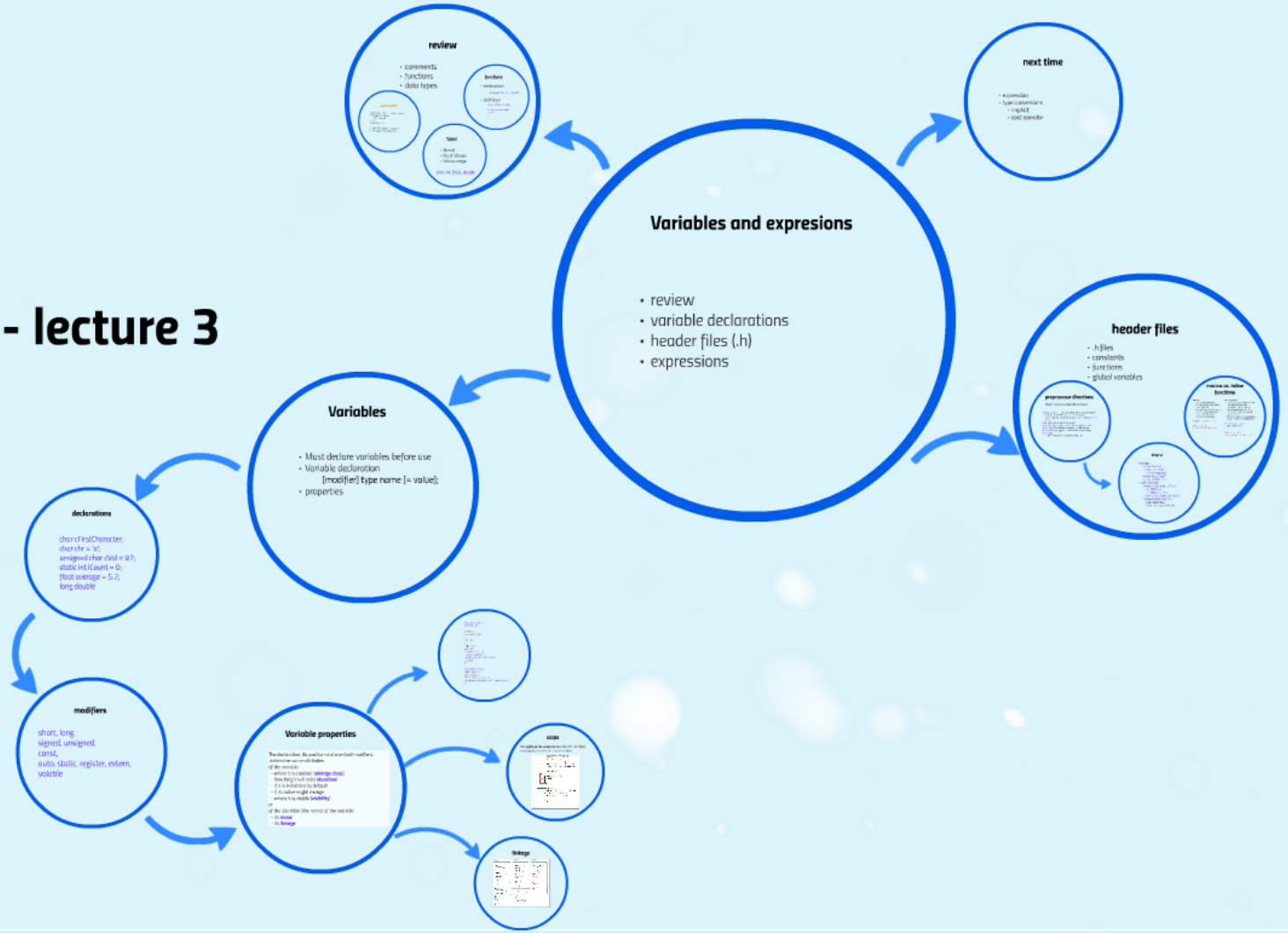
