CS101C: Introduction to Programming (Using C)

Autumn 2025

Nikhil Hegde
Achyut Mani Tripathi

Week13: Unions, Preprocessor

Unions

- Format is similar to that of structs.
- Used to create your own types (like structs)
- Used in embedded programming, compiler construction

```
union newtype{
    int i;
    char c;
    float f;
};
```

see union.c in codeexample (shown in next slide)

```
#include<stdio.h>
union newtype{
        int i;
        char c;
        float f;
};
int main(){
        //define a variable of type newtype
        union newtype x;
        //assign value to the integer member of x
        x.i=100;
        //read the value from x's integer member
        printf("x's integer value is:%d\n",x.i);
        //assign value to char member of x
        x.c='C';
        printf("x's character value is:%c\n",x.c);
        //assign value to float member of x
        x.f=1.23;
        printf("x's float value is:%f\n",x.f);
        //print the size of x
        size_t xsize=sizeof(x);
        printf("x's size is:%zu\n",xsize);
```

Recap: if this line were to be newtype x; what code changes would you have to make?

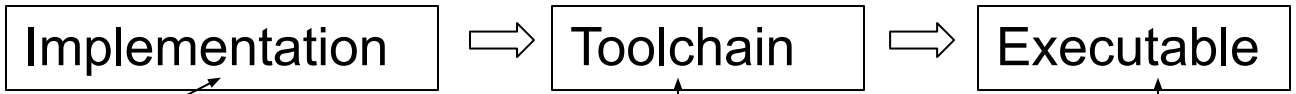
```
        //Fun: assign integer member a value of 0x12345641
        x.i=0x12345641;
        //now print the value in character member of x.
        printf("x's character value is:%c\n",x.c);
        //why do you see the output that you see?

}
```

Recap: what is another way to obtain the size of x without using x here?

Creating a Program (Program Development Environment)

- How to create a program and execute?

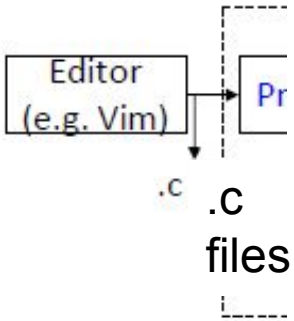


- Tools involved: editors, IDEs, documentation tools
- Tools that are involved: preprocessor, compiler, assembler, loader, linker
- Tools that are involved: Profilers, debuggers, code coverage tools, testing harnesses etc.

Exercise: what is the entry point of execution?

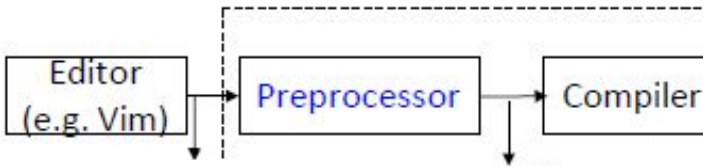
Creating a Program

- Create your c program file



Creating a Program

- Preprocess your c program file



.c .cpp / .cc / .C files

.cpp / .cc / .C files

- Removes comments from your program,
- Expands #include statements
- Substitutes macros
- Performs conditional compilation

Macro Substitution

```
#define identifier substituted_text
```

E.g.

```
#define MAXCHARS 1024
char line[MAXCHARS];
```

see macro_subst2.c in codeexample

Macro Substitution (example2)

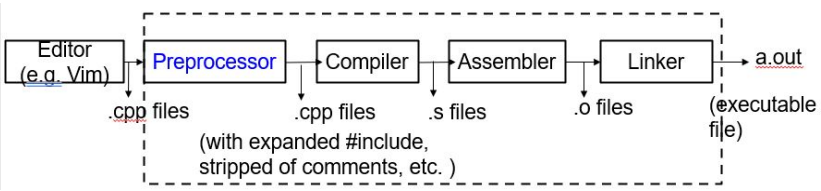
```
#define identifier substituted_text
E.g.
#define ABSDIFF(a, b)  ((a)>(b) ? (a)-(b) : (b)-(a))

int main(){
    int x=10, y=20,z;
    z=ABSDIFF(x,y);
    printf(“%d\n”,z);
}
```

see macro_subst1.c in codeexample. Check “discussed in class” comment.

Conditional Compilation

- Set of 6 preprocessor directives and an operator.
  - #if
  - #ifdef
  - #ifndef
  - #elif
  - #else
  - #endif
- Operator ‘defined’



see hashif.c in codeexample.

