CS 31 Discussion

ABDULLAH-AL-ZUBAER IMRAN

WEEK 10: FINAL REVIEW

Review

Compilation/syntax errors

- Some common syntax errors:
  - Missing semicolons at ends of statements
  - Missing brackets around blocks
  - Missing namespace or #include definitions
  - Misspelled variables or names

Runtime/logic errors

- Some common runtime errors:
  - Division by 0
  - Overflow (e.g.: trying to hold a really big number in an int variable that exceeds its bounds)

Strings, chars, loops

String

```
#include <string>
```

Operation	What it does	Example
`string s = “hello”;` `string s = “!!!”;`	Declare strings s and s2	
`s.length()` or `s.size()`	Return the length of s	`cout << s.size(); // prints 5`

`s[i]` or `s.at[i]`	Return i-th character. (`i` should be integer between 0 and size-1 (inclusive))	`cout << s[1]; // prints ‘e’`
		`cout << s.at(0); // prints ‘h’`

`s + s2`	Concatenate two strings	`cout << s + s2; // prints “hello!!!”`

Strings, chars, loops

Char

Characters can be represented by a unique integer value.

ASCII(American Standard Code for Information Interchange) defines the mapping between characters and integers.

Operation	What it does
`char c;`	Declare a character c
`isspace(c)`	True if c is a whitespace character
`isalpha(c)`	True if c is a letter
`isdigit(c)`	True if c is a digit
`islower(c)`	True is c is a lowercase letter
`isupper(c)`	True if c is a uppercase letter

`<cctype>` library

Strings, chars, loops

`cin >> var;` command accesses input characters, ignores whitespace, and ignores the newline at the end of the user's input. We use this to get numerical input, and store it in variable "`var`".

`getline(cin, s);` command consumes all characters up to, and including, the newline character. It then throws away the newline, and stores the resulting string in `s`. We use this to gather string inputs. (requires `<string>` library)

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

int main () {
  string inputString;
  int inputInt;
  cout << "Enter a number: ";
  cin >> inputInt;
  cout << "Input was: " << inputInt << endl;
  cout << "Enter a string: ";
  getline(cin, inputString);
  cout << "Input was: " << inputString << endl;
}
```

```
Input:
32
world
Output:
Enter a number: 32
Input was: 32
Enter a string: Input was:
```

Stream

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main () {
6   string inputString;
7   int inputInt;
8   cout << "Enter a number: ";
9   cin >> inputInt;
10  cout << "Input was: " << inputInt << endl;
11  cout << "Enter a string: ";
12  getline(cin, inputString);
13  cout << "Input was: " << inputString << endl;
14}
```

```
Input:
32
world
Output:
Enter a number: 32
Input was: 32
Enter a string:Input was:
```

At line 9:

The value 32 is stored in the variable `inputInput`, '\n' is left in the `cin` stream.

At line 12:

The `getline(cin, inputString)` consumes '\n', then discards the newline '\n', and stores the string left to `inputString`. Since there is nothing left, null character '' is stored to `inputString`.

Stream

Use `cin.ignore(n,pattern)` when we've used `cin` and then directly after use `getline`.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main () {
6   string inputString;
7   int inputInt;
8   cout << "Enter a number: ";
9   cin >> inputInt;
10  cout << "Input was: " << inputInt << endl;
11  cin.ignore(10000, '\n');
11  cout << "Enter a string: ";
12  getline(cin, inputString);
13  cout << "Input was: " << inputString << endl;
14}
```

```
Input:
32
world
Output:
Enter a number: 32
Input was: 32
Enter a string: world
Input was: world
```

To fix it, we will need to consume the extra newline.

`cin.ignore(n, pattern)` ignores n characters or until the first encountered instance of pattern from input stream.

At line 11:

`cin` `'\n'` `cin`

`cin.ignore(10000, '\n');` command consumes either 10000 characters, or discards all the characters until the first encountered `'\n'` (inclusive).

At line 12:

`cin` `'w' 'o' 'r' 'l' 'd' '\n'` `inputString` `world`

The `getline(cin, inputString)` consumes "world\n", then discards the newline `'\n'`, and stores the string left to `inputString`.

CStrings

C strings are **null-terminated**

```
char s[10] = "HOWAREYOU";
```

H	O	W	A	R	E	Y	O	U	\0

Operation	What it does
`strlen(s)`	Returns the length of `s`, not counting `'\0'`.
`strcpy(t,s)`	Copies the string `s` to `t`. (Notes: `t=s` won't do the job. Also, `strcpy` doesn't do the size check for you. You must make sure there's enough space in t yourself.)
`strncpy(t,s,n)`	Copies the first n characters of `s` to `t`. (Note: No size check.)
`strcat(t,s)`	Appends `s` to the end of `t`. (Notes: No size check. `t += s` won't do the job.)
`strcmp(t,s)`	Compares `t` and `s`. Returns `0` if they are equal, something greater than `0` if `t > s`, and something less than `0` if `t < s`. (Note: `t == s` or `t < s` won't do the job.)

