Subject: Time for little quiz!
Posted by mirek on Tue, 04 Apr 2006 12:02:34 GMT
View Forum Message <> Reply to Message

What is this construct supposed to do (and why):

```
void CreatePalette(const RGBA *s, int count, RGBA *palette, int ncolors)
{
 delete new sPalMaker(s, count, palette, ncolors);
}
```

Mirek

Subject: Re: Time for little quiz!
Posted by unodgs on Tue, 04 Apr 2006 17:12:28 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 04 April 2006 08:02What is this construct supposed to do (and why):

```
void CreatePalette(const RGBA *s, int count, RGBA *palette, int ncolors)
{
 delete new sPalMaker(s, count, palette, ncolors);
}
```

Mirek

It look very interesting.. but I have no idea what this construction is supposed to do..

I would use delete new if I'd like to create object on heap (but what for?) and to end object live as soon as possible.

The next reason that came to my mind is you don't have to name the object..

But why delete new sPalMaker.. is better than simply
{

```
  sPalMaker pm(s, count, palette, ncolors);
} ?
```

PS: Maybe this is stupid, but will compiler ignore creating the object if it is not created on heap because the pm isn't used further?

Subject: Re: Time for little quiz!
Posted by fudadmin on Tue, 04 Apr 2006 18:04:14 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 04 April 2006 13:02What is this construct supposed to do (and why):

```
void CreatePalette(const RGBA *s, int count, RGBA *palette, int ncolors)
{
 delete new sPalMaker(s, count, palette, ncolors);
}
```

Mirek

I'm lazy to think and test but my guess are some keywords:
some kind of clever pointers and/or references, auto counting ... memory management, deletion on creation when something related is not deleted but made free for shared use...

Subject: Re: Time for little quiz!
Posted by mirek on Tue, 04 Apr 2006 18:36:36 GMT
View Forum Message <> Reply to Message

Well, sorry for teasing, it is just an interesting solution for simple problem:

sPalMaker is "procedural object", in fact, it is routine wrapped in the class instance. All relevant work is done in constructor.

Now the trouble is that sizeof(sPalMaker) > 65KB, something I do not like to put to the stack (otherwise solution you propose would be OK as well). So "delete new" does exactly what I want - offloads memory requirements to the heap.

Mirek

Subject: Re: Time for little quiz!
Posted by victorb on Wed, 05 Apr 2006 08:21:44 GMT
View Forum Message <> Reply to Message

Mirek,

Could'nt you change this for a static function and allocate on the heap inside the static function? That might be easier to understand?

Victor

Subject: Re: Time for little quiz!
Posted by mirek on Wed, 05 Apr 2006 08:53:10 GMT
View Forum Message <> Reply to Message

That would be much more code to write. In fact, there is a couple of methods in class, and class intance here is a sort of replacement of Pascal's embedded function/procedures.

BTW, as this is quite nice implementation of median cut palette algorithm, and I was trying quite hard to google some code for this, maybe placing it here will make further implementors search easier:

