SGV SARC

Crystal C/C++ Tools

Crystal C/C++ Tools is a revolutionary suite of review, visualization and editing tools.

Edit Tools

- **Automatic Formatting**

  Formats the code as you edit – in real-time.

  You save time and effort …

  and you get easy-to-read code.

- **Tokens Panel**

  Edit word-by-word instead of character-by-character.

- **Comments Panel**

  Add comments to your code with just a few clicks.

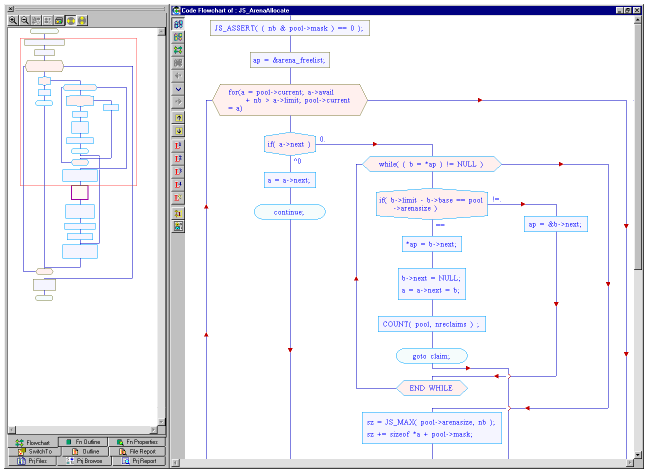
Use Crystal C for most of your editing and reviewing.

Continue to use your IDE for compiling, debugging and minor editing.

Review and Visualization Tools

- **Interactive Flowcharts**

  Easy to understand and easy to remember.



- **Overview of Files**

  Familiarize yourself with a set of source files.

- **Software Metrics**

  Volume metrics, Halstead's, McCabe's metrics.

