CSC230

Intro to C++ Lecture 10

Outline

- Memory Management

Dynamic memory in C

void * malloc(int num_bytes)

- Function defined in stdlib.h
- Allocates num_bytes of bytes and return a pointer to the allocated block of memory

free(void* ptr)

- Function defined in stdlib.h
- Return the memory pointed by ptr. The memory will be re-used by subsequent malloc calls

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
  int *ptr;

  ptr = (int *)malloc(sizeof(int));

  *ptr = 25;
  free(ptr);
  return 0;
}
```

Both gcc and g++ can compile this code

Dynamic memory in C++

new

- Allocates memory from heap
- Returns a **pointer** to the memory
  - double *ptr = new double;
  - int *array = new int[24];

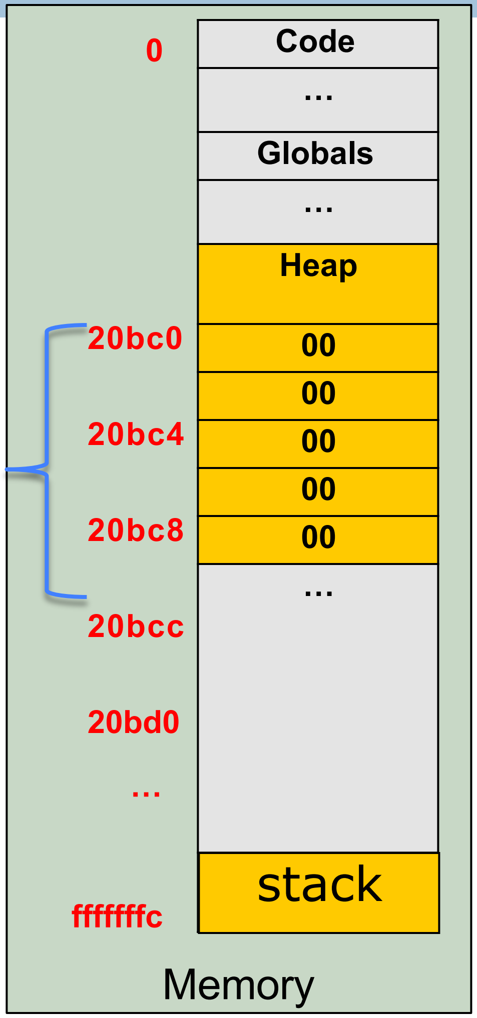
delete

- Returns the memory to heap
- followed by the pointer to the data that you want to deallocate
  - delete ptr;
  - delete [] ptr; // ptr is a pointer to an array

Dynamic memory allocation

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

int main(int argc, char *argv[]){
  int count;
  cin >> count;
  int *salary=new int[count];
  delete [] salary;
  return 0;
}
```



new allocates:

salary[0]

salary[1]

salary[2]

salary[3]

salary[4]

What to fill?