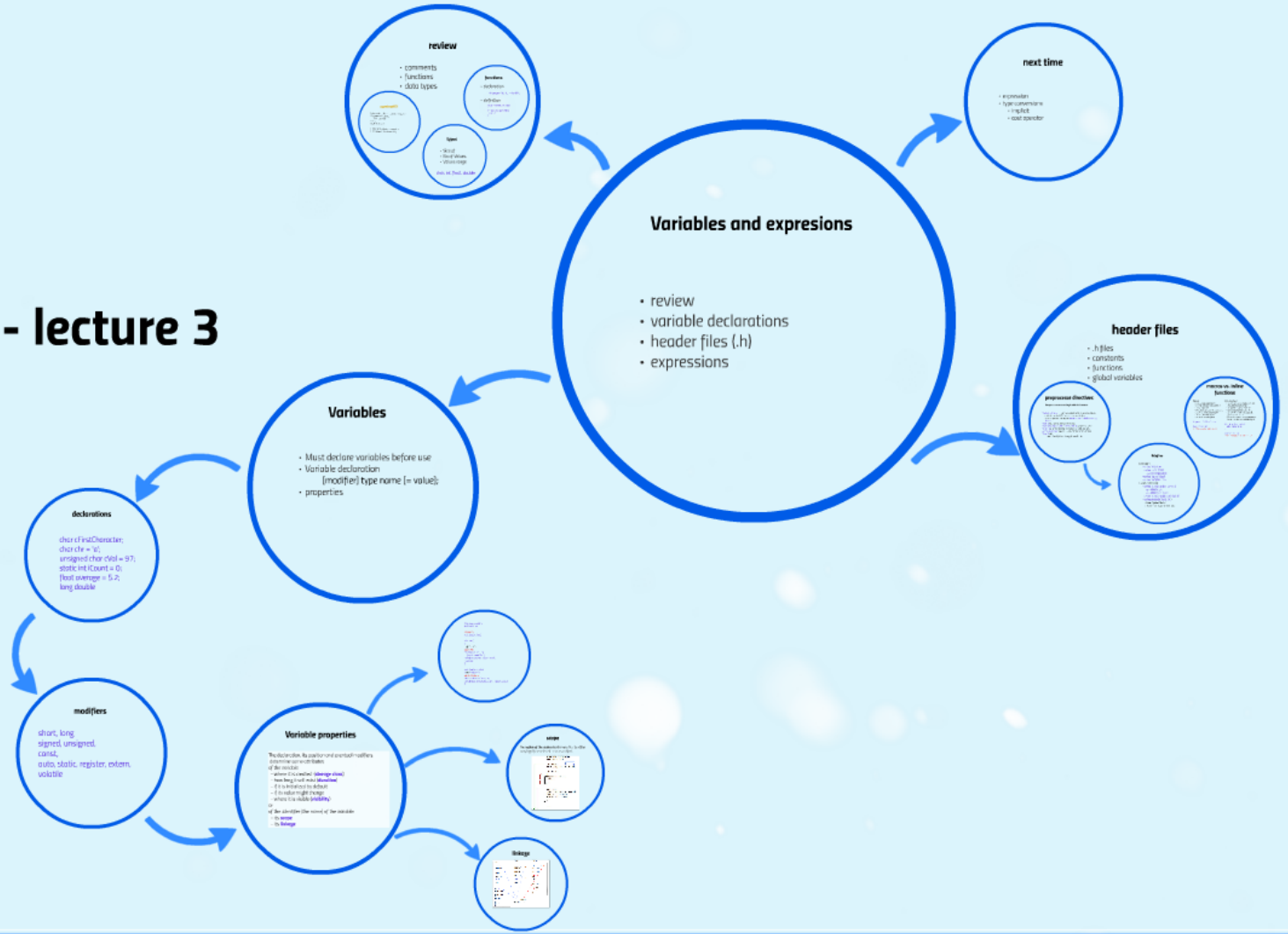
Programming C - lecture 3