When the Code is Not Formatted Properly - You Lose Time

File Edit View Create Insert Selected text Goto Project Tools Window Help

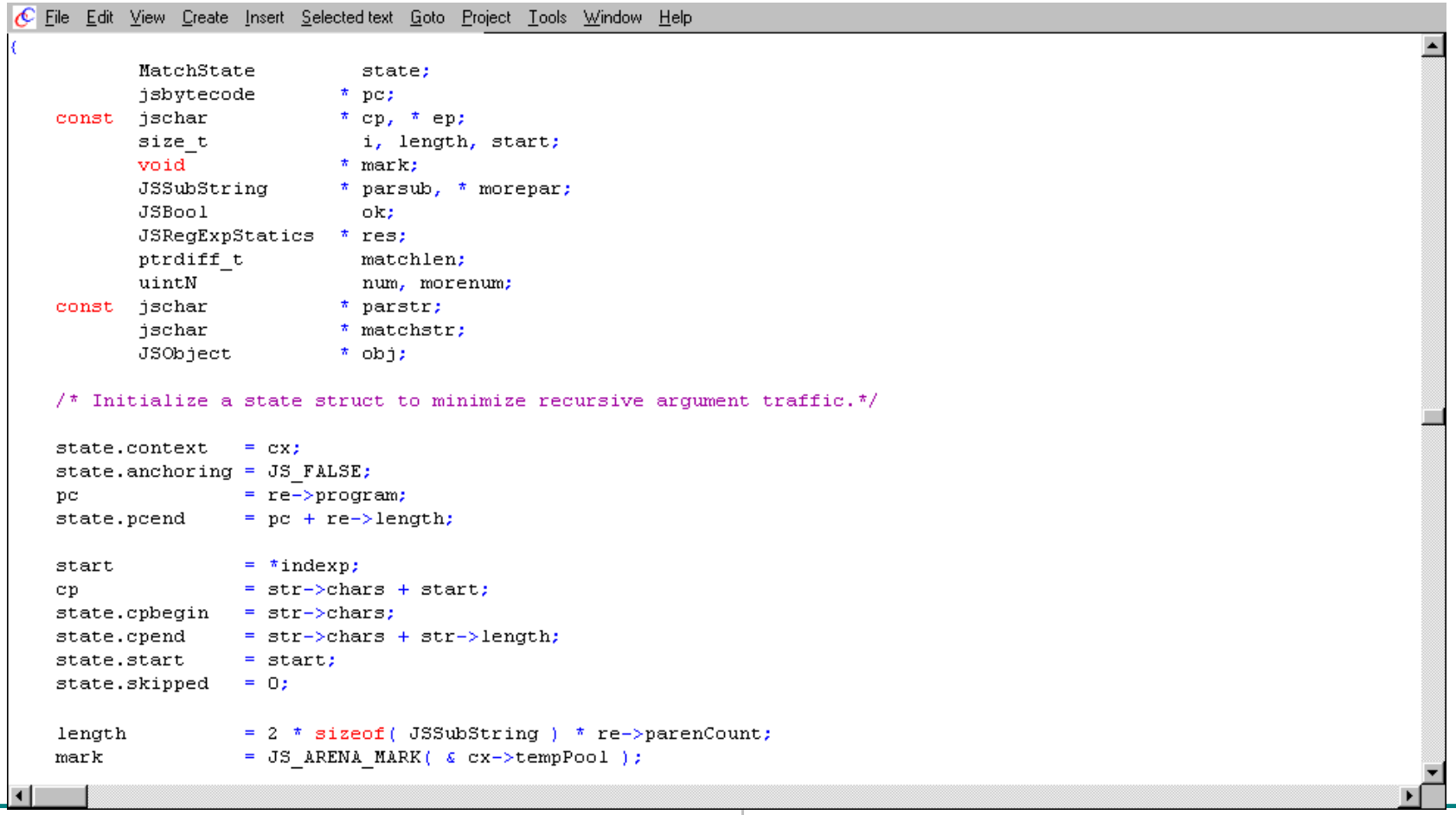
```
{
    MatchState state;
    jsbytecode *pc;
    const jschar *cp, *ep;
    size_t i, length, start;
    void *mark;
    JSSubString *parsub, *morepar;
    JSBool ok;
    JSRegExpStatics *res;
    ptrdiff_t matchlen;
    uintN num, morenum;
    const jschar *parstr;
    jschar *matchstr;
    JSObject *obj;

    /* Initialize a state struct to minimize recursive argument traffic.*/

    state.context = cx;
    state.anchoring = JS_FALSE;
    pc = re->program;
    state.pcend = pc + re->length;

    start = *indexp;
    cp = str->chars + start;
    state.cpbegin = str->chars;
    state.cpend = str->chars + str->length;
    state.start = start;
    state.skipped = 0;

    length = 2 * sizeof(JSSubString) * re->parenCount;
    mark = JS_ARENA_MARK(&cx->tempPool);
```

Your effort is wasted in

visually separating each declaration and statement.

- It makes the code hard to understand.
- You may not notice errors in the code.
- It wastes everyone's time during the code's life.

With Automatic Formatting, the Code is Easy to Read



1. Crystal C formats the code as you edit.

   You save the time and effort of manual formatting.

2. You can easily understand the code above.

   You can focus on the program logic.

   Good formatting saves time.

Use the Tokens-Panel to Edit Code

- Edit in an easygoing manner:

  no spelling errors; less fatigue at the end of an edit session.

  1. Variables, functions appear in the names' area – as per their likelihood.
  2. When an operator is more likely,
     the full set of operators appears in place of names.
  3. When you do not see the name you need,
     click one or two leading chars or click a prefix.

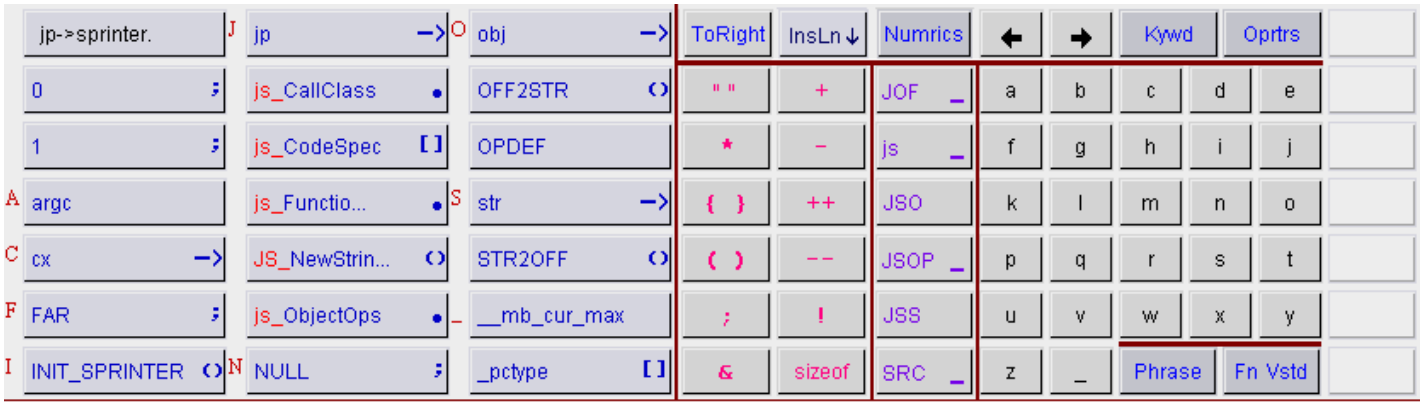
- Automatic formatting takes care of the rest.

