C PROGRAMING

Ketan Kore

Sunbeam Infotech

Storage class

	Storage	Initial value	Life	Scope
auto / local	Stack	Garbage	Block	Block
register	CPU register	Garbage	Block	Block
static	Data section	Zero	Program	Limited
extern / global	Data section	Zero	Program	Program

- Each running process have following sections:
  - Text
  - Data
  - Heap
  - Stack
- Storage class decides
  - Storage (section)
  - Life (existence)
  - Scope (visibility)
- Accessing variable outside the scope raise compiler error.

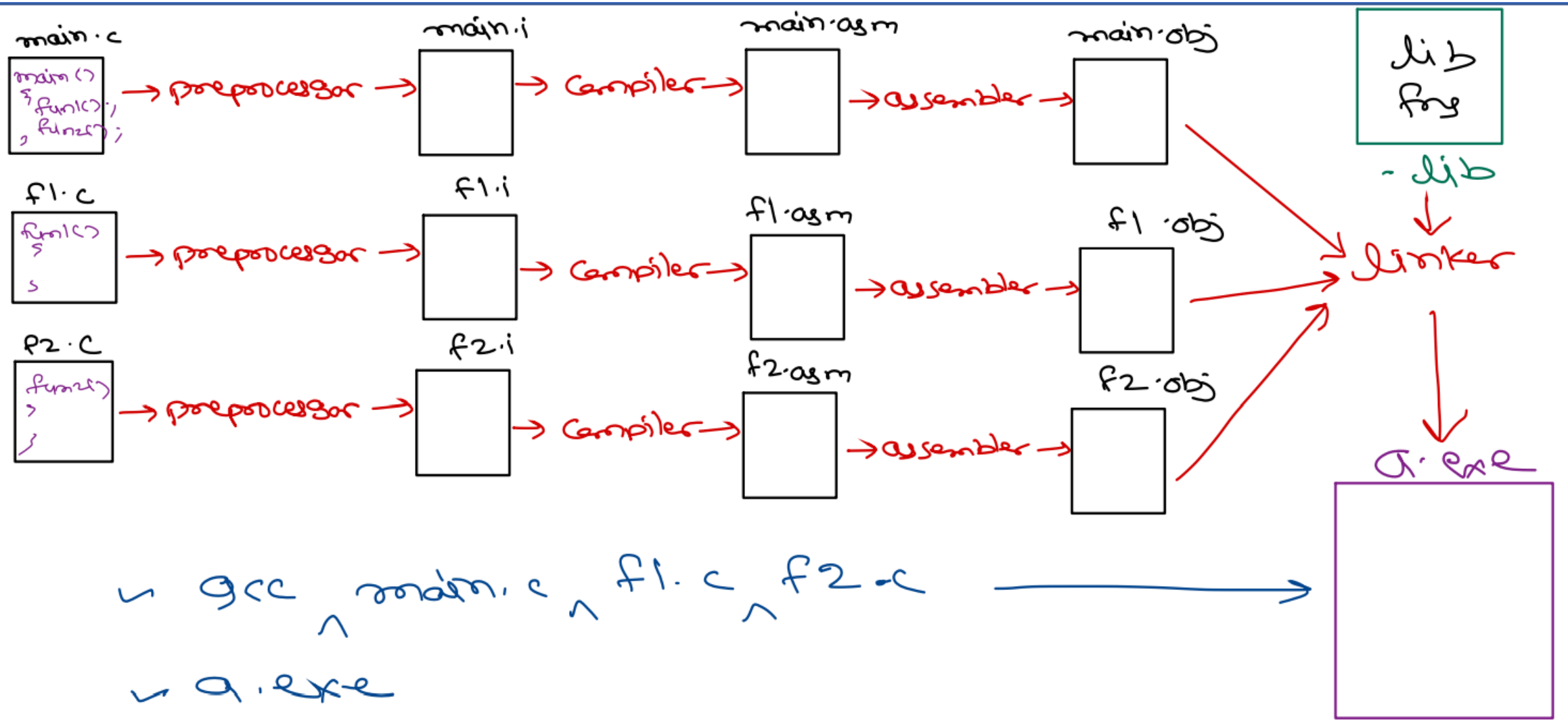
Storage class

- Local variables declared inside the function.
  - Created when function is called and destroyed when function is completed.
- Global variables declared outside the function.
  - Available through out the execution of program.
  - Declared using extern keyword, if not declared within scope.
- Static variables are same as global with limited scope.
  - If declared within block, limited to block scope.
  - If declared outside function, limited to file scope.
- Register is similar to local storage class, but stored in CPU register for faster access.
  - register keyword is request to the system, which will be accepted if CPU register is available.

Recursion

- Function calling itself is called as recursive function.
- To write recursive function consider
  - Explain process/formula in terms of itself
  - Decide the end/terminating condition
- Examples:
  - $n! = n * (n-1)!$ $0! = 1$
  - $x^y = X * x^{y-1}$ $x^0 = 1$
  - $T_n = T_{n-1} + T_{n-2}$ $T_1 = T_2 = 1$
  - factors(n) = 1^{st} prime factor of n * factors(n)

- Advantages
  1. Simplified Readable programs (Divide and conquer Problems)
- Disadvantages
  - Needs More Space (FAR on stack)
  - Needs More Time (FAR Created)


main.c
main()
{
fun1();
fun2();
}
preprocessor
main.i
compiler
main.asm
assembler
main.obj
lib
fns
-lib
f1.c
fun1()
{
}
preprocessor
f1.i
compiler
f1.asm
assembler
f1.obj
linker
f2.c
fun2()
{
}
preprocessor
f2.i
compiler
f2.asm
assembler
f2.obj
a.exe
gcc main.c f1.c f2.c
a.exe

Recursion execution

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int main() {
   int res;
   res = fact(5);
   printf("%d", res);
   return 0;
}
```

5! = 5 * 4!

4! = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

0! = 1

Thank you!

Ketan Kore<Ketan.Kore@sunbeaminfo.com>