Programming C - lecture 3



Variables and expresions

- review
- variable declarations
- header files (.h)
- expressions

review

- comments
- functions
- data types

comments

Comments: /* this is a simple comment */
Can span multiple lines
/* This comment
spans
multiple lines */

- Completely ignored by compiler
- Can appear almost anywhere

functions

- declaration

```
int sum(int iVal1, int iVal2);
```

- definition

```
int sum(int iVal1, int iVal2)
{
/* here is the body of this
funtion */
}
```

types

- Size of
- No of Values
- Values range

char, int, float, double

comments

Comments: /* this is a simple comment */
Can span multiple lines
/ * This comment
spans
multiple lines */

- Completely ignored by compiler
- Can appear almost anywhere

functions

- declaration

```
int sum(int iVal1, int iVal2);
```

- definition

```
int sum(int iVal1, int iVal2)
{
/* here is the body of this
funtion */
}
```

types

- Size of
- No of Values
- Values range

char, int, float, double

Variables

- Must declare variables before use
- Variable declaration

  [modifier] type name [= value];
- properties

declarations

```
char cFirstCharacter;
char chr = 'a';
unsigned char cVal = 97;
static int iCount = 0;
float average = 5.2;
long double
```

modifiers

short, long
signed, unsigned,
const,
auto, static, register, extern,
volatile

Variable properties

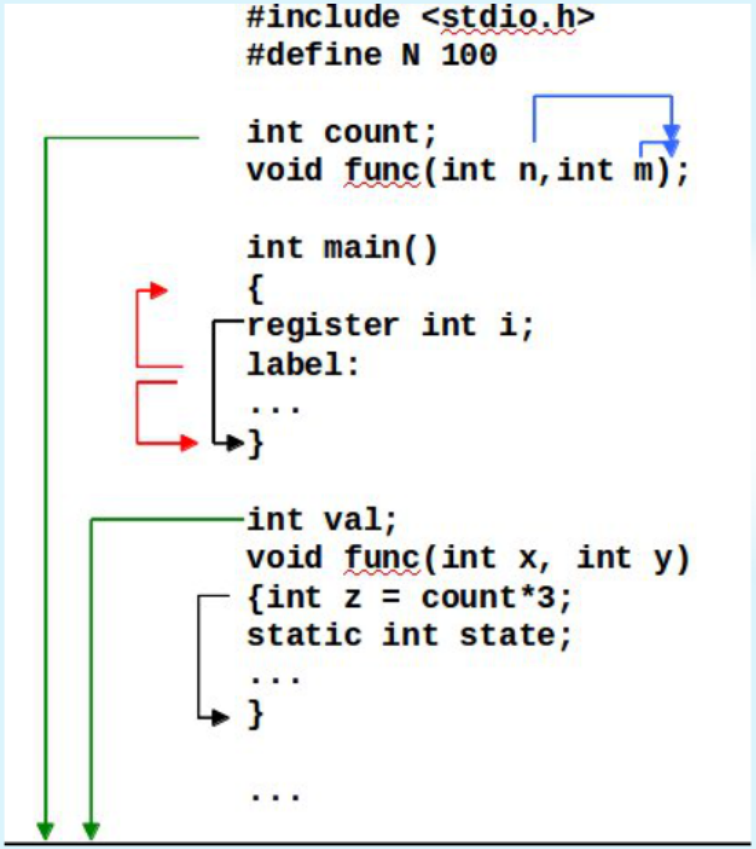
The declaration, its position and eventual modifiers determine some attributes
of the variable:
- where it is created (**storage class**)
- how long it will exist (**duration**)
- if it is initialized by default
- if its value might change
- where it is visible (**visibility**)

or
of the *identifier* (the name) of the variable:
- its **scope**
- its **linkage**

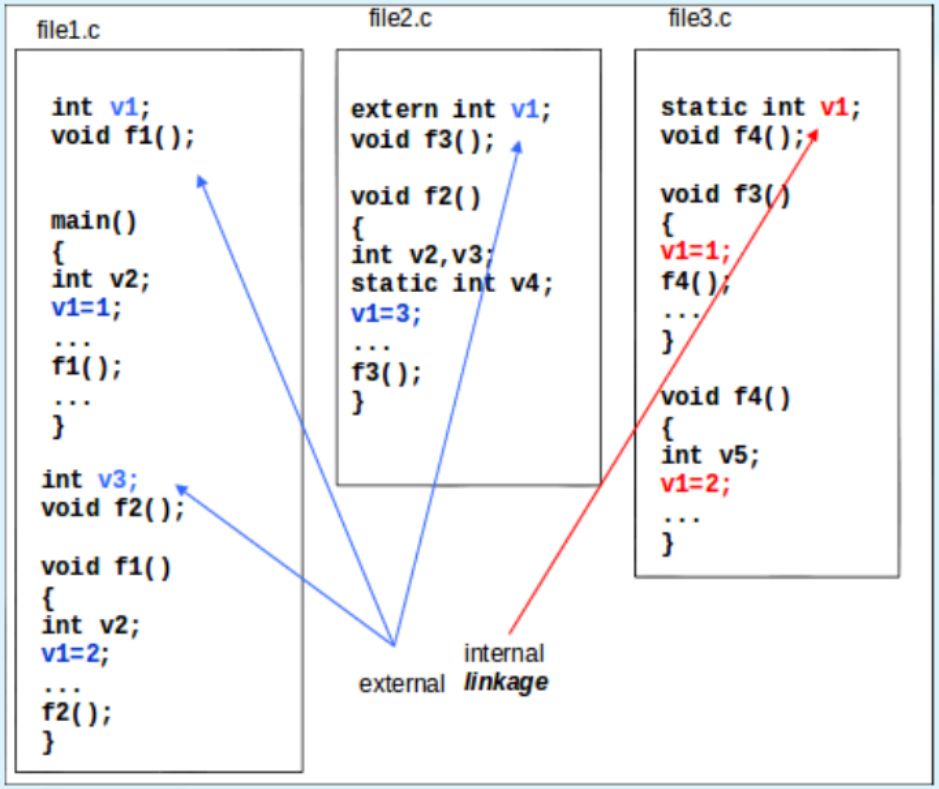
```c
#include <stdio.h>
#define N 100

int count;
void func(int ,int );

int main()
{
register int i;
count=0;
for(i=0; i<N; i = i+1,)
    func(i, count*2+1);
printf("no. of calls: %d\n", count);
return 0;
}

void func(int x, int y)
{int z = count*3;
static int state;
state = state + x*x + y - z;
printf("iteration%d: state=%d", ++count, state);
}
```

scope

the **region of the source text** where the identifier may legally be referred to access object.



