How can a function with 'varargs' retrieve the contents of the stack?

Asked 16 years, 1 month ago Modified 16 years, 1 month ago Viewed 5k times

20

Normally, in Delphi one would declare a function with a variable number of arguments using the 'array of const' method. However, for compatibility with code written in C, there's an much-unknown 'varargs' directive that can be added to a function declaration (I learned this while reading Rudy's excellent 'Pitfalls of convering' document).

As an example, one could have a function in C, declared like this :

```
void printf(const char *fmt, ...)
```

In Delphi, this would become :

```
procedure printf(const fmt: PChar); varargs;
```

My question is : How can I get to the contents of the stack when implementing a method which is defined with the 'varargs' directive?

I would expect that some tooling for this exists, like Dephi translations of the va_start(), va_arg() and va_end() functions, but I can't find this anywhere.

Please help!

PS: Please don't drift off in discussions about the 'why' or the 'array of const' alternative - I need this to write C-like patches for functions inside Xbox games (see the Delphi Xbox emulator project 'Dxbx' on sourceforge for details).

delphi variadic-functions

edited Nov 18, 2008 at 13:02

asked Nov 18, 2008 at 10:26

PatrickvL
4,164 ● 3 ● 31 ● 46

3 Answers

Sorted by: Highest score (default)

22

OK, I see the clarification in your question to mean that you need to implement a C import in Delphi. In that case, you need to implement varargs yourself.

The basic knowledge needed is the C calling convention on the x86: the stack grows downwards, and C pushes arguments from right to left. Thus, a pointer to the last declared argument, after it is incremented by the size of the last declared argument, will point to the tail argument list. From then, it's simply a matter of reading the argument out and incrementing the pointer by an appropriate size to move deeper into the stack. The x86 stack in 32-bit mode is 4-byte aligned generally, and this also means that bytes and words are passed as 32-bit integers.

Anyhow, here's a helper record in a demo program that shows how to read out data. Note that Delphi seems to be passing Extended types in a very odd way; however, you likely won't have to worry about that, as 10-byte floats aren't generally widely used in C, and aren't even implemented in the latest MS C, IIRC.

```
{$apptype console}

type
  TArgPtr = record
  private
    FArgPtr: PByte;
    class function Align(Ptr: Pointer; Align: Integer): Pointer; static;
  public
    constructor Create(LastArg: Pointer; Size: Integer);
    // Read bytes, signed words etc. using Int32
    // Make an unsigned version if necessary.
    function ReadInt32: Integer;
    // Exact floating-point semantics depend on C compiler.
    // Delphi compiler passes Extended as 10-byte float; most C
    // compilers pass all floating-point values as 8-byte floats.
    function ReadDouble: Double;
    function ReadExtended: Extended;
    function ReadPChar: PChar;
    procedure ReadArg(var Arg; Size: Integer);
  end;

constructor TArgPtr.Create(LastArg: Pointer; Size: Integer);
begin
  FArgPtr := LastArg;
  // 32-bit x86 stack is generally 4-byte aligned
  FArgPtr := Align(FArgPtr + Size, 4);
end;

class function TArgPtr.Align(Ptr: Pointer; Align: Integer): Pointer;
begin
  Integer(Result) := (Integer(Ptr) + Align - 1) and not (Align - 1);
end;

function TArgPtr.ReadInt32: Integer;
begin
```

```
  ReadArg(Result, SizeOf(Integer));
end;

function TArgPtr.ReadDouble: Double;
begin
  ReadArg(Result, SizeOf(Double));
end;

function TArgPtr.ReadExtended: Extended;
begin
  ReadArg(Result, SizeOf(Extended));
end;

function TArgPtr.ReadPChar: PChar;
begin
  ReadArg(Result, SizeOf(PChar));
end;

procedure TArgPtr.ReadArg(var Arg; Size: Integer);
begin
  Move(FArgPtr^, Arg, Size);
  FArgPtr := Align(FArgPtr + Size, 4);
end;

procedure Dump(const types: string); cdecl;
var
  ap: TArgPtr;
  cp: PChar;
begin
  cp := PChar(types);
  ap := TArgPtr.Create(@types, SizeOf(string));
  while True do
  begin
    case cp^ of
      #0:
      begin
        Writeln;
        Exit;
      end;

      'i': Write(ap.ReadInt32, ' ');
      'd': Write(ap.ReadDouble, ' ');
      'e': Write(ap.ReadExtended, ' ');
      's': Write(ap.ReadPChar, ' ');
    else
      Writeln('Unknown format');
      Exit;
    end;
    Inc(cp);
  end;
end;

type
  PDump = procedure(const types: string) cdecl varargs;
var
  MyDump: PDump;

function AsDouble(e: Extended): Double;
begin
  Result := e;
end;
```

```
function AsSingle(e: Extended): Single;
begin
  Result := e;
end;

procedure Go;
begin
  MyDump := @Dump;

  MyDump('iii', 10, 20, 30);
  MyDump('sss', 'foo', 'bar', 'baz');

  // Looks like Delphi passes Extended in byte-aligned
  // stack offset, very strange; thus this doesn't work.
  MyDump('e', 2.0);
  // These two are more reliable.
  MyDump('d', AsDouble(2));
  // Singles passed as 8-byte floats.
  MyDump('d', AsSingle(2));
end;

begin
  Go;
end.
```

Share

Improve this answer

Follow

edited Nov 19, 2008 at 15:47

answered Nov 18, 2008 at 12:32

Barry Kelly

42.1k ● 5 ● 118 ● 193

1 This looks great! I was surprised to see there's indeed no need to use assembly for getting to the ESP register contents. Thanks for this - great example too! – PatrickvL Nov 21, 2008 at 7:58

1 Note that the code needs adaptation if it is to work on x64 - the Align function in particular truncates pointers to 32-bit values. – Barry Kelly May 20, 2014 at 12:37

Tried to pass string arguments as VAR by setting VAR in the cdecl function and the procedural type declaration. In Delphi 7 the VarArgs call only passes the 1st Arg as VAR. The rest are passed as CONST. – Guy Gordon Jul 29, 2019 at 22:13

2

I found this (from a guy we know :))

> To write this stuff properly you'll need to use BASM, Delphi's built in assembler, and code the call sequence in asm. Hopefully you've got a good idea of what you need to do. Perhaps a post in the .basm group will help if you get stuck.

Share

Improve this answer

edited May 23, 2017 at 12:00

answered Nov 18, 2008 at 10:56

Follow

Community Bot
1 ● 1

Uli Gerhardt
14k ● 1 ● 47 ● 84

0

Delphi doesn't let you implement a varargs routine. It only works for importing external cdecl functions that use this.

Since varargs is based on the cdecl calling convention, you basically need to reimplement it yourself in Delphi, using assembly and/or various kinds of pointer manipulation.

Share

Improve this answer

Follow

edited Nov 18, 2008 at 21:04

answered Nov 18, 2008 at 10:59

Michael Madsen
55k ● 8 ● 74 ● 83

No, the list of arguments is only zero-terminated if the caller passes zero as the last argument. The page you cite says so. The printf function doesn't need to have a zero to terminate the list because it can figure out how many arguments there are based on the format string.
– Rob Kennedy Nov 18, 2008 at 15:14