```
#include <cstring>
```

Strings, chars, loops

If-else statements

symbol	Meaning
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
&&	AND
||	OR

Strings, chars, loops

Switch statements

```
switch (expression) {
  case constant1:
    group-of-statements-1;
    break;
  case constant2:
    group-of-statements-2;
    break;
  .
  .
  .
  default:
    default-group-of-statements;
}
```

Switch example	If-else equivalent

`switch (x) {`
`  case 1: cout << "x is 1";`
`  break;`
`case 2:`
`  cout << "x is 2";`
`  break;`
`default:`
`  cout << "value of x unknown";`
`}`
	`if (x == 1) {`
	`  cout << "x is 1";`
	`} else if (x == 2) {`
	`  cout << "x is 2";`
	`} else {`
	`  cout << "value of x unknown";`
	`}`

Strings, chars, loops

Loops let you repeat the same or similar task multiple times. Three primitive loops in C++: `while`, `do-while`, and `for`.

`while` loop

```
while (condition)
  body;
```

1. Evaluate the `condition`.
   If `true`,
   2. run the body.
   Go back to 1,
   If `false`,
   exit the while loop.

`do-while` loop

```
do {
  body;
} while (condition);
```

1. Execute the body.
2. Evaluate the condition
   If `true`,
   Go back to 1,
   If `false`,
   exit the while loop.

Don't forget the ';' here.

Notice: The body in do-while loop will be executed once no matter what.

Strings, chars, loops

for loop

```
for (initialization; condition; update)
  body;
```

1. Execute `initialization`.
2. Evaluate the `condition`.
   If `true`,
   3. Run the `body`.
   4. Do the `update`.
   Go back to 2.
   If `false`,
   exit the `for` loop.

functions, reference, scope

function declaration (function prototype/signature)

function body (function implementation)

function call

A reference variable is an alias, or put simple, another name for an already existing variable.

<type>& <name>

arrays

Declare an array

- <type> <name>[size]

```
int a[5] = {1, 2, 3, 5, 7};
int a[] = {1, 2, 3, 5, 7};
```

Out-of-boundary access not allowed!

```
void fibo10(int fib[]);
void fibo10(int fib[], int n);
```

Pointers

Pointers store memory addresses and are assigned a type corresponding to the type of the variable that they point to.

<type>* <name> // declares a pointer of the given <type> and calls it <name>.

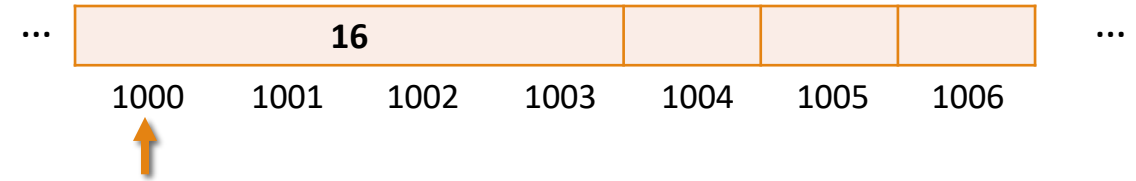
```
int* ptrAge;
bool* ptrValid;
char* ptrName;
```

To **initialize** pointers

```
int age = 16;
int* prtAge = &age;
```

or

```
int* ptrAge;
ptrAge = &age;
```



Pointers and Arrays

int arr[100];

arr is actually a pointer(int*)

Special for arr, the pointee can't change.

In order to get the value of arr[1]

- arr[1]
- *(arr+1)

Pointer Arithmetic

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

int main(){
  int arr[100];
  int var = 100;
  for (int i = 0; i < 100; i++)
    arr[i] = i;
  cout << arr+1 << endl;
  cout << arr+2 << endl;
}
```

```
0x7ffce0d80114
0x7ffce0d80118
```

arr is pointing to the “integer”. A integer is of 4 bytes on this platform.

