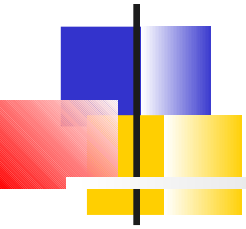
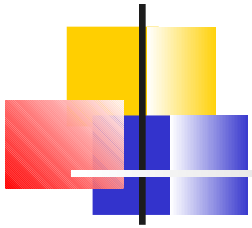


Enumerations and Unions

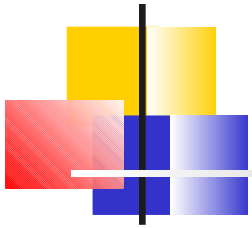
Chapter 16



Objectives

You will be able to:

- Define enumerations to represent discrete values in a C program.
- Define unions to permit the same memory space to hold different types.



Enumerations

- Enumerations allow us to define meaningful names for integer codes used to represent discrete values.

Enumerations

```
#include <stdio.h>

typedef enum
{
     red,
     blue,
     green,
     yellow
} color_t;

int main ( void )
{
    color_t cur_color = green;

    printf ("The color green has the value %d\n", cur_color);
    return 0;
}
```

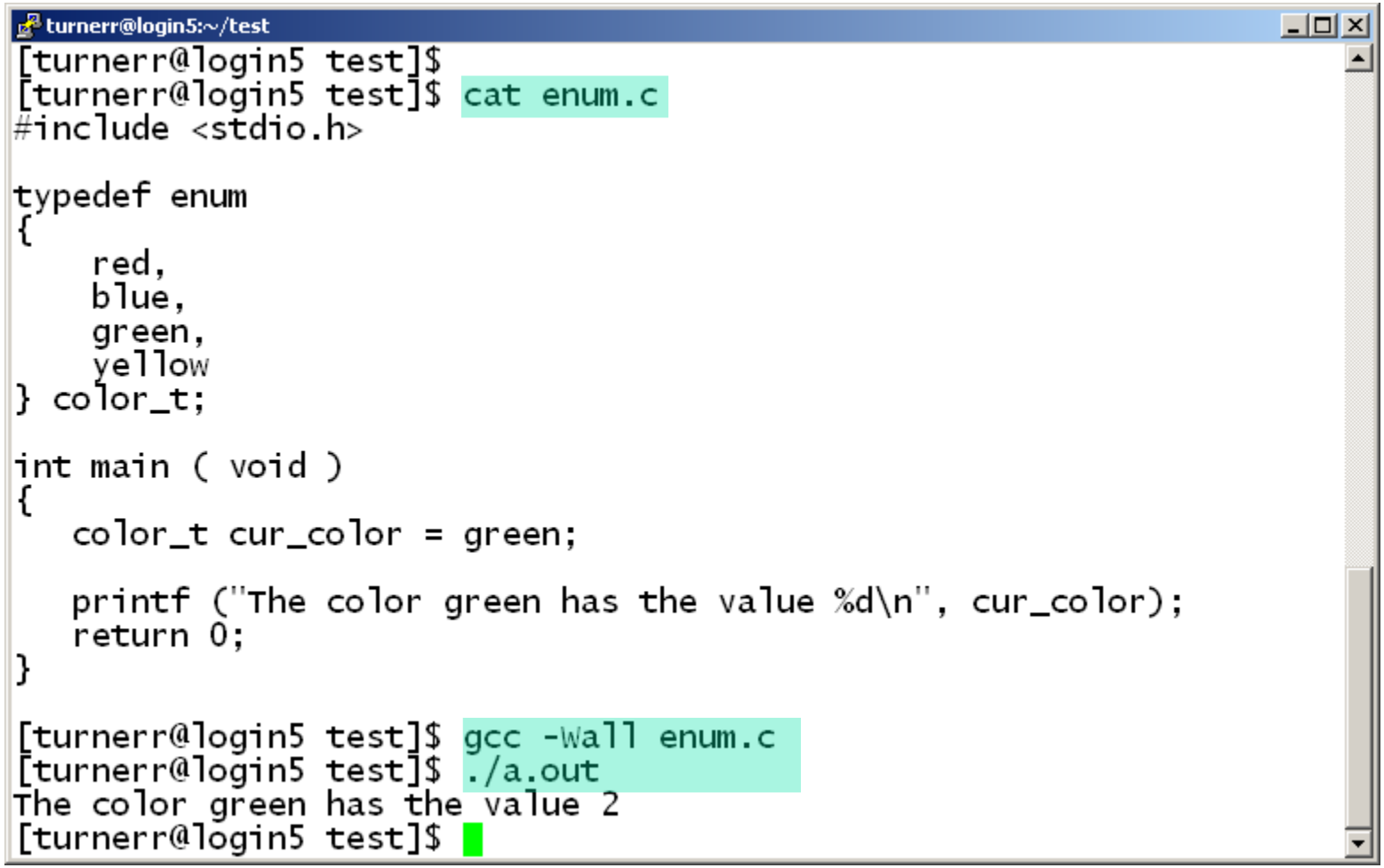
Defines “color” as a name that can be used as a type.

