d;

   cout << *(ptr * 2) << endl;
}
```

Pointer Arithmetic and Operators - 2

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

int main() {

    double d[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    double* ptr = d;

    cout << *(ptr * 2) << endl;
}
```

Q: What is the output of this program?

Pointer Arithmetic and Operators - 2

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

int main() {

    double d[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    double* ptr = d;

    cout << *(ptr * 2) << endl;
}
```

Q: What is the output of this program?

A: Compile Error!

Pointer Arithmetic and Operators - 2

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

int main() {

   double d[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
   double* ptr = d;

   cout << *(ptr * 2) << endl;
}
```

Q: What is the output of this program?

A: Compile Error!

Pointers are NOT defined on multiplication or division.

Comparing pointers

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main() {
    char c[] = "cat";
    char t[] = "dog";
    char* cPtr = c;
    char* tPtr = t;
    int len = strlen(c);

    for (int i = 0; i < len; i++) {
        if (cPtr < tPtr) {
                cout << "cPtr: " << cPtr << endl;
        }
        else {
                cout << "tPtr: " << tPtr << endl;
        }
    }
}
```

Comparing pointers

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main() {
    char c[] = "cat";
    char t[] = "dog";
    char* cPtr = c;
    char* tPtr = t;
    int len = strlen(c);

    for (int i = 0; i < len; i++) {
        if (cPtr < tPtr) {
                cout << "cPtr: " << cPtr << endl;
        }
        else {
                cout << "tPtr: " << tPtr << endl;
        }
    }
}
```

Q: What is the output of this program?

Comparing pointers

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main() {
    char c[] = "cat";
    char t[] = "dog";
    char* cPtr = c;
    char* tPtr = t;
    int len = strlen(c);

    for (int i = 0; i < len; i++) {
        if (cPtr < tPtr) {
               cout << "cPtr: " << cPtr << endl;
        }
        else {
               cout << "tPtr: " << tPtr << endl;
        }
    }
}
```

Q: What is the output of this program?

A: Unpredictable behavior!

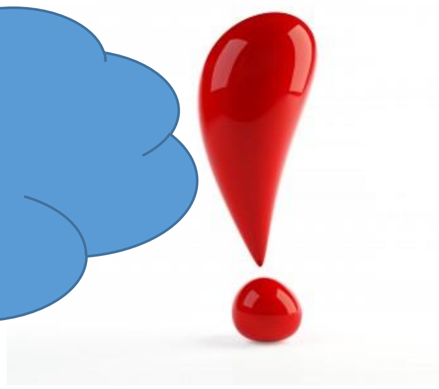
Comparing pointers

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main() {
    char c[] = "cat";
    char t[] = "dog";
    char* cPtr = c;
    char* tPtr = t;
    int len = strlen(c);

    for (int i = 0; i < len; i++) {
        if (cPtr < tPtr) {
                cout << "cPtr: " << cPtr << endl;
        }
        else {
                cout << "tPtr: " << tPtr << endl;
        }
    }
}
```

Q: What is the output of this program?

A: Unpredictable behavior!

We DO NOT know where variables will be placed in memory at runtime.



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

int main() {
   char c[] = "catdog";
   char* cPtr = c;
   int len = strlen(c);

   for (int i = 0; i < len; i++)
   {
      if (cPtr < &c[3]) {
      cout << *cPtr << endl;
      cPtr++;
      }
   }
}
```

Pointers *within* the same array are always guaranteed to be comparable using <, >, etc. because array elements are located contiguously in memory.

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

int main() {
   char c[] = "catdog";
   char* cPtr = c;
   int len = strlen(c);

   for (int i = 0; i < len; i++)
   {
      if (cPtr < &c[3]) {
      cout << *cPtr << endl;
      cPtr++;
      }
   }
}
```

Pointers *within* the same array are always guaranteed to be comparable using <, >, etc. because array elements are located contiguously in memory.

Q: What is the output of this program?

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

int main() {
   char c[] = "catdog";
   char* cPtr = c;
   int len = strlen(c);

   for (int i = 0; i < len; i++)
   {
      if (cPtr < &c[3]) {
      cout << *cPtr << endl;
      cPtr++;
      }
   }
}
```

Pointers *within* the same array are always guaranteed to be comparable using <, >, etc. because array elements are located contiguously in memory.

Q: What is the output of this program?

A: c
   a
   t

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

int main() {
   int i[] = { 1, 2, 3, 4 };
   int* p1 = i;
   int* p2 = (i + 2);

   cout << (p1 - p2) << endl;
   cout << (p2 - p1) << endl;
}
```

When two pointers are operating in the same array, we can even subtract them to learn their *offsets,* not their difference in bytes.

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

int main() {
   int i[] = { 1, 2, 3, 4 };
   int* p1 = i;
   int* p2 = (i + 2);

   cout << (p1 - p2) << endl;
   cout << (p2 - p1) << endl;
}
```

Q: What is the output of this program?

When two pointers are operating in the same array, we can even subtract them to learn their *offsets,* not their difference in bytes.

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

int main() {
   int i[] = { 1, 2, 3, 4 };
   int* p1 = i;
   int* p2 = (i + 2);

   cout << (p1 - p2) << endl;
   cout << (p2 - p1) << endl;
}
```

Q: What is the output of this program?

