0.1 Inline fragments

It is possible to insert inline code listings by wrapping them with the paragraph character `int *pointer = nullptr;`. It is equally simple to create a fragment using a `\begin{fragment}{NAME}` block. You can reference a fragment using `\refFragment<NAME>`. To insert LaTeX code inside a block, wrap it with the paragraph character.

Notice the page number of the referenced fragment's definition next to its fragment label in the reference.

```
⟨A fragment⟩≡
  pointer++;
  ⟨A fragment 1⟩
}
```

It is possible to reference undefined fragments as well.

If a referenced fragment is not defined anywhere, we don't try to display a page number.

```
⟨Another fragment⟩≡
void main() {
  ⟨Undefined fragment⟩
}
```

For completeness' sake, you can specify a prefix to customize definitions.
Use `\begin{fragment}[PREFIX]{NAME}`.

```
⟨A fragment⟩+≡
  pointer--;
```

0.2 File fragments

It's nice to separate the source code from the LaTeX code. Especially if you want to refactor or change the code later. For this, there are several commands to include external code.

UTF8 encoding doesn't work well. Save the external source using eg Windows 1252 encoding.

```
⟨A file fragment⟩≡
void main() {
  // do something
  ⟨A fragment 1⟩
}
```

The fragment looks like this in `sourceFragments.cpp`:

I have replaced the paragraph character with ¶ in the listing because I cannot display it.

```
//<A file fragment> =
void main() {
  // do something
  ¶\refFragment<A fragment>¶
}
//<end>
```

The simplest command to include a file fragment is `\fileFragment{FRAGMENT}{FILENAME}`. Alternatively you can use `\defFileFragment<FRAGMENT>{FILENAME}`.

```
⟨Another file fragment⟩≡
int fac( int n ) {
  int value = 1;
  for( ; n > 0 ; n-- ) {
    value *= n;
  }
  return value;
}
```

Usually you are going to include many different fragments from the same file, and there is an even nicer syntax for this: You can set a "current" file using `\setCurrentFragmentFile{FILENAME}` and then include fragments using `\defCurrentFileFragment<FRAGMENT>`.

⟨*A file fragment*⟩≡
```
void main() {
  // do something
  ⟨A fragment 1⟩
}
```

⟨*Another file fragment*⟩≡
```
int fac( int n ) {
  int value = 1;
  for( ; n > 0 ; n-- ) {
    value *= n;
  }
  return value;
}
```

This was created using

```
\setCurrentFragmentFile{sourceFragments.cpp}
\defCurrentFileFragment<A file fragment>
\defCurrentFileFragment<Another file fragment>
```

Last but not least, there is also support for prefixes:

⟨*Another file fragment*⟩+≡
```
int fac_r( int n ) {
  if( n <= 1 ) {
    return 1;
  }
  return n * fac_r( n - 1 );
}
```

Since this fragment has the same name as an already existing fragment, we can't identify it in the external file just using that name. Instead we support an additional hidden tag.

```
//<Another file fragment;recursive> =
int fac_r( int n ) {
  if( n <= 1 ) {
    return 1;
  }
  return n * fac_r( n - 1 );
}
//<end>
```

To display the fragment above the following commands could be used (either one of them):

⟨*Another file fragment*⟩+≡
```
int fac_r( int n ) {
  if( n <= 1 ) {
    return 1;
  }
  return n * fac_r( n - 1 );
}
```

```
\defTaggedFileFragment<Another file fragment;recursive>+={sourceFragments.cpp}
\defTaggedCurrentFileFragment<Another file fragment;recursive>+=
\taggedFileFragment{+}{Another file fragment}{recursive}{sourceFragments.cpp}
```

The general syntax is:

```
\defTaggedFileFragment<FRAGMENT;TAG>PREFIX={FILENAME}
\defTaggedCurrentFileFragment<FRAGMENT;TAG>PREFIX=
\taggedFileFragment{PREFIX}{FRAGMENT}{TAG}{FILENAME}
```