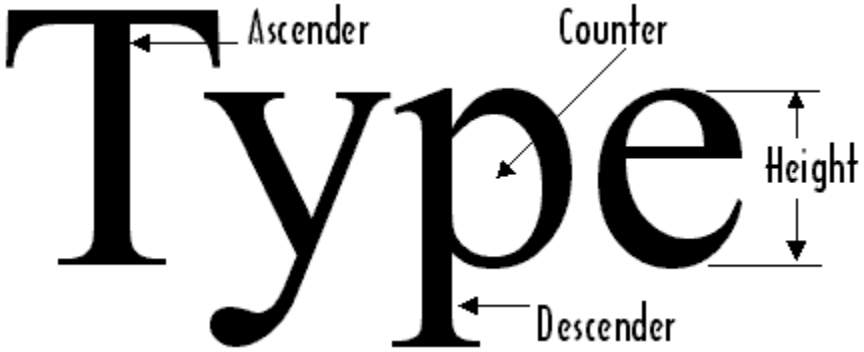
Unions and Typedefs



Symbol Table

- A symbol table consists of several pieces of information

NAME	TYPE	VALUE
X	INT	13
Y	INT	0
DIST	FLOAT	12.87
FIRST_INIT	CHAR	'T'
SSN	LONG INT	222 22 2222

Symbol Table Structure

```
struct symbol {
        char name[256];
        enum typ { t_int,t_float,t_char,t_long_int} type;
        ?????? value;
} symbolTable[100];
```

What type of data should value be?

Symbol Table Structure

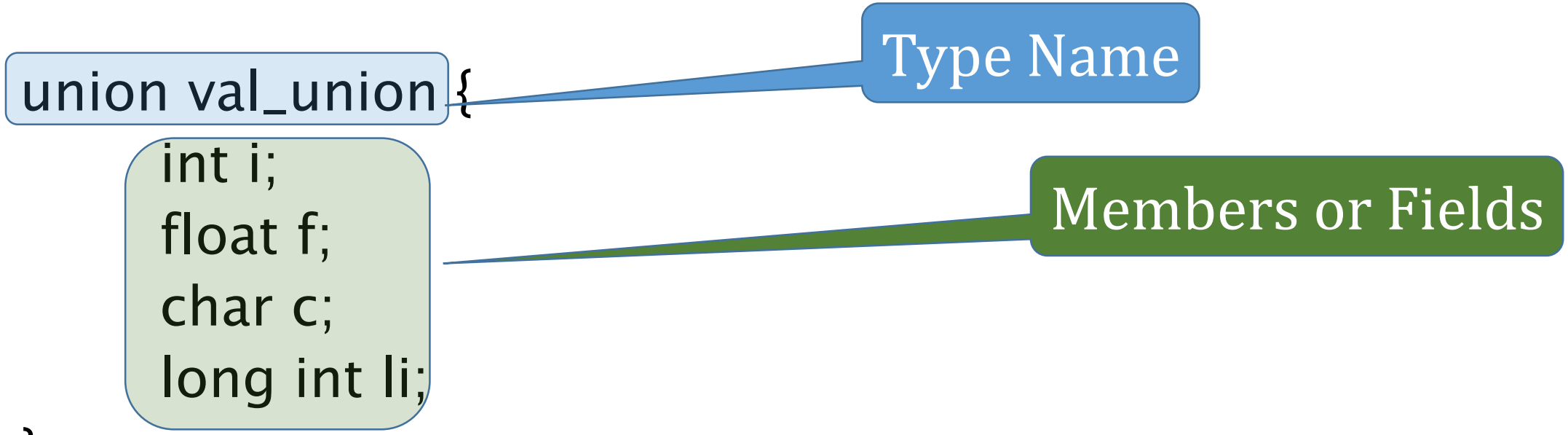
```
struct symbol {
        char name[256];
        enum typ { t_int,t_float,t_char,t_long_int} type;
        int intValue;
        float floatValue;
        char charValue;
        long int liValue;
} symbolTable[100];
```

Why reserve space for each of these?
We will only use one value for each symbol!

