Loose ends

Adapted from materials by Dr. Carrier

Switch statements

Similar to if - else if -else blocks

```
char c = SomeFunction();
switch(c){
  case ‘x’:
    DigForTreasure();
    break;
  case ‘r’:
    StartPirateMode();
    break;
  default:
    printf(“Not a pirate letter :(“);
}
```

Switch statements

What’s different here? What happens?

```
char c = SomeFunction();
switch(c){
  case ‘x’:
    DigForTreasure();
  case ‘r’:
    StartPirateMode();
  default:
    printf(“Not a pirate letter :(“);
}
```

Switch statements

What’s different here? What happens?

```
char c = SomeFunction();
switch(c){
  case ‘x’:
    DigForTreasure();
  case ‘r’:
    StartPirateMode();
  default:
    printf(“Not a pirate letter :(“);
}
```

Be careful! If you leave out a `break`, you’ll “fall through”

Enumerations (enums)

- Enums let you name “options”

Enumerations (enums)

- Enums let you name “options”
    - Can increase readability

Enumerations (enums)

- Enums let you name “options”
    - Can increase readability
    - … but doesn’t actually add any new functionality

Enumerations (enums)

- Enums let you name “options”
  - Can increase readability
  - … but doesn’t actually add any new functionality
- All values are just an integer underneath

Enumerations (enums)

- Enums let you name “options”
  - Can increase readability
  - … but doesn’t actually add any new functionality
- All values are just an integer underneath
  - Typically start at 0 and count up

Enumerations (enums)

- Enums let you name “options”
    - Can increase readability
    - … but doesn’t actually add any new functionality
- All values are just an integer underneath
    - Typically start at 0 and count up
    - But we can override that if we want!

Enumerations (enums)

- Enums let you name “options”
  - Can increase readability
  - … but doesn’t actually add any new functionality
- All values are just an integer underneath
  - Typically start at 0 and count up
  - But we can override that if we want!

```
enum rank {
  CAPTAIN,
  FIRST_MATE,
  QUARTERMASTER
};
```

Enumerations (enums)

- Enums let you name “options”
  - Can increase readability
  - … but doesn’t actually add any new functionality
- All values are just an integer underneath
  - Typically start at 0 and count up
  - But we can override that if we want!

```
enum rank {
  CAPTAIN,
  FIRST_MATE,
  QUARTERMASTER
};
enum rank my_rank = QUARTERMASTER;
```

The return of typdef

What does typedef do?

The return of typdef

What does typedef do?

Gives an existing type a new name!

The return of typdef

What does typedef do?

Gives an existing type a new name!

We can use that here to trim down ‘`enum rank`’

The return of typdef

What does typedef do?

Gives an existing type a new name!

We can use that here to trim down ‘`enum rank`’

```
typedef enum rank {
  CAPTAIN = 5,
  FIRST_MATE = 4,
  QUARTERMASTER = 3
} rank;
rank your_rank = CAPTAIN;
```

Unions

What does a struct do?

Unions

What does a struct do?

Store multiple pieces of data together as a bundle

Unions

What does a struct do?

Store multiple pieces of data together as a bundle

Unions

What does a struct do?

Store multiple pieces of data together as a bundle

A union does the opposite: Stores multiple types of data in the same place

Unions

What does a struct do?

Store multiple pieces of data together as a bundle

A union does the opposite: Stores multiple types of data in the same place

- All types use the same memory
- Can only store one at a time
- You are responsible for interpreting it!

Union example

```
union data {
  int i;
  double d;
  char c;
};
union data var;
data.d = 3.14;
```

Other keywords

A variable preceded by `const` cannot be changed

- Can make code more readable
- Compiler keeps you from changing it
- Optimizations

Other keywords

A variable preceded by `const` cannot be changed

- Can make code more readable
- Compiler keeps you from changing it
- Optimizations

A variable preceded by `static` maintains its value outside of the normal scope

- E.g., a static int in a function will have the same value across function calls

Other libraries

Working with standard libraries is easy

- No extra work
- E.g., stdio.h

You can also use custom libraries

- May need to pass additional flags to gcc
    - Linker flags: `-l` or `-L` options
    - Includes: `-I` options