#if

```
#if <constant-expression>
printf("CS101");
#endif
```

//This line is compiled only if <constant-expression> evaluates to a non-zero while preprocessing

```
#define COMP 0
#if COMP
printf("CS101");
#endif
```

No compiler error

```
#define COMP 2
#if COMP
printf("CS101")
#endif
```

Compiler throws error about missing semicolon

see hashifdef.c in codeexample.

#ifdef

```
#ifdef identifier
printf("CS101");
#endif
```

//This line is compiled only if identifier is defined before the previous line is seen while preprocessing.

`identifier` does not require a value to be set. Even if set, does not care about 0 or non-zero.

```
#define COMP
#ifdef COMP
printf("CS101")
#endif
```

```
#define COMP 0
#ifdef COMP
printf("CS101")
#endif
```

```
#define COMP 2
#ifdef COMP
printf("CS101")
#endif
```

All three snippets throw compiler error about missing semicolon

Code this yourself!

#else and #elif

```
1. #ifdef identifier1
2. printf(“Summer”);
3. #elif identifier2
4. printf(“Fall”);
5. #else
6. printf(“Spring”);
7. #endif
```

//preprocessor checks if identifier1 is defined. if so, line 2 is compiled. If not, checks if identifier2 is defined. If identifier2 is defined, line 4 is compiled. Otherwise, line 6 is compiled.

Code this yourself!

`defined` operator

Example:

```
#if defined(COMP)
printf(“Spring”);
#endif
```

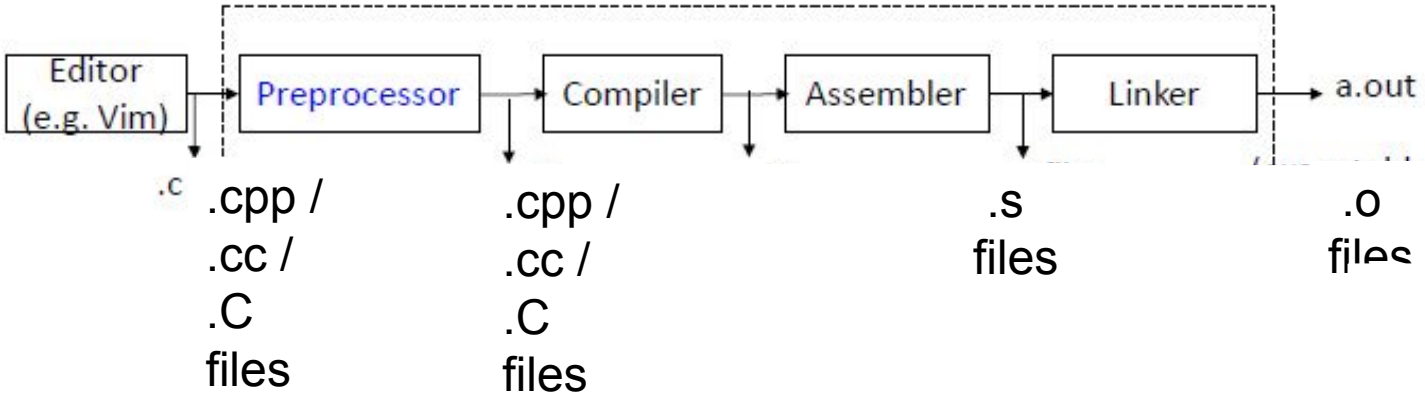
//same as if #ifdef COMP

```
#if defined(COMP1) || defined(COMP2)
printf(“Spring”);
#endif
```

//if either `COMP1` or `COMP2` is defined, the `printf` statement is compiled.
As with `#ifdef`, `COMP1` or `COMP2` values are irrelevant.