Median cut palette algorithm in C++:

```
struct sPalMaker {
 int       histogram[RASTER_MAP_R][RASTER_MAP_G][RASTER_MAP_B];
 int       colorcount;
 struct Dim {
  int l, h;

  operator int() { return h - l; }
 };
 enum { G = 0, R = 1, B = 2 };
 void Copy(Dim *d, Dim *s)   { d[R] = s[R]; d[G] = s[G]; d[B] = s[B]; }
 struct Box {
  int     volume;
  int     colorcount;
  int     population;
  Dim     dim[3];
  int     avg_r, avg_g, avg_b;

  Dim&    operator[](int i)    { return dim[i]; }
```

```
 };

 void Update(Box& box, int ii);

 sPalMaker(const RGBA *s, int count, RGBA *palette, int ncolors);
};

void sPalMaker::Update(Box& x, int ii)
{
 x.colorcount = 0;
 x.population = 0;
 int a[3][RASTER_MAP_MAX];
 ZeroArray(a[R]);
 ZeroArray(a[G]);
 ZeroArray(a[B]);
 x.avg_r = x.avg_g = x.avg_b = 0;
 for(int r = x[R].l; r < x[R].h; r++)
  for(int g = x[G].l; g < x[G].h; g++)
   for(int b = x[B].l; b < x[B].h; b++) {
    int q = histogram[r][g][b];
    a[R][r] += q;
    a[G][g] += q;
    a[B][b] += q;
    x.avg_r += q * r;
    x.avg_g += q * g;
    x.avg_b += q * b;
    x.colorcount += (-q >> 31) & 1;
    x.population += q;
   }
 for(int i = 0; i < 3; i++) {
  Dim& d = x[i];
  while(d.l < d.h && a[i][d.l] == 0)
   d.l++;
  while(d.h > d.l && a[i][d.h - 1] == 0)
   d.h--;
 }
 x.volume = x[R] * x[G] * x[B];
}

static byte sRc(int avg, int pop, int div)
{
 return Saturate255(255 * (avg + (pop >> 1)) / pop / (div - 1));
}

sPalMaker::sPalMaker(const RGBA *s, int count, RGBA *palette, int ncolors)
{
 ASSERT(ncolors <= 256);
 ZeroArray(histogram);
```

```
 for(const RGBA *e = s + count; s < e; s++)
  histogram[s->r >> RASTER_SHIFT_R][s->g >> RASTER_SHIFT_G][s->b >>
RASTER_SHIFT_B]++;
 Buffer<Box> box(256);
 box[0][R].l = 0;
 box[0][R].h = RASTER_MAP_R;
 box[0][G].l = 0;
 box[0][G].h = RASTER_MAP_G;
 box[0][B].l = 0;
 box[0][B].h = RASTER_MAP_B;
 Update(box[0], 0);
 if(box[0].population == 0)
  return;
 colorcount = box[0].colorcount;
 count = 1;
 int method = 1;
 while(count < ncolors) {
  int ii = -1;
  int maxv = 0;
  if(2 * count > ncolors)
   method = 1;
  for(int i = 0; i < count; i++) {
   int v = method ? box[i].volume : box[i].colorcount;
   if(box[i].colorcount > 1 && v > maxv) {
    ii = i;
    maxv = v;
   }
  }
  if(ii < 0)
   break;
  Box& b = box[ii];
  int ci = b[R] > b[G] ? b[B] > b[R] ? B : R : b[B] > b[G] ? B : G;
  if(b[ci] == 1) {
   if(method == 1)
    break;
   method = 1;
  }
  else {
   int m = (b[ci].l + b[ci].h) >> 1;
   Box& b1 = box[count];
   b1 = b;
   b[ci].h = m;
   b1[ci].l = m;
   Update(b, ii);
   Update(b1, count++);
  }
 }
 for(int i = 0; i < count; i++) {
```

```
  RGBA& c = palette[i];
  Box& x = box[i];
  c.r = sRc(x.avg_r, x.population, RASTER_MAP_R);
  c.g = sRc(x.avg_g, x.population, RASTER_MAP_G);
  c.b = sRc(x.avg_b, x.population, RASTER_MAP_B);
  c.a = 255;
 }
}

void CreatePalette(const RGBA *s, int count, RGBA *palette, int ncolors)
{
 ASSERT(ncolors <= 256);
 delete new sPalMaker(s, count, palette, ncolors);
}
```

Subject: Re: Time for little quiz!
Posted by mirek on Wed, 05 Apr 2006 08:54:39 GMT
View Forum Message <> Reply to Message

PS, note this nice gem as well:

```
x.colorcount += (-q >> 31) & 1;
```

Mirek

Subject: Re: Time for little quiz!
Posted by gprentice on Wed, 05 Apr 2006 09:20:55 GMT
View Forum Message <> Reply to Message

luzr wrote on Wed, 05 April 2006 20:54PS, note this nice gem as well:

```
x.colorcount += (-q >> 31) & 1;
```

Mirek

Do you feel like explaining?

I'm sure you know that right shift of a signed negative value has an implementation defined result and that int isn't always 32 bits and ... why would you write obscure code like this ???

Graeme

Subject: Re: Time for little quiz!
Posted by mirek on Wed, 05 Apr 2006 10:04:57 GMT
View Forum Message <> Reply to Message

gprentice wrote on Wed, 05 April 2006 05:20luzr wrote on Wed, 05 April 2006 20:54PS, note this nice gem as well:

x.colorcount += (-q >> 31) & 1;

Mirek

Do you feel like explaining?

I'm sure you know that right shift of a signed negative value has an implementation defined result and that int isn't always 32 bits and ... why would you write obscure code like this ???