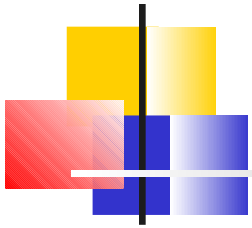
A user-defined type.

Defines a variable of type color.

cur_color can be used like an integer

Program Running





Specifying Values for enums

```
#include <stdio.h>

typedef enum
{
     red = 2,
     blue = 4,
     green = 6,
     yellow
} color_t;

int main ( void )
{
   color_t cur_color = green;

   printf ("The color green has the value %d\n", cur_color);
   return 0;
}
```

Program Running

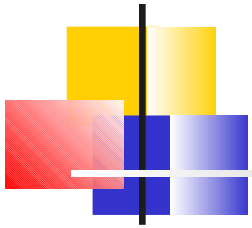
turnerr@login5:~/test

```
[turnerr@login5 test]$
[turnerr@login5 test]$ cat enum.c
#include <stdio.h>

typedef enum
{
    red = 2,
    blue = 4,
    green = 6,
    yellow
} color_t;

int main ( void )
{
   color_t cur_color = green;

   printf ("The color green has the value %d\n", cur_color);
   return 0;
}

[turnerr@login5 test]$ gcc -Wall enum.c
[turnerr@login5 test]$ ./a.out
The color green has the value 6
[turnerr@login5 test]$
```

Value Not Specified

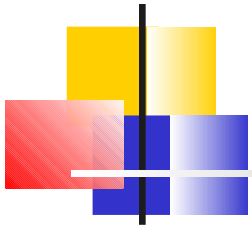
turnerr@login5:~/test

```
[turnerr@login5 test]$ cat enum.c
#include <stdio.h>

typedef enum
{
    red = 2,
    blue = 4,
    green = 6,
    yellow
} color_t;

int main ( void )
{
   color_t cur_color = yellow;

   printf ("The color yellow has the value %d\n", cur_color);
   return 0;
}

[turnerr@login5 test]$
[turnerr@login5 test]$ gcc -Wall enum.c
[turnerr@login5 test]$ ./a.out
The color yellow has the value 7
[turnerr@login5 test]$
```



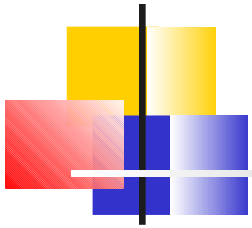
Enumerations

- Enumerations actually *are* integers.
- Defining the enumeration just permits us to create meaningful names for a collection of integer values.



typedef enum

- Similar in effect to #define
- Avoids some of the problems of #define
  - Handled by the compiler rather than by the preprocessor.
  - Not textual substitution
  - Compiler can provide better error messages.