Example Union

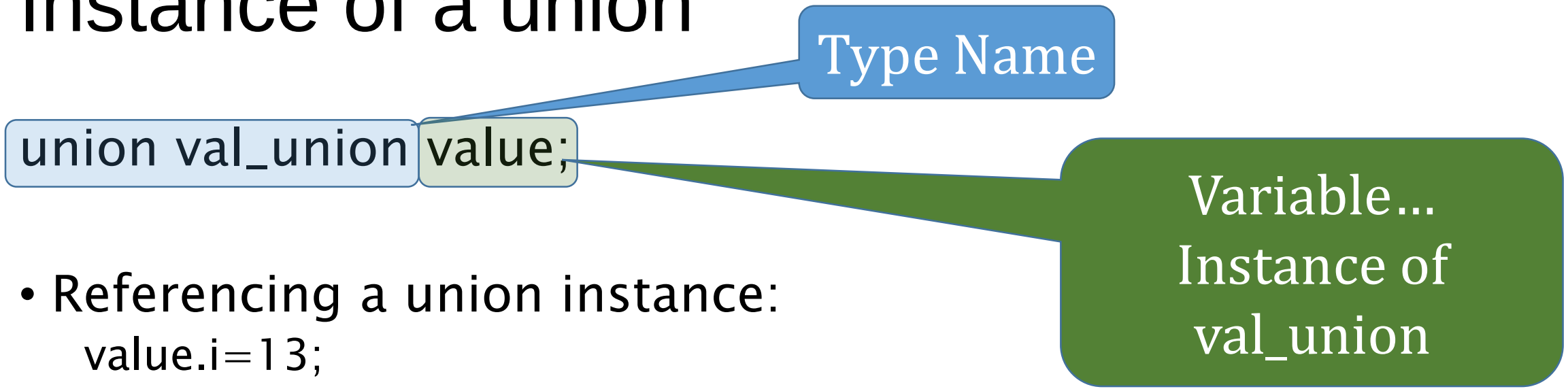
```
union val_union {
        int i;
        float f;
        char c;
        long int li;
};
```

Anatomy of a Union



- Like structures or enums, this defines a new data type
- Like structures, contains a list of sub-fields
- Unlike structures, all sub-fields occupy the same memory