Graeme

To avoid conditional jump in inner loop:

I need

if(q) colorcount++;

Now I know that q is positive number or zero. By negating positive number, you obviously get 1 in the highest bit. By negating zero, you get zero...

BTW, U++ guarantees int to be at least 32 bits (in other words, it requires it as well Also, right shift is implementation defined, however it guarantees that it is a shift... (I mean, what is implementation defined is content of high bits, but obviously, existing bits have to be shifted...)

On the similar theme:

inline byte Saturate255(int x) { return byte(~(x >> 24) & (x | (-(x >> 8) >> 24)) & 0xff); }

jump-less equivalent of min(max(x, 0), 255) (important in image processing).

Mirek

Subject: Re: Time for little quiz!
Posted by gprentice on Wed, 05 Apr 2006 10:23:34 GMT
View Forum Message <> Reply to Message

ok, well sorry to be dumb but ... why do you want to avoid the conditional jump. Why do you think x.colorcount += (-q >> 31) & 1; is going to be more efficient than if(q) colorcount++; (or x.colorcount += bool(q);)

The compiler could be smart enough to generate a test and jump instead of shifting 31 times anyway or doing something else ??

BTW - I was thinking of 64 bit machines (regarding 32 bit ints) - but compilers for 64 bit seem to have standardized on 32 bits for int still?

Graeme

Subject: Re: Time for little quiz!
Posted by gprentice on Wed, 05 Apr 2006 10:31:56 GMT
View Forum Message <> Reply to Message

unodgs wrote on Wed, 05 April 2006 05:12luzr wrote on Tue, 04 April 2006 08:02What is this construct supposed to do (and why):

```
void CreatePalette(const RGBA *s, int count, RGBA *palette, int ncolors)
{
 delete new sPalMaker(s, count, palette, ncolors);
}
```

Mirek

It look very interesting.. but I have no idea what this construction is supposed to do..

I would use delete new if I'd like to create object on heap (but what for?) and to end object live as soon as possible.

The next reason that came to my mind is you don't have to name the object..

But why delete new sPalMaker.. is better than simply
```
{
  sPalMaker pm(s, count, palette, ncolors);
}  ?
```

PS: Maybe this is stupid, but will compiler ignore creating the object if it is not created on heap because the pm isn't used further?

The compiler can ignore creating the pm object if it can determine that doing so makes no difference to the output (observable behaviour) of the program. There are situations where the compiler can elide (means ignore/not do) a call to a copy constructor even if the copy constructor has side effects (modifies a global variable or something) - the famous return value optimisation (RVO) being one - but apart from these, the compiler has to do what you tell it if it makes a difference to the output of the program.

Graeme

Subject: Re: Time for little quiz!
Posted by mirek on Wed, 05 Apr 2006 11:14:01 GMT
View Forum Message <> Reply to Message

gprentice wrote on Wed, 05 April 2006 06:23ok, well sorry to be dumb but ... why do you want to avoid the conditional jump. Why do you think x.colorcount += (-q >> 31) & 1; is going to be more efficient than if(q) colorcount++; (or x.colorcount += bool(q);)

Most likely on any CPU since Pentium I: Failed branch prediction causes pipline stall - that is at least 12 CPU cycles wasted (31 on Prescott). Also, jumps cannot be fed to more executional units - while arithmetic ops can be. It is very likely that modern CPU will be doing those colorcount ops in the same time as operations around.

BTW, shifting 31 times is 1 cycle on AMD and PIII/Core/Conroe CPUs (unfortunately, it is more pricey for Netburst, however failed branch prediction is still even more pricey there).

Quote:
BTW - I was thinking of 64 bit machines (regarding 32 bit ints) - but compilers for 64 bit seem to have standardized on 32 bits for int still?

Yes, all major 64 bit CPUs today have 32bit ints. Well, to put it more straight, standard for all C compilers is to use 32 bit ints, other than that they support 8/16/32/64 integer quantities.

The reason is obvious - 32 bits is enough in most cases where integer quantity is to be used and 64 bit ints would eat much more space in data cache. BTW, AMD64 opcodes for 64 bit int operarations are 1 byte longer (they have "use 64 bits" prefix).

All that said, above jump-less technique would work with 64bit int as well.