Unions

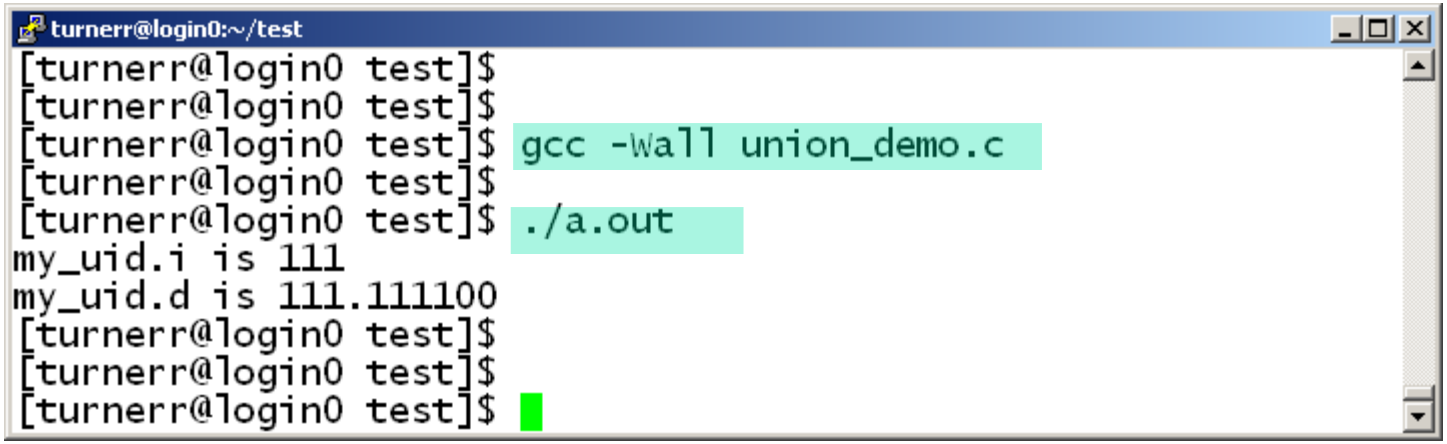
- Unions permit the same memory space to be used to hold different types.

- Looks like a struct.
  - But the different members occupy the same space.
  - Only one of them can be present at any time.

Example: Union

```
#include <stdio.h>

union
{
    int i;
    double d;
} my_uid;

int main()
{
    my_uid.i = 111;
    printf ("my_uid.i is %d\n", my_uid.i);

    my_uid.d = 111.1111;
    printf ("my_uid.d is %f\n", my_uid.d);

    return 0;
}
```

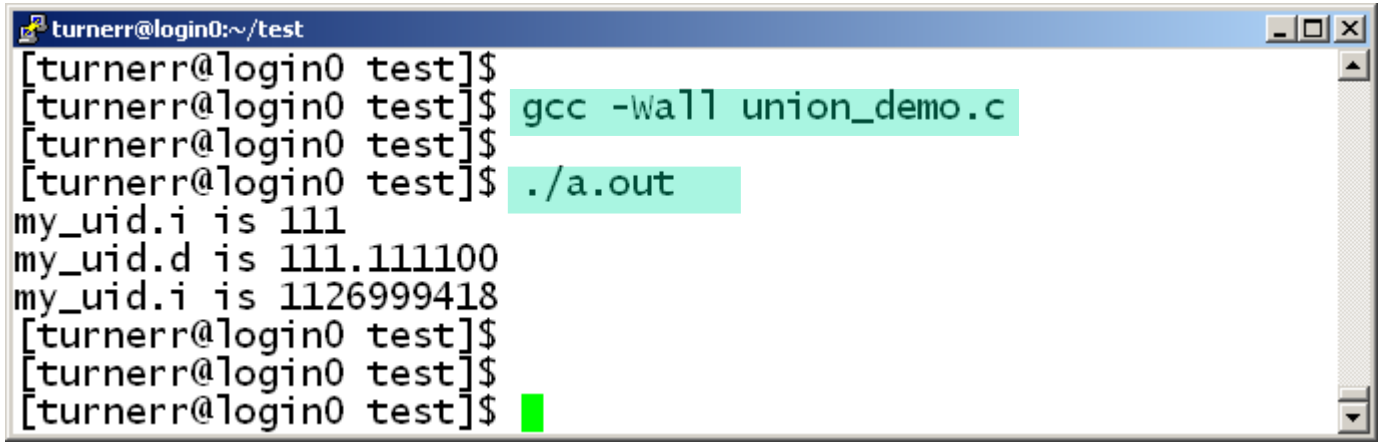
Example: Union



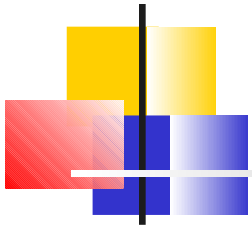
Look at member i again

```
#include <stdio.h>

union
{
    int i;
    double d;
} my_uid;

int main()
{
    my_uid.i = 111;
    printf ("my_uid.i is %d\n", my_uid.i);

    my_uid.d = 111.1111;
    printf ("my_uid.d is %f\n", my_uid.d);

    printf ("my_uid.i is %d\n", my_uid.i);
    return 0;
}
```

Look at member i again



You have to know which way the union is being used!