A: -2
   2

When two pointers are operating in the same array, we can even subtract them to learn their *offsets,* not their difference in bytes.

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

int main() {
int i[] = { 5, 6, 7, 8, 9 };
int* iPtr = i;
int len = 3;

for (int j = 0; j < len; j++) {
   if (iPtr < &i[3]) {
   // What will print here?
   cout << (&i[3] - iPtr) << endl;
   iPtr++;
      }
     }
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

int main() {
int i[] = { 5, 6, 7, 8, 9 };
int* iPtr = i;
int len = 3;

for (int j = 0; j < len; j++) {
   if (iPtr < &i[3]) {
   // What will print here?
   cout << (&i[3] - iPtr) << endl;
   iPtr++;
      }
     }
}
```

Q: What is the output of this program?

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

int main() {
int i[] = { 5, 6, 7, 8, 9 };
int* iPtr = i;
int len = 3;

for (int j = 0; j < len; j++) {
   if (iPtr < &i[3]) {
   // What will print here?
   cout << (&i[3] - iPtr) << endl;
   iPtr++;
      }
     }
}
```

Q: What is the output of this program?

A: 3
    2
    1

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

int main() {
   char c[] = "testing";
   int len = strlen(c);

   for (int i = 0; i < len / 2; i++) {
      char* front = &c[i];
      char* back = &c[len - i - 1];
      char stuff = *front;
      *front = *back;
      *back = stuff;
   }
      cout << c << endl;
}
```

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

int main() {
   char c[] = "testing";
   int len = strlen(c);

   for (int i = 0; i < len / 2; i++) {
      char* front = &c[i];
      char* back = &c[len - i - 1];
      char stuff = *front;
      *front = *back;
      *back = stuff;
   }
      cout << c << endl;
}
```

Q: What is the output of this program?

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

int main() {
   char c[] = "testing";
   int len = strlen(c);

   for (int i = 0; i < len / 2; i++) {
      char* front = &c[i];
      char* back = &c[len - i - 1];
      char stuff = *front;
      *front = *back;
      *back = stuff;
   }
      cout << c << endl;
}
```

Q: What is the output of this program?

A: gnitset

Pointers and Functions

- Function prototype:

```
void pointerFunc(int* i, int p[]) {
...
}
```

- Pointers are passed by value! meaning a copy of the address is made and is named by the parameter. The values that pointer can dereference and change are NOT copied.
- Function can also return a pointer

```
int* findInt(int arr[], int len, int match) {
...
}
```

Example

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

void set(int* a){
     *a = 4;
};

int main() {
   int m = 0;
   set(&m);
   cout << "m is " << m << endl;
}
```

Example

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

void set(int* a){
     *a = 4;
};

int main() {
   int m = 0;
   set(&m);
   cout << "m is " << m << endl;
}
```

Q: What is the output of this program?

Example

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

void set(int* a){
     *a = 4;
};

int main() {
   int m = 0;
   set(&m);
   cout << "m is " << m << endl;
}
```

Q: What is the output of this program?

A: m is 4

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

void func(int arr[])
{
    arr++;
    cout << arr[0] << endl;
    cout << *arr << endl;
    *arr = 5;
}
void gunk(int* arr)
{
    arr++;
    cout << arr[0] << endl;
    cout << *arr << endl;
    *arr = 10;
}
int main() {
    int ouch[] = { 1, 2, 3 };
    func(ouch);
    gunk(ouch);
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

void func(int arr[])
{
    arr++;
    cout << arr[0] << endl;
    cout << *arr << endl;
    *arr = 5;
}
void gunk(int* arr)
{
    arr++;
    cout << arr[0] << endl;
    cout << *arr << endl;
    *arr = 10;
}
int main() {
    int ouch[] = { 1, 2, 3 };
    func(ouch);
    gunk(ouch);
}
```

Q: What is the output of this program?

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

void func(int arr[])
{
    arr++;
    cout << arr[0] << endl;
    cout << *arr << endl;
    *arr = 5;
}
void gunk(int* arr)
{
    arr++;
    cout << arr[0] << endl;
    cout << *arr << endl;
    *arr = 10;
}
int main() {
    int ouch[] = { 1, 2, 3 };
    func(ouch);
    gunk(ouch);
}
```

Q: What is the output of this program?

A: 2
2
5
5

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

int* findInt(int arr[], int len, int match) {
    for (int i = 0; i < len; i++) {
        if (arr[i] == match) {
        return &arr[i];
    }
}

        return NULL;
};

int main() {
    int i[] = { 5, 6, 7, 8, 9 };
    int* ptr = findInt(i, 5, 7);

    if (ptr != NULL) {
        cout << *ptr << endl;
    }
    else {
        cout << "null!" << endl;
    }
}
```

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

int* findInt(int arr[], int len, int match) {
    for (int i = 0; i < len; i++) {
        if (arr[i] == match) {
        return &arr[i];
    }
}

        return NULL;
};

int main() {
    int i[] = { 5, 6, 7, 8, 9 };
    int* ptr = findInt(i, 5, 7);

    if (ptr != NULL) {
        cout << *ptr << endl;
    }
    else {
        cout << "null!" << endl;
    }
}
```

Q: What is the output of this program?

