# Memory Model

Object-Oriented Programming with C++

# Memory Model

# What are these variables?

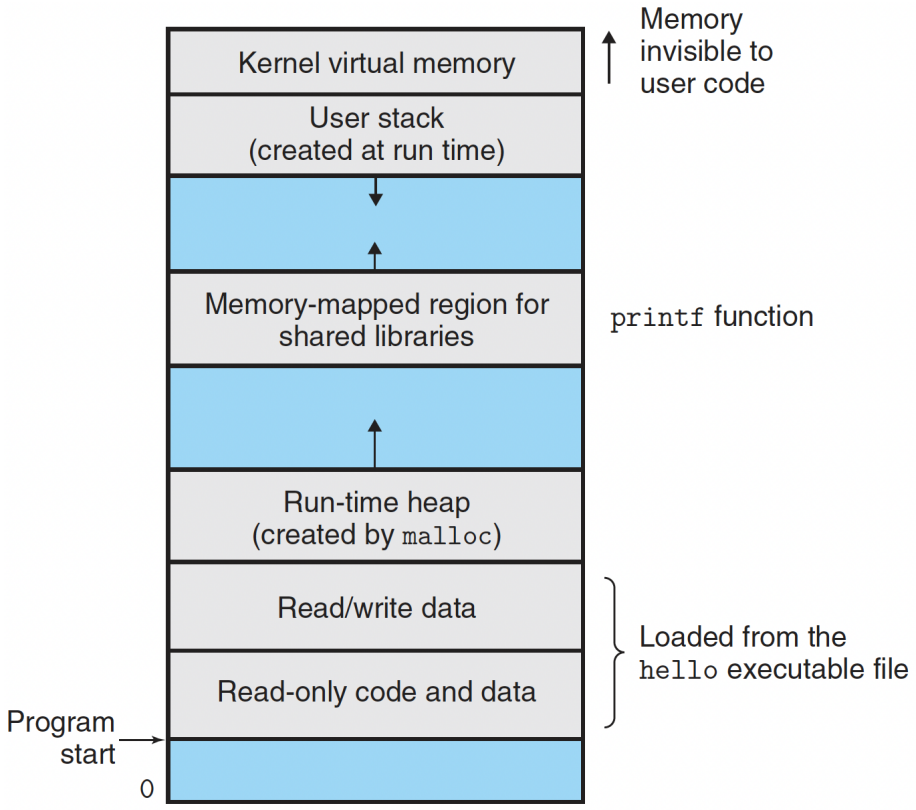
```
int i;              // global vars.
static int j;       // static global vars.

void f()
{
  int k;            // local vars.
  static int l;     // static local vars.

  int *p = malloc(sizeof(int)); // allocated vars.
}
```

# Where are they in memory?

- stack
  - local vars
- heap
  - dynamically allocated vars.
- code/data
  - global vars
  - static global vars
  - static local vars



# Global vars

- vars defined outside any functions
- can be shared btw .cpp files
- extern

# Extern

- extern is a declaration says there will be such a variable somewhere in the whole program
- “such a” means the type and the name of the variable
- global variable is a definition, the place for that variable

# Static

- static global variable inhibits access from outside the .cpp file
- so as the static function

# Static local vars

- static local variable keeps value in between visits to the same function
- is initialized at its first access

# Static

- for global stuff:
  - access restriction
- for local stuff:
  - persistence

# Pointers to Objects

# Pointers to objects

```
string s = "hello";
string* ps = &s;
```

# Operators with pointers

- get address

```
ps = &s;
```

- get the object

```
(*ps).length()
```

- call the function

```
ps->length()
```

# Two ways to access

- `string s;`
  - s is the object itself
  - At this line, object s is created and initialized
- `string *ps;`
  - ps is a pointer to an object
  - the object ps points to is not known yet.

# Assignment

```
string s1, s2;
s1 = s2;

string *ps1, *ps2;
ps1 = ps2;
```

# Dynamically Allocated Memory

# Dynamic memory allocation

- *new* expression

```
new int;
```

```
new Stash;
```

```
new int[10];
```

- *delete* expression

```
delete p;
```

```
delete[] p;
```

# `new` and `delete`

- `new` is the way to allocate memory as a program runs. Pointers become the only access to that memory.
- `delete` enables you to return memory to the memory pool when you are finished with it.

# Dynamic arrays

- The new operator returns the address of the first element of the block.

```
int *psome = new int[10];
```

- The presence of the brackets tells the program that it should free the whole array, not just the element

```
delete[] psome;
```

# The *new-delete* mechanism

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

Student *q = new Student();
Student *r = new Student[10];

delete p;
delete[] a;

delete q;
delete r;
delete[] r;
```

# Tips for *new* and *delete*

- Don't mix-use `new/delete` and `malloc/free`.
- Don't `delete` the same block of memory twice.
- Use `delete` (no brackets) if you've used `new` to allocate a single entity.
- Use `delete[]` if you've used `new[]`.
- `delete` the *null pointer* is safe (nothing happens).