listings-rust demo

Tobias Denkinger
tobias.denkinger@tu-dresden.de

April 29, 2024

Some identifiers

```
// keywords
as break const continue crate dyn else enum extern fn for if impl in let loop
match mod move mut pub ref return Self self static struct super trait type
union unsafe use where while

// reserved keywords (for future use)
abstract alignof become box do final macro offsetof override priv proc pure
sizeof typeof unsized virtual yield

// strings
"this␣is␣a␣test"

// characters
'a' 'b' 'c'

// primitive types
bool char f32 f64 i8 i16 i32 i64 isize str u8 u16 u32 u64 unit usize i128 u128

// some common type and value constructors
Box Option Result String Vec Some None Ok Err true false

// some common traits
Copy Send Sized Sync Drop Fn FnMut FnOnce ToOwned Clone PartialEq PartialOrd
Eq Ord AsRef AsMut Into From Default Iterator Extend IntoIterator
DoubleEndedIterator ExactSizeIterator SliceConcatExt ToString

// some common macros
assert! assert_eq! assert_ne! cfg! column! compile_error! concat!
concat_idents! debug_assert! debug_assert_eq! debug_assert_ne! env! eprint!
eprintln! file! format! format_args! include! include_bytes! include_str!
line! module_path! option_env! panic! print! println! select! stringify!
thread_local! try! unimplemented! unreachable! vec! write! writeln!
```

Example file

```rust
use std::rc::Rc;

/// upside-down tree with a designated position (the *stack pointer*)
/// and *nodes* of type ‘A‘.
#[derive(Clone, Debug)]
pub struct TreeStack<A> {
    parent: Option<(usize, Rc<TreeStack<A>>)>,
    value: A,
    children: Vec<Option<Rc<TreeStack<A>>>>,
}

impl<A> TreeStack<A> {
    /// Creates a new ‘TreeStack<A>‘ with root label ‘a‘.
    pub fn new(a: A) -> Self {
        TreeStack { value: a, children: Vec::new(), parent: None }
    }

    /// Applies a function ‘FnMut(&A) -> B‘ to every node in a ‘TreeStack<A>‘.
    pub fn map<F, B>(&self, f: &mut F) -> TreeStack<B>
        where F: FnMut(&A) -> B,
    {
        let new_value = f(&self.value);
        let new_parent = match self.parent {
            Some((i, ref p)) => Some((i, Rc::new(p.map(f)))),
            None => None,
        };
        let new_children = self.children
                               .iter()
                               .map(|o| o.clone().map(|v| Rc::new(v.map(f))))
                               .collect();
        TreeStack {
          parent: new_parent,
          value: new_value,
          children: new_children
        }
    }
}
```