Mirek

Subject: Re: Time for little quiz!
Posted by hojtsy on Wed, 05 Apr 2006 12:26:40 GMT
View Forum Message <> Reply to Message

luzr wrote on Wed, 05 April 2006 06:04
Now I know that q is positive number or zero. By negating positive number, you obviously get 1 in the highest bit.I am afraid there is no guarantee provided by the C++ standard for the bit encoding of negative numbers. Depending on any specific encoding could be either implementation-defined or undefined behaviour. I suggest some other variants which have standard-defined behaviour.

```
if(q) colorcount++;           // supposed to be slow(?)
colorcount += (-q >> 31) & 1;  // luzr's solution
coloccount += bool(q);         // suggested solution 1
coloccount += (q != 0);        // suggested solution 2
coloccount += !!q;             // suggested solution 3
```

Subject: Re: Time for little quiz!
Posted by victorb on Wed, 05 Apr 2006 12:52:14 GMT
View Forum Message <> Reply to Message

Interesting to see that none of the 3 suggested solution generate a conditional jump... they make use of the "setne" opcode...
I don't have the cycle count for each instruction but it might be as optimal as "(-q >> 31) & 1" with the benefit of being understandable.

-> I vote for coloccount += (q != 0);

Victor

Subject: Re: Time for little quiz!
Posted by mirek on Wed, 05 Apr 2006 12:53:45 GMT
View Forum Message <> Reply to Message

In theory, you are right, Howver, it depends on what you praise more: Speed or possible protability to non-existent platform?

There are more cases like this when you have to narrow your focus.

Should we support machines with int less than 32 bits?

Should we support machines that follow neither big-endian nor little-endian mode?

Personally, it is OK with me if there is nothing in U++ that would limit it running on x86, PowerPC, ARM and Sparc. If there are implementation defined issues that differ in above architectures, we should handle them. If there is something that does not differ (like binary representation of integers), I simply do not care!

Mirek

Subject: Re: Time for little quiz!
Posted by mirek on Wed, 05 Apr 2006 12:56:19 GMT
View Forum Message <> Reply to Message

victorb wrote on Wed, 05 April 2006 08:52Interesting to see that none of the 3 suggested solution generate a conditional jump... they make use of the "setne" opcode...
I don't have the cycle count for each instruction but it might be as optimal as "(-q >> 31) & 1" with the benefit of being understandable.

-> I vote for coloccount += (q != 0);

Victor

"setne" is a good point. However, it does not seem to be used with all compilers on all CPUs.... (but I guess it would deserve the conditional compilation at least).

Mirek

Subject: Re: Time for little quiz!
Posted by victorb on Wed, 05 Apr 2006 13:04:50 GMT
View Forum Message <> Reply to Message

"Setne" gets generated with GCC on a pentium M.

BTW,

I remove my vote for "colorcount += (q != 0)" adding an int with a boolean is not so nice...

And I propose suggested #4: "colorcount += (q)?1:0;" (this will generate the same code as #3 with GCC).

Funny to see how many post such a small statement "(-q >> 31) & 1" could generate

---

Subject: Re: Time for little quiz!
Posted by hojtsy on Wed, 05 Apr 2006 13:50:37 GMT
View Forum Message <> Reply to Message

victorb wrote on Wed, 05 April 2006 09:04I remove my vote for "colorcount += (q != 0)" adding an int with a boolean is not so nice...At least it has standard-defined behaviour, which is an improvement over shifting negative numbers.

victorb wrote on Wed, 05 April 2006 09:04
And I propose suggested #4: "colorcount += (q)?1:0;" (this will generate the same code as #3 with GCC). You don't really need parenthesis there: "colorcount += q ? 1 : 0;" First I thought that this variant may generate a branch.

victorb wrote on Wed, 05 April 2006 09:04Funny to see how many post such a small statement "(-q >> 31) & 1" could generate I think that the big challenge is not to write fast & obfuscated code, but rather write fast and clean code. This example is small enough to test whether this is possible in C if you take the "fast" aspect to the extreme. I see the obfuscated version as practically assembly magic, even if it is written with C statements: the question is whether there is a C++ standard-conformant solution with the same or better runtime performance.