Creating a Program

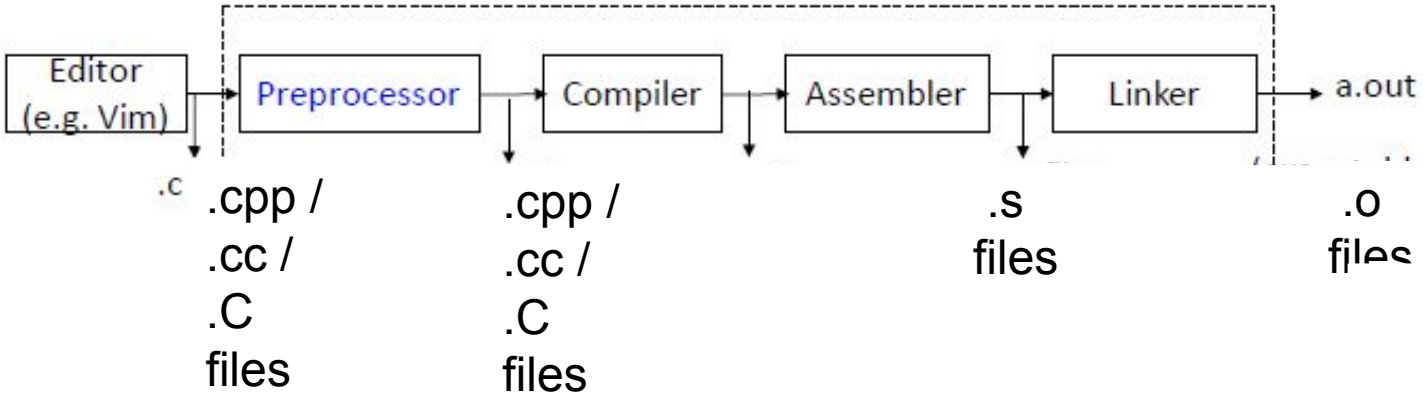
- Get machine code that is part of libraries*



* Depending upon how you get the library code, *linker* or *loader* may be involved.

Creating a Program

- `gcc helloworld.c`

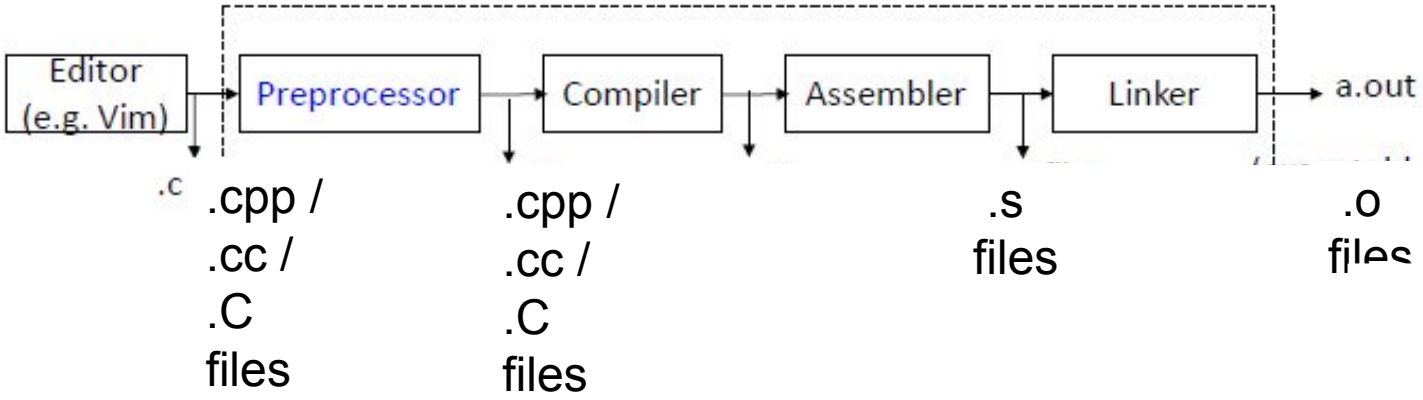


`gcc` is a command to translate your source code (by invoking a collection of tools)

Above command produces `a.out` from `.c` file

Creating a Program

- `gcc helloworld.c -lm`



The `-lm` option tells the linker to link with math library (e.g. if you are using pow function in your .c file)

Creating a Program

- `gcc:` other options
  - `-Wall` - Show all warnings
  - `-o myexe` - create the output machine code in a file called `myexe`

  `More available. We will not see them in this class.`

Creating a Program

- The steps just discussed are ‘compiled’ way of creating a program. E.g. C
- Interpreted way: alternative scheme where source code is ‘interpreted’ / translated to machine code piece by piece e.g. Python
- Pros and Cons.
  - Compiled code runs faster, takes longer to develop
  - Interpreted code runs normally slower, often faster to develop

Creating a Program - Executable

- `a.out` is a pattern of 0s and 1s laid out in memory – sequence of machine instructions
- How do we execute the program?
  - `./a.out` <optional command line arguments>