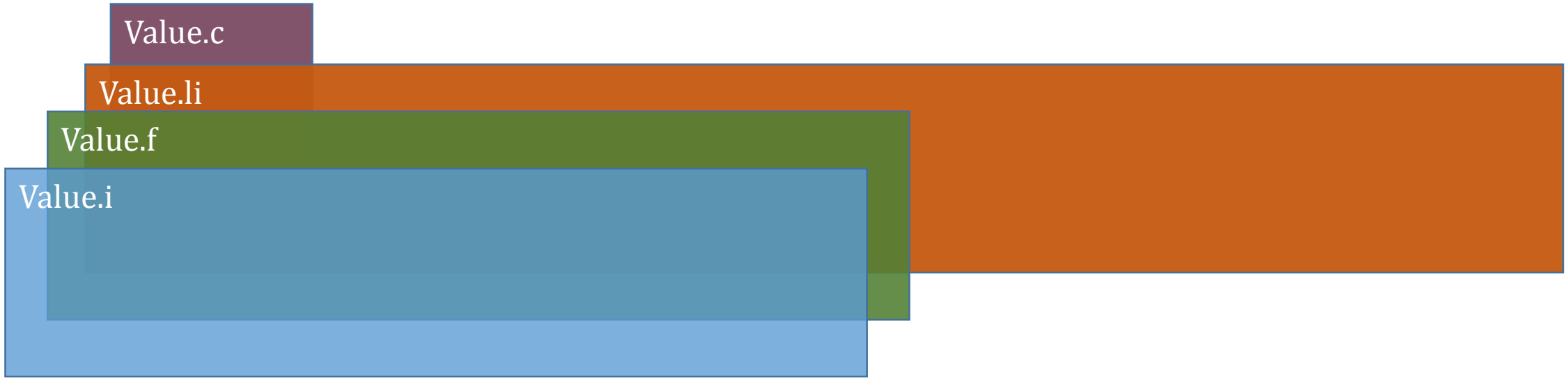
Instance of a union



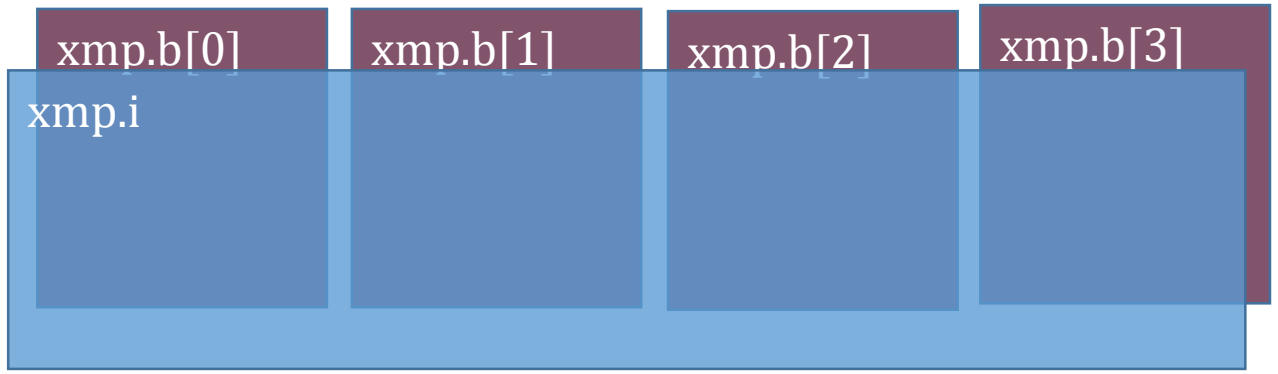
- Referencing a union instance:
  value.i=13;
  or
  value.f=12.87;
- Like structures – reference fields using union name . field
- Like structures, field names are NOT variable names

Union Instances in Memory

- All fields start at the same place
- Entire Union is large enough to hold the largest field



Using Unions to Look at Internals




```
union { int i;
        char b[4];
      } xmp;
xmp.i=347;
printf("347 is represented as %2x %2x %2x %2x\n",
       xmp.b[0],xmp.b[1],xmp.b[2],xmp.b[3]);
```

347 is represented as 5B 01 00 00

Pointers to Unions

- Like structures, we can use -> pointer notation to access fields

```
union val_union * vptr;
vptr=(union val_union*)malloc(sizeof(union val_union));
vptr->c='T';
…
if (vptr->c=='X') printf("Hi Xavier");
free(vptr);
```

Symbol Table Structure

```
struct symbol {
        char name[256];
        enum typ { t_int,t_float,t_char,t_long_int} type;
        union val_union value;
} st[100];
…
if (st[i].type==t_float)
        printf("%s=%f\n",st[i].name, st[i].value.f);
```

Defining New Types

- Built in types: char, int, float, short int, long int, double, etc.
- Arrays – extended types
- Pointers – extended types
- Derived types: structures, enums, unions
  - Must include “struct lnode” or “union val_union”

Derived Types can get Complicated

- struct lnode* nodeList[10];
  - nodelist is an array of 10 pointers to lnode structures

- union val_union **vptrs;
  - vptrs is a pointer to one or more pointers to one or more instances of a val_union union