Subject: Re: Time for little quiz!
Posted by mirek on Wed, 05 Apr 2006 15:42:33 GMT
View Forum Message <> Reply to Message

victorb wrote on Wed, 05 April 2006 09:04
I think that the big challenge is not to write fast & obfuscated code, but rather write fast and clean code. This example is small enough to test whether this is possible in C if you take the "fast" aspect to the extreme. I see the obfuscated version as practically assembly magic, even if it is written with C statements: the question is whether there is a C++ standard-conformant solution with the same or better runtime performance.

Just my stance: While "setne" solution is even better and it is my fault that I missed that possibility, I believe that for time critical routine _implementation_ it is OK to be fast at the cost of being obfuscated. And this routine can be quite limiting in tasks like webservers that generate gifs...

When optimizing, assembler magic is what you often have to use... (actually, I even plan to write some nImage routines in assembler

Mirek

Subject: Re: Time for little quiz!
Posted by gprentice on Thu, 06 Apr 2006 12:29:41 GMT
View Forum Message <> Reply to Message

gprentice wrote on Wed, 05 April 2006 21:20luzr wrote on Wed, 05 April 2006 20:54PS, note this nice gem as well:

x.colorcount += (-q >> 31) & 1;

Mirek

Do you feel like explaining?

I'm sure you know that right shift of a signed negative value has an implementation defined result and that int isn't always 32 bits and ... why would you write obscure code like this ???

Graeme

FWIW - Visual C++ and GCC describe their implementation defined behavior for integer bit operations - most likely all GCC versions do the sign extension mentioned and all VC versions treat signed as unsigned.

http://msdn2.microsoft.com/en-US/library/dxda59dh(VS.80).asp x

http://gcc.activeventure.org/Integers-implementation.html#In tegers-implementation

Graeme

Subject: Re: Time for little quiz!
Posted by mirek on Thu, 06 Apr 2006 13:56:59 GMT
View Forum Message <> Reply to Message

gprentice wrote on Thu, 06 April 2006 08:29
FWIW - Visual C++ and GCC describe their implementation defined behavior for integer bit operations - most likely all GCC versions do the sign extension mentioned and all VC versions treat signed as unsigned.

[url=http://msdn2.microsoft.com/en-US/library/dxda59dh(VS.80).aspx
http://msdn2.microsoft.com/en-US/library/dxda59dh(VS.80).asp x[/url]

http://gcc.activeventure.org/Integers-implementation.html#In tegers-implementation

Graeme

Yes, this implementation defined behaviout is in fact quite consistent across all platforms. I guess that expecting behaviour you describe would not hurt portability in any way....

Mirek

Subject: Re: Time for little quiz!
Posted by victorb on Thu, 06 Apr 2006 14:17:03 GMT
View Forum Message <> Reply to Message

Graeme,

If I understand your previous topic well, you are stating that there might be an inconsistency between GCC and VC because they do not handle integer the same way and you are afraid this might cause errors with "(-q >> 31) & 1".

However if you look at the following URL:
http://msdn2.microsoft.com/en-US/library/7x62187h(VS.80).aspx
You will see that VC handle right shifts the same way as GCC by etending the sign bit (for negative number as "-q").

VC:
"Right shifts preserve the sign bit. When a signed integer shifts right, the most-significant bit remains set. When an unsigned integer shifts right, the most-significant bit is cleared."

GCC:
"Signed >> acts on negative numbers by sign extension."

Victor

Subject: Re: Time for little quiz!
Posted by mirek on Thu, 06 Apr 2006 14:25:43 GMT
View Forum Message <> Reply to Message

victorb wrote on Thu, 06 April 2006 10:17Graeme,

If I understand your previous topic well, you are stating that there might be an inconsistency between GCC and VC because they do not handle integer the same way and you are afraid this might cause errors with "(-q >> 31) & 1".

However if you look at the following URL:
http://msdn2.microsoft.com/en-US/library/7x62187h(VS.80).aspx
You will see that VC handle right shifts the same way as GCC by etending the sign bit (for negative number as "-q").

VC:
"Right shifts preserve the sign bit. When a signed integer shifts right, the most-significant bit remains set. When an unsigned integer shifts right, the most-significant bit is cleared."

GCC:
"Signed >> acts on negative numbers by sign extension."