- int*
- double*
- char*
- **char****
- color*

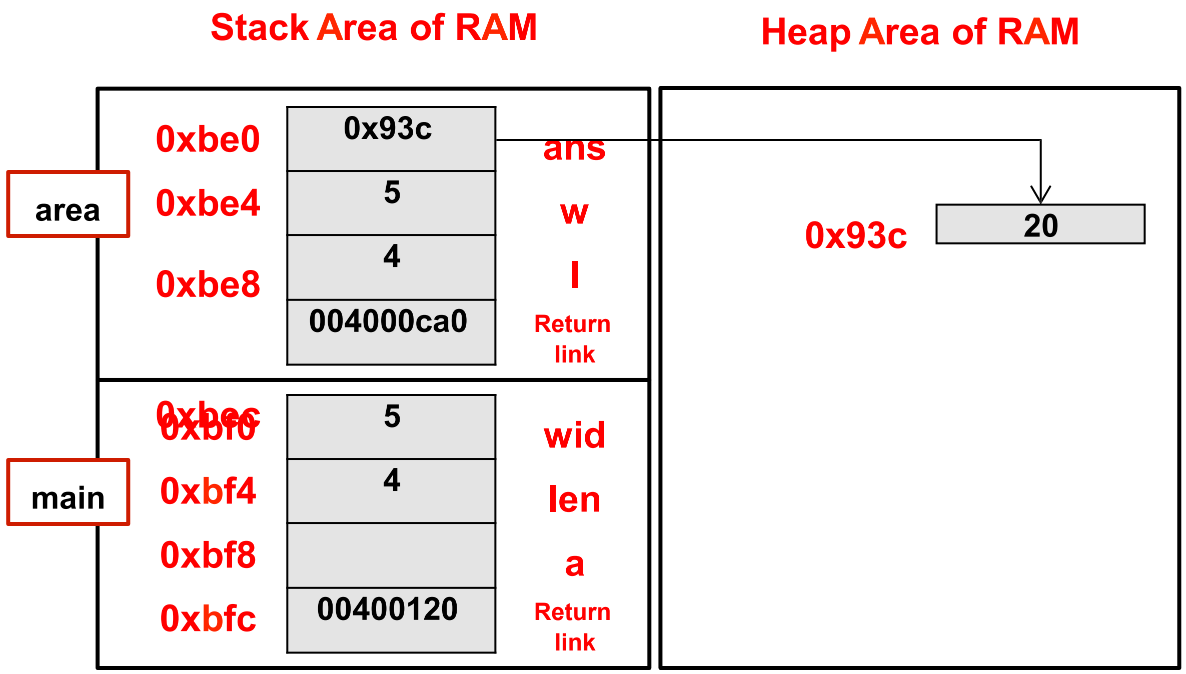
→

- ______ ptr = new int;
- ______ ptr = new double;
- ______ ptr = new char[10];
- ______ ptr = new char*[10];
- ______ ptr = new color;

Dynamic allocation

Dynamic Allocation

- Stored in heap
  - Pointer accesses it
- Exists until user 'delete' it
  - If it is not deleted, the data exists in heap even pointer dies



```
int  area(int, int);

int  main()
{
   int  wid  =  5,  len  =  4,
     a; a  =  area(wid,len);
}

int area(int  w,  int  l)
{
  int* ans = new int;
   *ans  =  w  *  l;
   return *ans;
}
```

Dynamic allocation

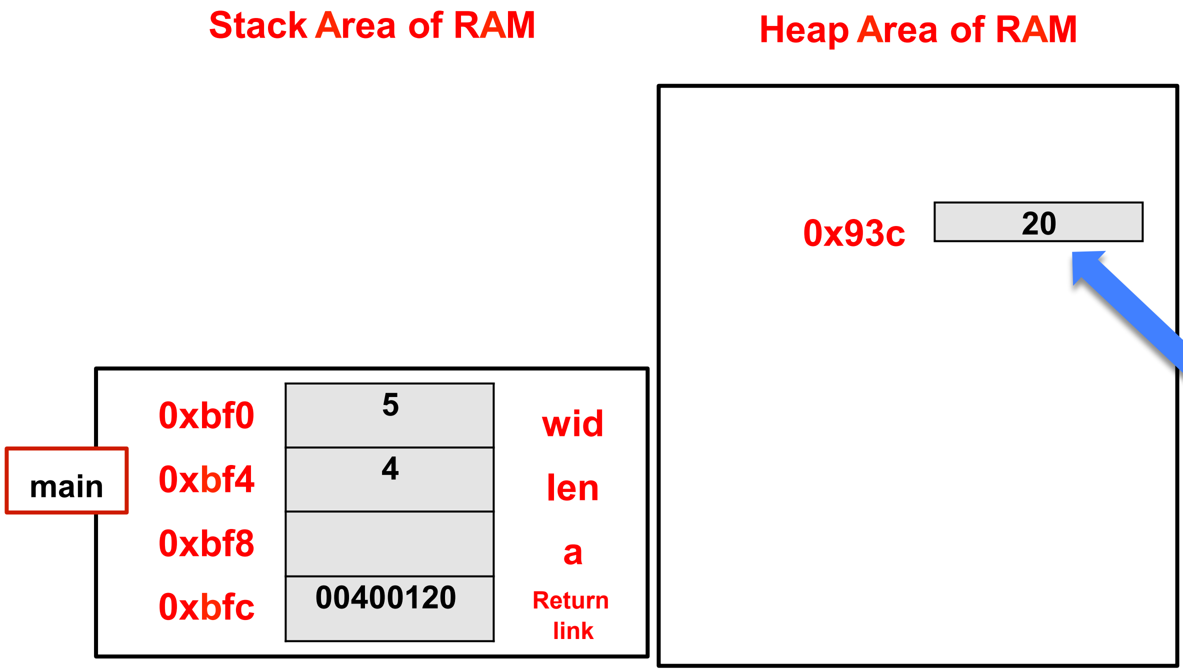
Dynamic Allocation

- Stored in heap
    - Pointer accesses it
- Exists until user ‘delete’ it
    - If it is not deleted, the data exists in heap even pointer dies

```
int  area(int, int);

int  main()
{
   int  wid  =  5,  len  =  4,
     a; a  =  area(wid,len);
}

int area(int  w,  int  l)
{
  int* ans = new int;
   *ans  =  w  *  l;
   return *ans;
}
```



Memory Leak
No pointer to it. Memory space is wasted.