Unions as Struct Members

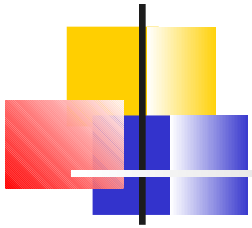
- Unions are normally used as members of structs.
  - Another member of the struct specifies which version of the union is present.
  - Use an enum for the alternatives.

union_demo2.c

```
#include <stdio.h>

typedef enum
{
    integer_kind,
    double_kind
} Kind_of_Number;

typedef struct
{
    Kind_of_Number kind;
    union
    {
        int i;
        double d;
    } number;
} int_or_double;
```

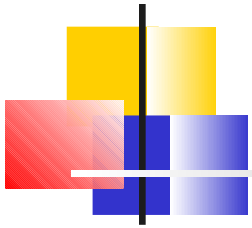
New user-defined type

New user-defined type



Function display_union

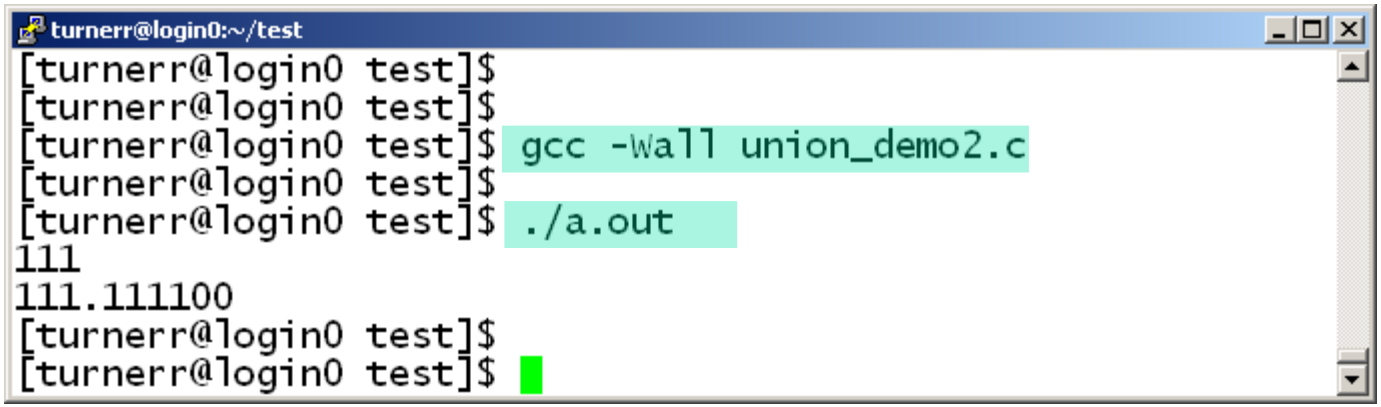
```
void display_union(int_or_double n)
{
    if (n.kind == integer_kind)
    {
        printf ("%d\n", n.number.i);
    }
    else
    {
        printf ("%f\n", n.number.d);
    }
}
```

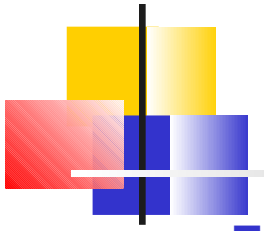


union_demo2.c

```
int main( void )
{
    int_or_double my_uid;

    my_uid.kind = integer_kind;
    my_uid.number.i = 111;

    display_union(my_uid);

    my_uid.kind = double_kind;
    my_uid.number.d = 111.1111;

    display_union(my_uid);
    return 0;
}
```

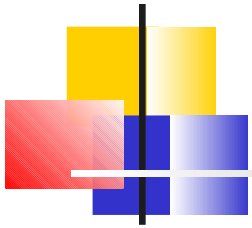
union_demo2.c Running





Summary

- Enumerations
  - Permit us to define meaningful names for numeric codes that represent a set of discrete values.
  - Similar to #define but without some of the problems.
- Unions
  - Permit us to use the same memory space in different ways.
  - Typically within a struct.
  - You must have some way to know which member is present in a union at any time.
    - Typically an enum within the same struct.



Assignment

- Read Chapter 16.
- Do the examples from this presentation for yourself.

End of Presentation