  You need not waste your time in spacing, indenting etc.

You will create better code in less time.

Edit Word-by-Word - not Character-by-Character

Context Sensitive Tokens-Panel



Consider:

```
cx   = jp->sprinter.context;
size = jp->sprinter.size;
```

- You can enter the statement:

```
cx   = jp->sprinter.context;
```

by simply clicking

cx, =, jp ->, sprinter., context, ;

- After you have entered

```
cx   = jp->sprinter.context;
size =
```

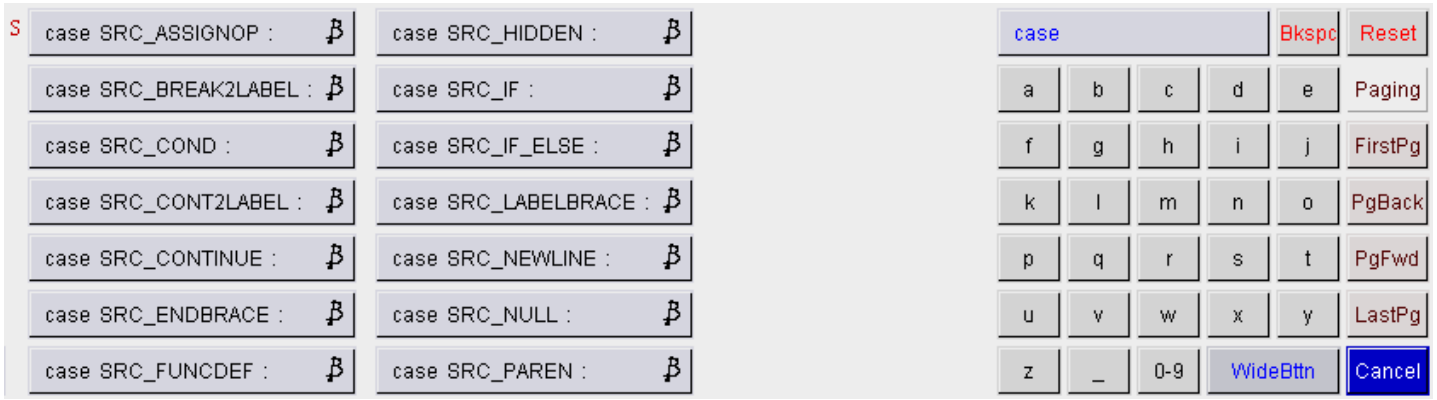
Here you may need jp->sprinter. again.

Crystal C displays jp->sprinter. in the Tokens-Panel.

- Enter jp->sprinter. with one click.

Enter Whole Phrases with a Click

Entering case's in a switch



- Say you are entering the cases in a switch:

```
switch( type )  {
   case SRC_LABEL:
   case SRC_LABELBRACE:
   case SRC_BREAK2LABEL:
   case SRC_CONT2LABEL:
   case SRC_FUNCDEF:
      atomIndex = js_GetSrcNoteOffset( sn, 0 );
      break;
```

- After you have entered

```
switch( type )  {
   case SRC_LABEL:
```

Crystal C displays a set of cases as above.

- Click the buttons to enter the cases you need.

Use the Comment-Phrases Panel to Add Comments

- Crystal C provides a simple tool for commenting.

  You can add comments to:

  - the code you have just implemented or
  - or the legacy code that you are deciphering.

- By reusing the objects' meanings,

  Crystal C helps you compose the comments.

- When you comment, you are likely to uncover errors in the code.