Dynamic allocation

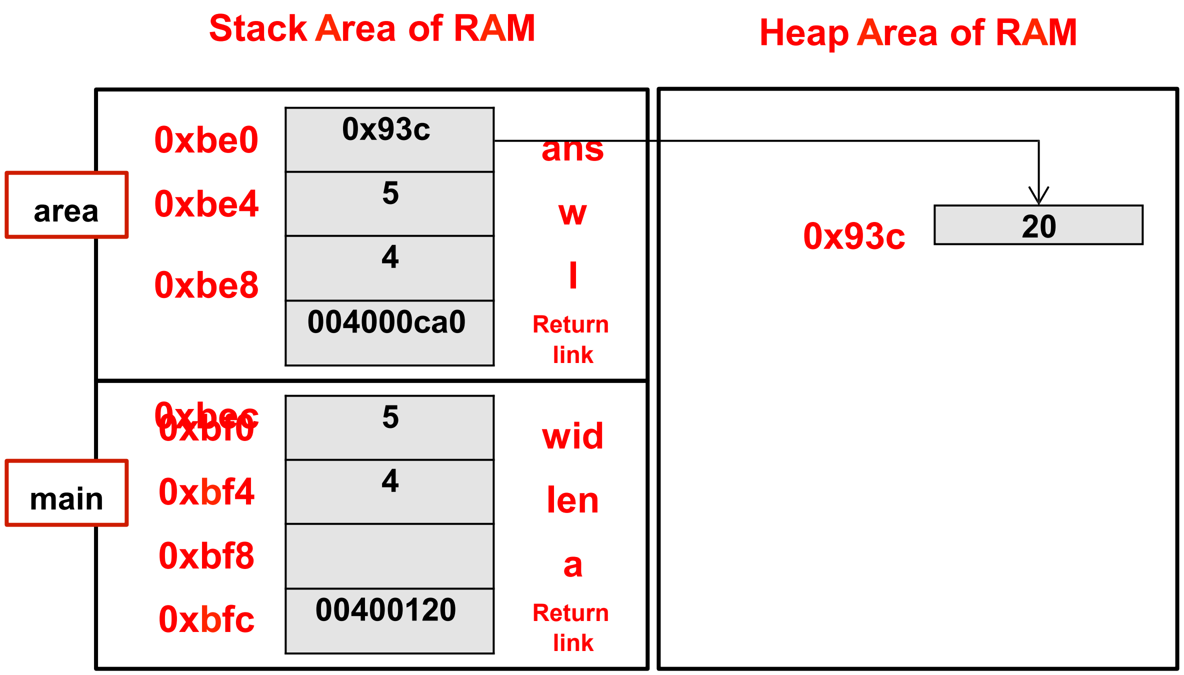
Dynamic Allocation

- Stored in heap
  - Pointer accesses it
- Exists until user 'delete' it
  - If it is not deleted, the data exists in heap even pointer dies

```
int  area(int, int);

int  main()
{
   int  wid  =  5,  len  =  4,
    a; a  =  area(wid,len);
}

int area(int  w,  int  l)
{
  int* ans = new int;
   ans  =  &l;
   return *ans;
}
```