linkage


file1.c
file2.c
file3.c
int v1;
void f1();
main()
{
int v2;
v1=1;
...
f1();
...
}
int v3;
void f2();
void f1()
{
int v2;
v1=2;
...
f2();
}
extern int v1;
void f3();
void f2()
{
int v2,v3;
static int v4;
v1=3;
...
f3();
}
static int v1;
void f4();
void f3()
{
v1=1;
f4();
...
}
void f4()
{
int v5;
v1=2;
...
}
external
internal
linkage

header files

- .h files
- constants
- functions
- global variables

preprocesor directives

Preprocessor macros begin with # character

#include <stdio.h> – read the contents of the header file stdio.h
- stdio.h: standard I/O functions for console, files
- other important header files: ctype.h, math.h, stdlib.h, string.h, time.h

#define used for constants and macros
#if , #ifdef , #ifndef , #else , # elif , #endif compiler switches
#pragma providing additional information to the compiler
#error , #warning trigger a custom compiler error/warning
#undef msg
remove the definition of msg at compile time

#define

A constant
```
#define MAX 120
#define LIMIT 1000
    -> int array[LIMIT];
#define Name "Gigel"
#define NoOfBits 16u
```
A function (macro)
```
#define MAX(a, b) ((a < b) ? a : b
    a = MAX(4, 5);
    a = MAX(2+2, 2+3);
#define MAX(a, b) ((a) < (b) ? (a) : (b)
#define firstBit(a) (a) & (1u)
```
- insert parenthesis
- there is no type restriction

macros vs. inline functions

Macros:
- are just ***text substitution***
- are replaced ***before compilation***
- are not type safe
- the compile phase will report errors
- use if you ***don't expect return***
- can expand other macros
- can result in ***side effects***

```
#define max(a,b) ((a<b) ? b : a)

X = max(++a,++b);
/* X=((++a<++b) ? ++b : ++a) */
```

Inline functions:
- are functions whose body is directly ***injected into their call*** site.
- can provide scope for variables
- ***are not guaranteed*** to be inlined
- some inlining may not be possible. (recursive functions)
- Expressions passed as ***arguments*** to inline functions ***are evaluated*** first.

```
inline int max( int a, int b) {
    return ((a<b) ? b : a);
}

X = max(++a, ++b);
/*++a, ++b then X = ((a<b) ? b : a); */
```

preprocesor directives

Preprocessor macros begin with # character

#include <stdio.h> – read the contents of the header file stdio.h

- stdio.h: standard I/O functions for console, files
- other important header files: ctype.h, math.h, stdlib.h, string.h, time.h

#define used for constants and macros

#if , #ifdef , #ifndef , #else , # elif , #endif compiler switches

#pragma providing additional information to the compiler

#error , #warning trigger a custom compiler error/warning

#undef msg

remove the definition of msg at compile time

#define

A constant

```
#define MAX 120
#define LIMIT 1000
    -> int array[LIMIT];
#define Name "Gigel"
#define NoOfBits 16u
```

A function (macro)

```
#define MAX(a, b) ((a < b) ? a : b
    a = MAX(4, 5);
    a = MAX(2+2, 2+3);
#define MAX(a, b) ((a) < (b) ? (a) : (b)
#define firstBit(a) (a) & (1u)
```

- insert parenthesis
- there is no type restriction

macros vs. inline functions

Macros:

- are just ***text substitution***
- are replaced ***before compilation***
- are not type safe
- the compile phase will report errors
- use if you ***don't expect return***
- can expand other macros
- can result in ***side effects***

```
#define max(a,b) ((a<b) ? b : a)

X = max(++a,++b);
/* X=((++a<++b) ? ++b : ++a) */
```

Inline functions:

- are functions whose body is directly ***injected into their call*** site.
- can provide scope for variables
- ***are not guaranteed*** to be inlined
- some inlining may not be possible. (recursive functions)
- Expressions passed as ***arguments*** to inline functions ***are evaluated*** first.

```
inline int max( int a, int b) {
    return ((a<b) ? b : a);
}

X = max(++a, ++b);
/*++a, ++b then X = ((a<b) ? b : a); */
```

next time

- expression
- type conversions
  - implicit
  - cast operator

Programming C - lecture 3