- The little amount of time that you invest in adding the comments

  yields huge time-savings during the life of the code.

Why struggle with code without comments?

Without Comments, You will Take Longer to Understand the Code

File Edit View Create Insert Selected text Goto Project Tools Window Help

```
    ap = &arena_freelist;

    for(ar_p = pool->current; ar_p->avail + nb > ar_p->limit;
        pool->current = ar_p)
    {
        if(ar_p->next)                      /* move to next arena */
        {
            ar_p = ar_p->next;
            continue;
        }
        while( (free_p = *ap) != NULL)      /* reclaim a free arena */
        {
            if(free_p->limit - free_p->base == pool->arenasize)
            {
                *ap = free_p->next;
                free_p->next = NULL;
                ar_p = ar_p->next = free_p;
                goto claim;
            }
            ap = &free_p->next;
        }

        sz = JS_MAX(pool->arenasize, nb);
        sz += sizeof *ar_p + pool->mask;  /* header and alignment slop */
        free_p = malloc( sz );
        if( !free_p )
            return 0;
        ar_p= ar_p->next = free_p;
        ar_p->next = NULL;
        ar_p->limit = (jsuword)ar_p + sz;
claim:
        ar_p->base = ar_p->avail = JS_ARENA_ALIGN(pool, ar_p + 1);
    }
    p = (void *)ar_p->avail;
    ar_p->avail += nb;
    return p;
}
```

How quickly can you understand the code above -

so that you can modify it or fix it?

With Comments, You Zip through the Code

File Edit View Create Insert Selected text Goto Project Tools Window Help

```
    ap = &arena_freelist;                                   /*  Get ptr_to list of free Arenas                          */

    for(ar_p = pool->current;  ar_p->avail + nb > ar_p->limit;  /*  start at current Arena in pool ; (while the current     */
        pool->current = ar_p)                               /*      Arena does not have required space)                 */
    {
        if( ar_p->next )                                    /*    if ( current Arena is not the last in the chain )     */
        {
            ar_p = ar_p->next;                              /*      advance to next arena                               */
            continue;                                       /*      continue ;                                          */
        }
        while( ( free_p = *ap ) != NULL )                   /*    while ( a free Arena is available )                   */
        {
            if( free_p->limit - free_p->base == pool->arenasize )  /*      if ( space in free Arena == size of new arena )   */
            {
                *ap          = free_p->next;                /*        list_of free Arenas = advance to next             */
                free_p->next = NULL;                        /*        grab this free Arena                              */
                ar_p         = ar_p->next = free_p;         /*        link it to current Arena ; advance to it          */
                goto  claim;                                /*        goto claim                                        */
            }
            ap = &free_p->next;                             /*      advance to next free Arena                          */
        }                                                   /*    end_while                                             */

                                                            /*A free arena with enough free space was not found       */
        sz      = JS_MAX( pool->arenasize, nb );            /*    max of ( net size of new arena , number of bytes )  */
        sz     += sizeof *ar_p + pool->mask;                /*    Add header and alignment slop                       */
        free_p  = malloc( sz );                             /*    new Arena = malloc ( size )                         */
        if( !free_p )
            return 0;
        ar_p        = ar_p->next = free_p;                  /*    link new Arena to current Arena ; advance to it     */
        ar_p->next = NULL;                                  /*    link to next arena = NULL                           */
        ar_p->limit = ( jsuword ) ar_p + sz;                /*    Set limit in new Arena                              */
claim:
        ar_p->base = ar_p->avail = JS_ARENA_ALIGN( pool, ar_p + 1 ); /*    Align as per mask, Set next available , Set base  */
    }
    p           = ( void * ) ar_p->avail;                   /*  ptr_to required space = ptr_to next available byte     */
    ar_p->avail += nb;                                      /*  advance ptr_to next available byte                     */
    return p;                                               /*  return ptr_to required space                           */
}
```