Victor

Let me just note that for "(x >> 31) & 1" expression, as long as int is 32 bits or more, it does not matter how sign extension works

Mirek

Subject: Re: Time for little quiz!
Posted by nixnixnix on Mon, 01 Sep 2008 22:04:23 GMT
View Forum Message <> Reply to Message

ok I am not quite up to understanding the full consequences of the above conversation. However, the bit (excuse the pun) that interests me is:

how many bits does an "int" have when I compile on a 64 bit OS (linux or vista - I am trying both)?

If I want my documents to load and save (using Serialize(Stream&)) between 32 and 64 bit implementations of my software, do I need to replace all "int" with "int32"? I am guessing this is so but what else will I need to change please?

char is still 16 bit right?
float is always 32 bit right?
double is always 64 bit right?
what about bool?
what about String?

Nick

Subject: Re: Time for little quiz!
Posted by mirek on Tue, 02 Sep 2008 06:56:35 GMT
View Forum Message <> Reply to Message

[quote title=nixnixnix wrote on Mon, 01 September 2008 18:04]ok I am not quite up to understanding the full consequences of the above conversation. However, the bit (excuse the pun) that interests me is:

how many bits does an "int" have when I compile on a 64 bit OS (linux or vista - I am trying both)?

If I want my documents to load and save (using Serialize(Stream&)) between 32 and 64 bit implementations of my software, do I need to replace all "int" with "int32"? I am guessing this is so but what else will I need to change please?
[quote]

It "depends".

With x86-64, int is still 32 bits. And in fact, AFAIK, this is the most common arrangement for 64-bit systems.

To get "true" 64-bit integer, types vary more, but with U++, you can just use "int64".

Quote:
char is still 16 bit right?

char is always 8 bit.

wchar is now 16 bit, but it seems like we have to go 32-bit soon. But that should not affect the existing code too much.

Quote:
float is always 32 bit right?
double is always 64 bit right?

Yes.

Quote:
what about bool?

Frankly, I do not know But I believe it is 1 for both GCC and MSC.

Quote:
what about String?

Current implementation 16. May change (before 2007, it was equal to sizeof(void*)).

Mirek

Subject: Re: Time for little quiz!
Posted by nixnixnix on Tue, 02 Sep 2008 19:54:51 GMT
View Forum Message <> Reply to Message

Quote:
To get "true" 64-bit integer, types vary more, but with U++, you can just use "int64".

Quote:

char is still 16 bit right?

char is always 8 bit.

doh! thats what i meant to write.

So there is no reason you can think of why a document serialized by a 32 bit compilation of a program might not be able to be read by a 64 bit compilation of the same program?

Nick

Subject: Re: Time for little quiz!
Posted by mirek on Wed, 03 Sep 2008 12:18:25 GMT
View Forum Message <> Reply to Message

nixnixnix wrote on Tue, 02 September 2008 15:54
So there is no reason you can think of why a document serialized by a 32 bit compilation of a program might not be able to be read by a 64 bit compilation of the same program?

Now I am a little bit confused...

Are we speaking about Serialize(Stream&) ?

If yes, this is explicitly programmed to be platform independed. So even if "int" would be 64-bit, Serialize would store only (lower) 32.

Mirek

Subject: Re: Time for little quiz!
Posted by nixnixnix on Wed, 03 Sep 2008 14:38:16 GMT
View Forum Message <> Reply to Message

Ok, good to know. Thanks. It must be something else.

Have you tested much under 64 bit windows? I notice that all the code which distinguishes between windows and linux uses the #ifdef PLATFORM_WIN32 directive. Is this flag set when one is compiling 64 bit code under windows?

Nick

Subject: Re: Time for little quiz!
Posted by mrjt on Wed, 03 Sep 2008 14:51:34 GMT
View Forum Message <> Reply to Message

Win32 is simply the name of the API introduced with Window 95 and used in all Windows versions since then (though perhaps not for very much longer). So yes, it is.

Subject: Re: Time for little quiz!
Posted by mirek on Sun, 07 Sep 2008 08:12:35 GMT
View Forum Message <> Reply to Message

nixnixnix wrote on Wed, 03 September 2008 10:38Ok, good to know. Thanks. It must be something else.

Have you tested much under 64 bit windows? I notice that all the code which distinguishes between windows and linux uses the #ifdef PLATFORM_WIN32 directive. Is this flag set when one is compiling 64 bit code under windows?

Nick

IMO, you can say that Win32 is a subset of Win64...

Mirek