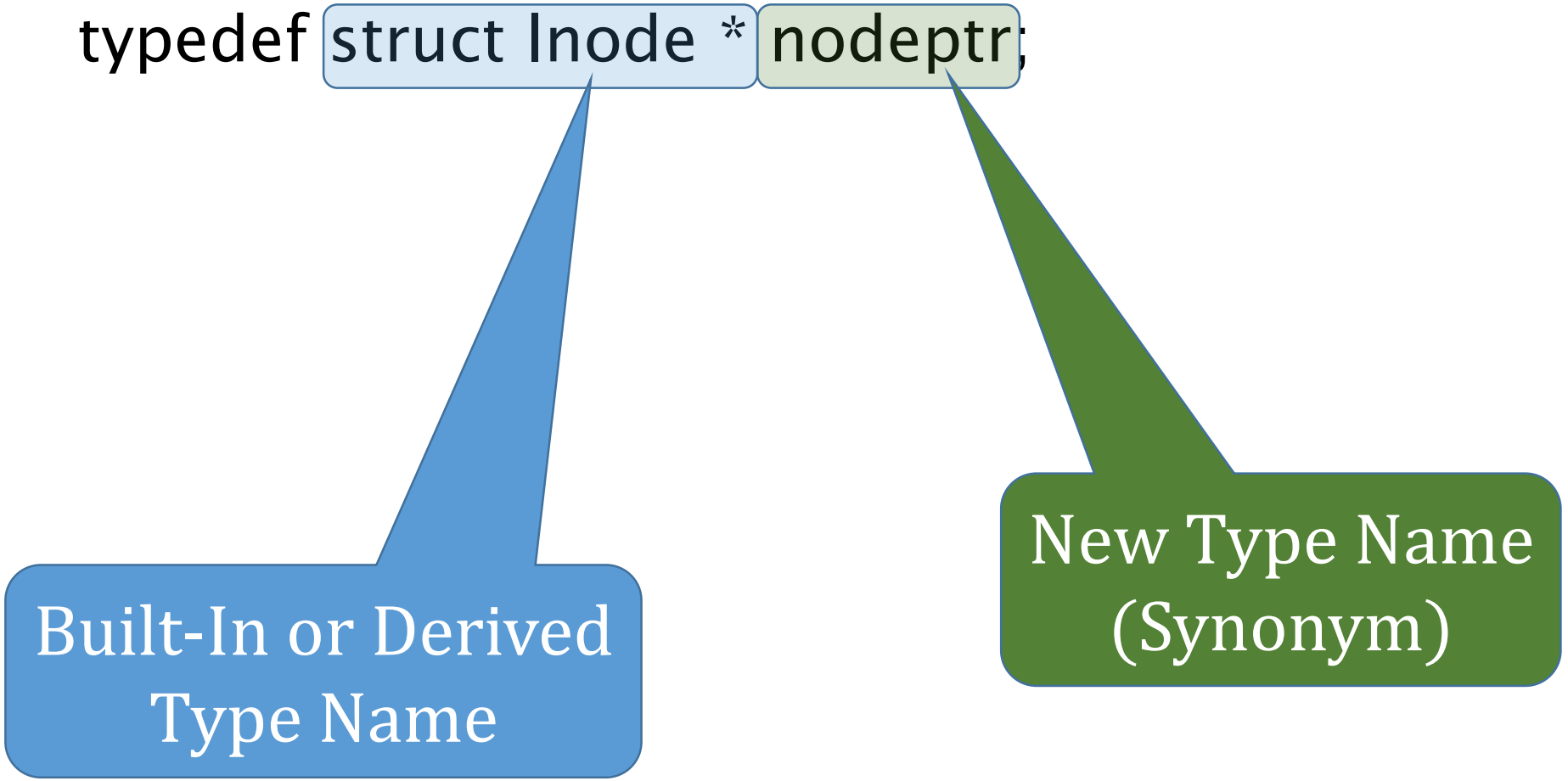
TypeDef… Synonym for Any Type

```
typedef struct lnode * nodeptr;

nodeptr head;
nodeptr newNode=makeLnode(12);
```

Anatomy of a Typedef

```
typedef struct lnode * nodeptr;
```



Why TypeDef

1. Makes code much more readable
   - It's much clearer to read an write "nodeptr" than "struct lnode *"
2. Improves portability of code

```
typedef float length_t;
length_t aToB;
length_t bToC;
…
```

Make this "double" to change all lengths!

Resources

- Programming in C, Chapter 16 (Working with Unions)
- Wikipedia Union Type https://en.wikipedia.org/wiki/Union_type
- Wikipedia Typedef https://en.wikipedia.org/wiki/Typedef