Dynamic allocation

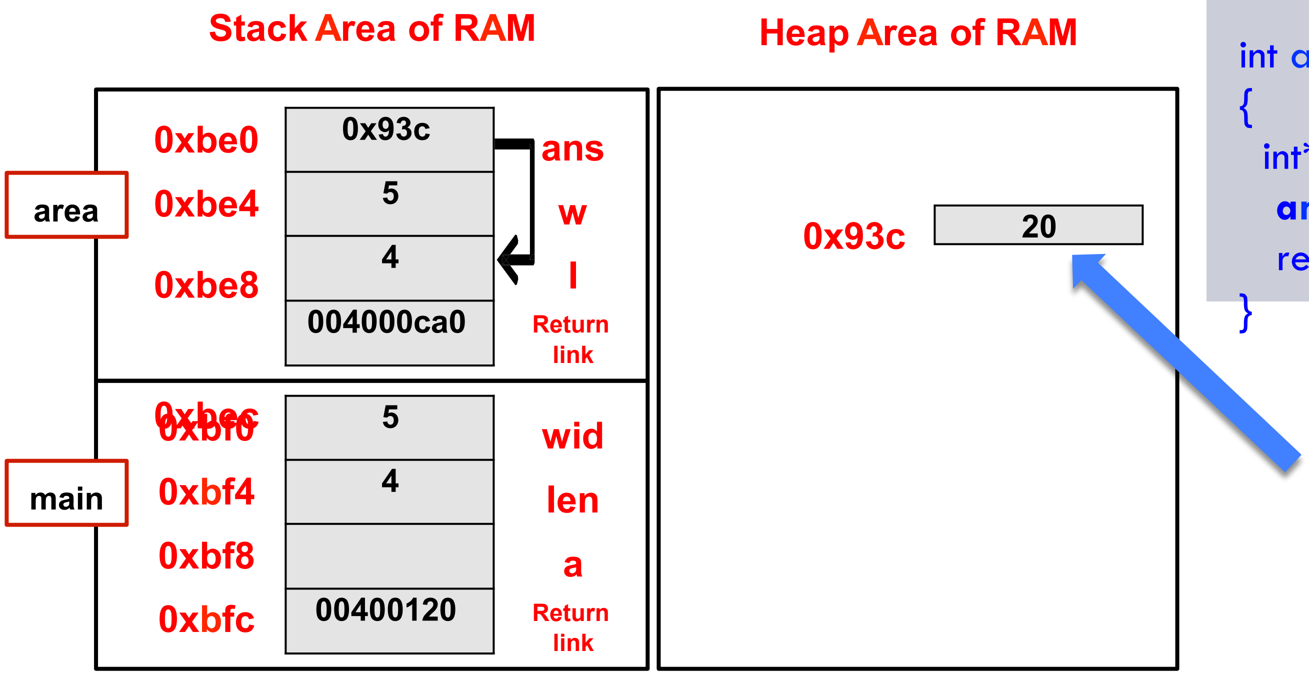
Dynamic Allocation

- Stored in heap
  - Pointer accesses it
- Exists until user 'delete' it
  - If it is not deleted, the data exists in heap even pointer dies

```
int  area(int, int);

int  main()
{
   int  wid  =  5,  len  =  4,
    a; a  =  area(wid,len);
}

int area(int  w,  int  l)
{
  int* ans = new int;
   ans  =  &l;
   return *ans;
}
```



Memory Leak

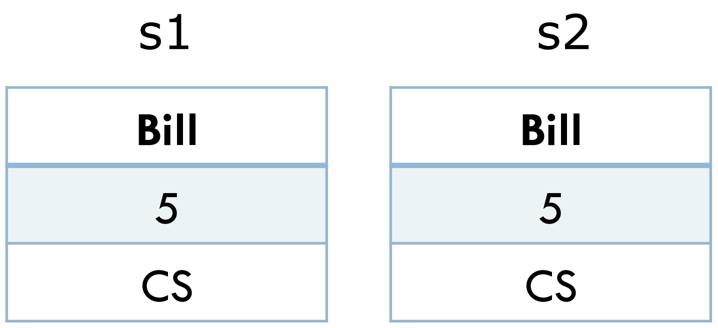
No pointer to it.

Memory space is wasted.

Object assignment

Assigning one struct or class object to another will cause an element by element copy of the source data.

```
#include<iostream>
using  namespace  std;
enum  {CS, MATH, BIO};
struct  student  {
    char  name[80];
    int  id;
    int  major;
};
int  main(int  argc,  char  *argv[])
{
    student  s1;
    strncpy(s1.name,"Bill",80);
    s1.id  =  5;
    s1.major  =  CS;
    student  s2  =  s1;
    return  0;
}
```



Objects in C++

- **Objects** can be created as local variables just like any basic data types in C++.

C++:

```
ComplexType num1;
```

Java: Nothing equivalent – Objects cannot be in **stack.**

Objects in **Heap**

C++:

ComplexType *num1 = new ComplexType(…);

Java:

ComplexType num1 = new ComplexType(…);

Arrays

- Basic data types and classes are treated the same way in C++, unlike Java.

C++: ComplexType **colors[5];**

Java: nothing equivalent.