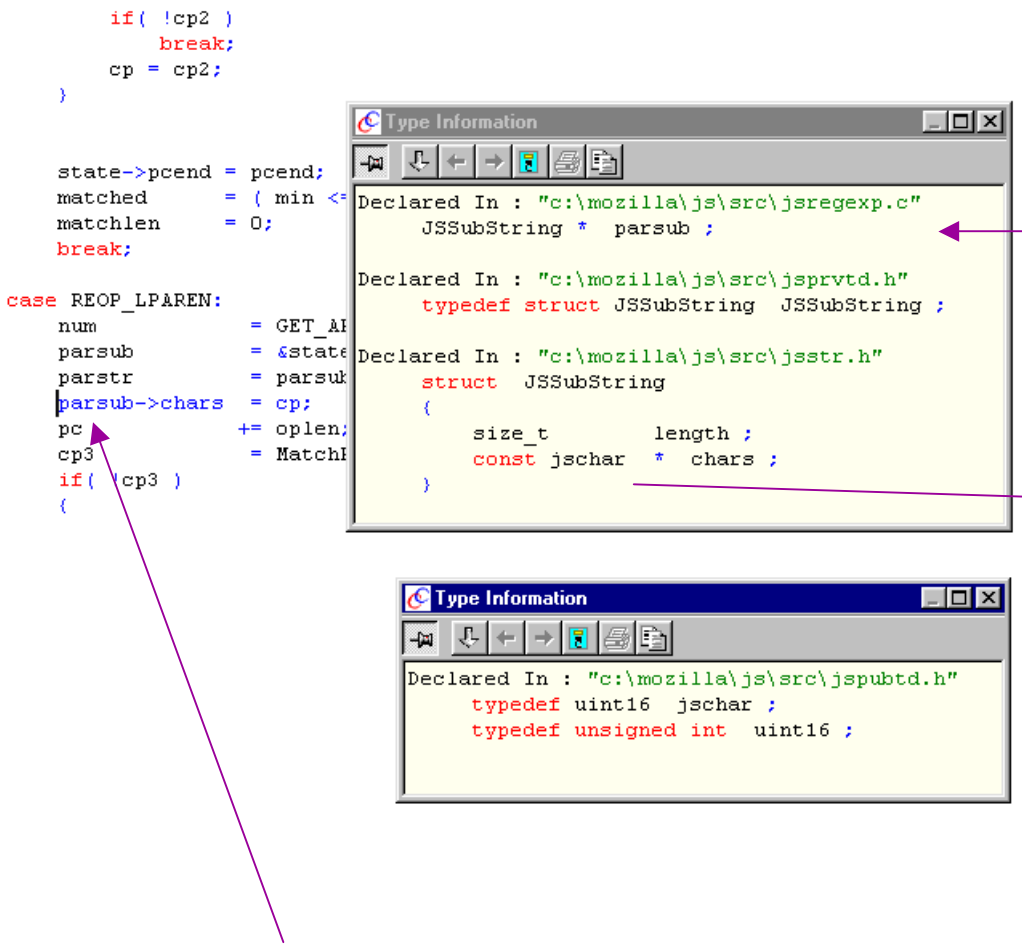
The comments above help you understand the code fully and clearly.

- Using the context-sensitive Comments-Panel, you can enter the comments above in 5 minutes.

Truly a Software Engineer's Editor

- Go to any recently visited function with just a click.
- Visit the source lines that you have modified in recent edit sessions.
- You may search/replace variables, structure-members etc.
  as per scope rules.
- View the full sequence of a type's definition without going to header files.
- Syntax errors are highlighted immediately as you edit.
  It cuts down the number of compile/edit iterations.
- View undeclared objects, unmatched delimiters, unused objects.

View Full Type-Information



The cursor is on parsub

It shows the sequence of

how and where parsub is declared

- when you double-click on jschar,

  the next window shows

  how and where jschar is declared.

- if  is selected,

  a new window is opened.

You do not have to go to the header files.

This afternoon, say you are planning to edit code with Crystal C:

1. Use the Tokens-Panel to edit code.
   You will save time. You can focus on the program logic.

2. Automatic formatting frees you from manually formatting the code.

3. Add comments to the code with the Comments Panel.
   - When you are adding comments, you are likely to catch errors.
     It will reduce test and debug time.

4. Also,
   - Highlight all modified lines to help you review code changes
   - Syntax-error highlighting in real-time
   - With flowcharts, take another look … check if all logical conditions are covered.

How does your present editor stack up?

Character-by-character editing, manual formatting, no support for commenting, no flowcharts –

Is it costing you an extra hour every day ?

SGV Sarc 907 Broad Oaks Drive, Herndon, Virginia 20170
Phone: 703-904-0678 Fax: 703-904-0155 www.sgvsarc.com