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

int* findInt(int arr[], int len, int match) {
    for (int i = 0; i < len; i++) {
        if (arr[i] == match) {
        return &arr[i];
    }
}

        return NULL;
};

int main() {
    int i[] = { 5, 6, 7, 8, 9 };
    int* ptr = findInt(i, 5, 7);

    if (ptr != NULL) {
        cout << *ptr << endl;
    }
    else {
        cout << "null!" << endl;
    }
}
```

Q: What is the output of this program?

A: 7

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

int* findInt(int arr[], int len, int match) {
    for (int i = 0; i < len; i++) {
        if (arr[i] == match) {
        return &arr[i];
    }
}

        return NULL;
};

int main() {
    int i[] = { 5, 6, 7, 8, 9 };
    int* ptr = findInt(i, 5, 7);

    if (ptr != NULL) {
        cout << *ptr << endl;
    }
    else {
        cout << "null!" << endl;
    }
}
```

Q: What is the output of this program?

A: 7

Could be:

```
int* findInt(int* arr,
int len, int match) {
```

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
void findInt(int arr[], int len, int match, int* ptr) {
    for (int i = 0; i < len; i++) {
    if (arr[i] == match) {
    ptr = &arr[i];
    return;
    }
    }
    ptr = NULL;
};
int main() {
    int i[] = { 5, 6, 7, 8, 9 };
    int* ptr = i;
    findInt(i, 5, 7, ptr);
    if (ptr != NULL) {
        cout << *ptr << endl;
    }
    else {
        cout << "null!" << endl;
    }
}
```

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
void findInt(int arr[], int len, int match, int* ptr) {
    for (int i = 0; i < len; i++) {
    if (arr[i] == match) {
    ptr = &arr[i];
    return;
    }
    }
    ptr = NULL;
};
int main() {
    int i[] = { 5, 6, 7, 8, 9 };
    int* ptr = i;
    findInt(i, 5, 7, ptr);
    if (ptr != NULL) {
        cout << *ptr << endl;
    }
    else {
        cout << "null!" << endl;
    }
}
```

Q: What is the output of this program?

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
void findInt(int arr[], int len, int match, int* ptr) {
    for (int i = 0; i < len; i++) {
    if (arr[i] == match) {
    ptr = &arr[i];
    return;
    }
    }
    ptr = NULL;
};
int main() {
    int i[] = { 5, 6, 7, 8, 9 };
    int* ptr = i;
    findInt(i, 5, 7, ptr);
    if (ptr != NULL) {
        cout << *ptr << endl;
    }
    else {
        cout << "null!" << endl;
    }
}
```

Q: What is the output of this program?

A: 5

Practice

- Define a function, ptrsToMinMax that takes in an arrays of ints, the size of that array, and two int pointers, and then sets each pointer equal to the address holding the minimum and maximum values within that array.

- What is the function prototype?

Practice

- Define a function, ptrsToMinMax that takes in an arrays of ints, the size of that array, and two int pointers, and then sets each pointer equal to the address holding the minimum and maximum values within that array.

- What is the function prototype?

```
void ptrToMinMax(int arr[], int*& min, int*&
max, int n);
```

Main for testing

```
int main() {
   int arr[] = { 1, 5, 0, 2, 4 };
   int* min;
   int* max;

   ptrToMinMax(arr, min, max, 5);

   assert(*min == 0);
   assert(*max == 5);
   cerr << "[!] ALL TESTS PASSED!" << endl;
}
```

```
void ptrToMinMax(int arr[], int*& min, int*& max, int n) {
   if (n <= 0) {
      return;
   }
   min = &arr[0];
   max = &arr[0];
   // Iterate through all n elements of arr
   for (int i = 0; i < n; i++) {
      if (arr[i] < *min) {
            min = &arr[i];
      }
   // [!] If the current element is greater than the set the max
   to that element
   if (arr[i] > *max) {
      max = &arr[i];  }
   }
}
```

Reference Type

- A reference variable contains the address of another variable i.e. it is an alias for another variable.
- Reference variable cannot be modified after declaration
- In CS31, we should avoid stand-alone reference variables
  - But reference parameters can be used as needed

With and Without reference

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

int main() {

int gamma = 26;
int nickname = gamma;

cout << nickname << " " << gamma << endl;
nickname = 77;
cout << nickname << " " << gamma << endl;
nickname++;
cout << nickname << " " << gamma << endl;
gamma++;
cout << nickname << " " << gamma << endl;

return 0;
}
```

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

int main() {

int gamma = 26;
int & nickname = gamma;

cout << nickname << " " << gamma << endl;
nickname = 77;
cout << nickname << " " << gamma << endl;
nickname++;
cout << nickname << " " << gamma << endl;
gamma++;
cout << nickname << " " << gamma << endl;

return 0;
}
```