Pointers and Arrays

You can treat an array variable like a pointer – well, it is a pointer. Therefore, the following are equivalent:

```
int findFirst(const string a[], int n, string target);
int findFirst(const string* a, int n, string target);
```

Recall

- Pass by value

```
int foo(int n);
```

- Pass by reference

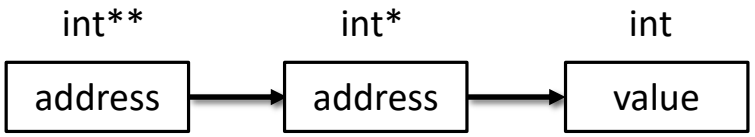
```
int foo(int &n);
```

- Pass by pointer

```
int foo(int a[]);   int foo(int* a);
```

Pointer to pointer

int** var;




```
#include <iostream>
using namespace std;
int main() {
  int var;
  int *ptr;
  int **pptr;
  var = 3000;
  ptr = &var;
  pptr = &ptr;
  cout << "Value of var = " << var << endl;
  cout << "Value available at *ptr = " << *ptr << endl;
  cout << "Value available at **pptr = " << **pptr << endl;
}
```

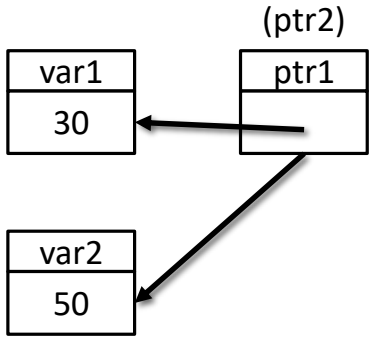
```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

Reference to Pointer

int* &ptr;

```
#include <iostream>
using namespace std;
int main() {
  int var1 = 30;
  int var2 = 50;
  int* ptr1 = &var1;
  int* &ptr2 = ptr1;
  cout << *ptr1 << endl;
  ptr2 = &var2;
  cout << *ptr1 << endl;
}
```

```
30
50
```



Struct

Structs are objects in C++ that represent "data structures", or variables, functions, etc. that are organized under a categorizing identifier.

Data Members are variable components of a given struct; they can be of any variable type.

```
struct <structName> {
  <member1_type> <member1_name>;
  <member2_type> <member2_name>;
  // ...etc.
}; // Remember the semicolon!
```

```
struct Student {
  string name;
  int id;
  string email;
  char grade;
};
```

Pointers to Structures

```
student *p;
```

You can refer to a member of the structure pointed by p by first dereferencing it and using the dot operator.

```
(*p).name
```

Or

```
p->name
```

This one works only if p is a pointer.

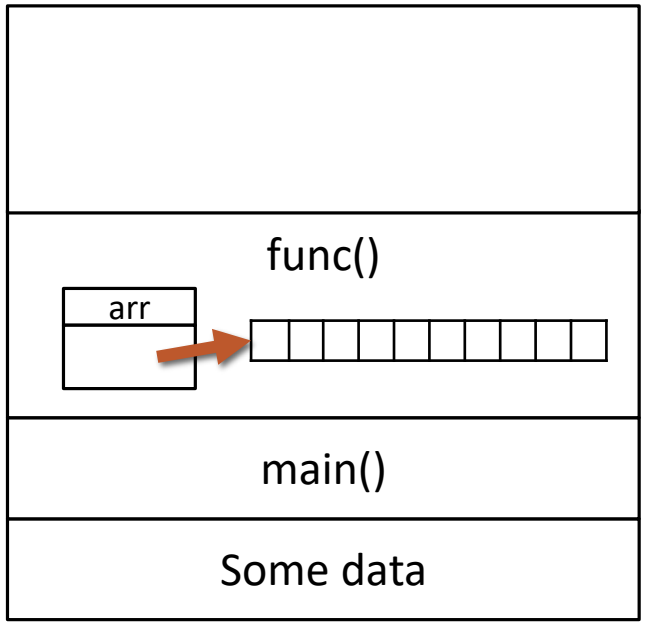
const

```
const int a1 = 3;

const int* a2

int const * a3;

int * const a4

Int const * const a5
```

- For member functions in class
- int Cat::age() const
- {
- /* code */
- }
- Ban age() in class Cat from beging anything which can attempt to alter any member variables in the object.

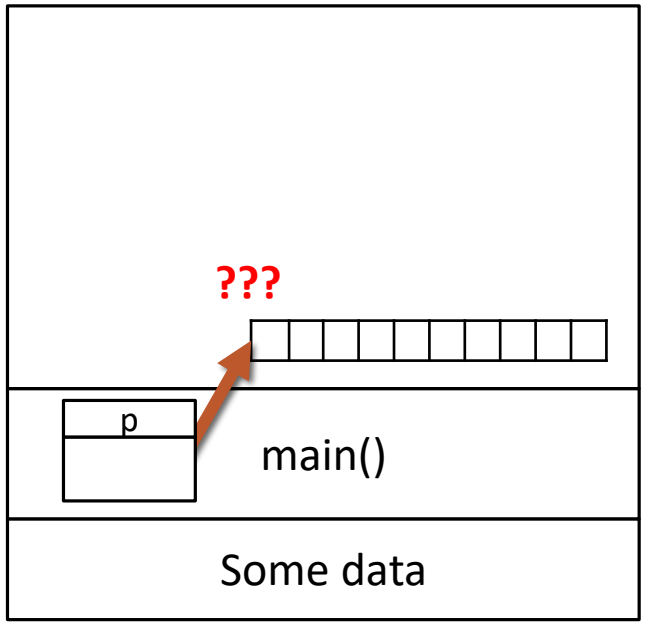
Dynamic Allocation of Memory

```
int *func();
int main()
{
 int *p = func();
 return 0;
}
int *func() {
 int arr[10];
 return arr;
}
```



Dynamic Allocation of Memory

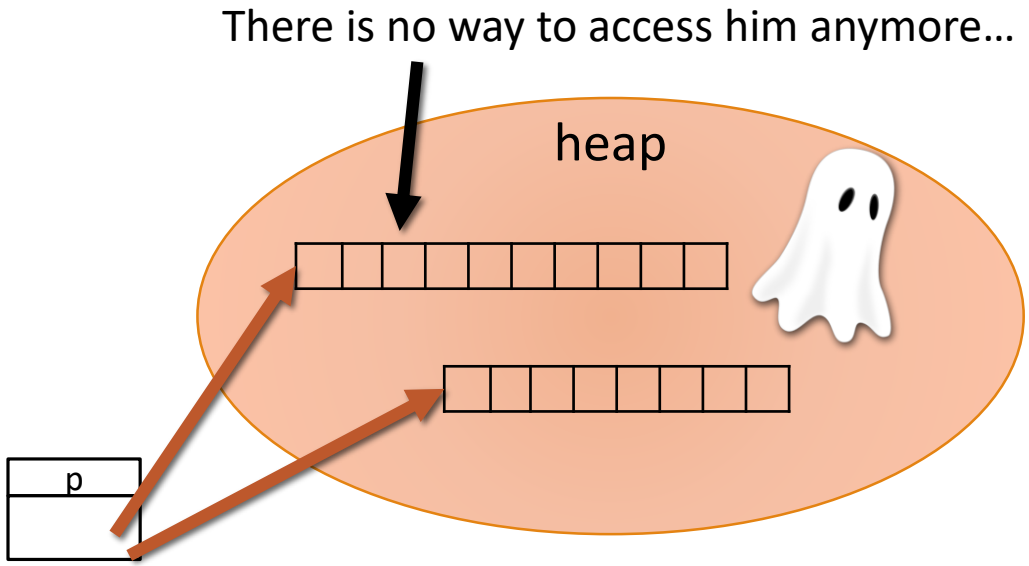
```
int *func();
int main()
{
 int *p = func();
 return 0;
}
int *func() {
 int arr[10];
 return arr;
}
```



Memory Leak

What happens here?

```
int *p;
p = new int[10];
p = new int[8];
```



A simple rule to remember:

For each new statement, there should be one delete statement.

Class - Constructors

A **constructor** is a member function that has the same name as the class, no return type, and automatically performs initialization when we declare an object:

```
class Cat {
public:
  Cat();
  int age();
  void meow();
private:
  int m_age;
};
Cat::Cat() {
  setAge(0);
  cout << "A cat is born" << endl;
}
```

When a constructor has no arguments, it is called the **default** constructor.
The compiler generates an empty one by default.

```
Cat kitty;
// uses default constructor -- Cat()
Cat *p1 = new Cat;
// uses Cat()
Cat *p2 = new Cat();
// uses Cat()
```

Class - Constructors

We can also create multiple constructors for a class by overloading it.

```
class Cat {
public:
  Cat(); // default constructor
  Cat(int initAge); // constructor with an initial age void meow();
  int age();
  void setAge(int newAge);
private:
  int m_age;
};
Cat::Cat(int initAge) {
  Cat(); // I can call the default constructor,
  setAge(initAge); // and then set the age to initAge
}
```

```
Cat kitty1(3);
```

Class - Constructors

Using initialization lists to initialize fields

```
class Cat {
public:
  Cat();
  int age();
  void setAge(int newAge);
private:
  int m_age;
  int m_weight;
  int m_gender;
};
```

```
Cat::Cat(): m_age(0), m_weight(10), m_gender(1)
{ /* some code */ }
```

Class -Destructors

A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

- The destructor should use `delete` to eliminate any dynamically allocated variables created by the object
- A destructor's name starts with ~, followed by the name of the class, with no return type or arguments.

```
class Cat {
public:
  Cat(int initAge);
  ~Cat();
  int age();
  void setAge(int newAge);
private:
  int m_age;
};
```

Thanks!

Questions?

Please complete online course evaluation by June 8!!

Good luck for the Final ☺

Some of the materials presented have been taken from earlier TA discussions