

Forum:[Hack Forums](#)

Thread: [Any Random Number Generators implementations that don't use CRT](#)

d4rkj4y - January 25, 2014, 3:18 am

I'm looking for a quick implementation of a random number generator in C++ that can generate a random number within a range, that doesn't use the CRT.

I plan to paste it into my code for the time being, and when I get round to it improve it / write my own one.

I found this : <http://www.bedaux.net/mtrand/>

But it doesn't generate within a range, correct me if I'm wrong but I'm in a rush.

Thanks

FurfUksake - January 25, 2014, 3:23 am

Have you tried one of the Mersenne Twister implementations?

<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/index.html>

<http://www.cs.hmc.edu/~geoff/mtwist.html>

Hope this helps.

Sorry I just reread your post.

With the implementation you listed it generates a float between 0 and 1. If you want it between a specific range I think you need to use the % operator and + operator like this:

Code `mtrand() % 100;`

I believe this example generates a random number between 0 and 99.

Code `mtrand() % 100 + 1;`

And I believe this creates a random number between 1 and 100.

Hope this helps.

d4rkj4y - January 25, 2014, 3:42 am

(Jan 25, 2014 04:23 AM)Veto Wrote: [\[CITE\]](#) Have you tried one of the Mersenne Twister implementations?

<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/index.html>

<http://www.cs.hmc.edu/~geoff/mtwist.html>

Hope this helps.

Hey thanks, but the files you recommended have dependencies to CRT, unfortunately.

I will just go with `CryptAcquireContext` -> `CryptGenRandom`, and format the output from there.

(Jan 25, 2014 04:23 AM)Veto Wrote: [\[CITE\]](#) Have you tried one of the Mersenne Twister

implementations?

<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/index.html>
<http://www.cs.hmc.edu/~geoff/mtwist.html>

Hope this helps.

Sorry I just reread your post.

With the implementation you listed it generates a float between 0 and 1. If you want it between a specific range I think you need to use the % operator and + operator like this:

Code `mtrand() % 100;`

I believe this example generates a random number between 0 and 99.

Code `mtrand() % 100 + 1;`

And I believe this creates a random number between 1 and 100.

Hope this helps.

Lol, I completely forgot about this, I have another implementation I should have done this for, thanks alot, can't believe my stupidity.

IRP_MJ_SCSI - January 25, 2014, 4:55 am

`RtlRandomEx()`

0xDEAD10CC - January 25, 2014, 4:50 pm

(Jan 25, 2014 05:55 AM)IRP_MJ_SCSI Wrote: [\[CITE\]](#) `RtlRandomEx()`

I was going to suggest this too. You'll have to dynamically link to this function, but I like that it generates a 64 bit integer instead of a 32 bit one. And to generate within a range, I don't see this as being a big deal, just use the % operator...

popov - January 25, 2014, 5:00 pm

OP i thought you were a "C++ FASM software engineer"?

d4rkj4y - January 25, 2014, 5:38 pm

(Jan 25, 2014 06:00 PM)popov Wrote: [\[CITE\]](#) OP i thought you were a "C++ FASM software engineer"?

I am huehuehuehue, just wanted a spoonfeed ?

(Jan 25, 2014 05:55 AM)IRP_MJ_SCSI Wrote: [\[CITE\]](#) `RtlRandomEx()`

Yah is good, thanks for recommending this.

0x1B - January 27, 2014, 11:41 am

Code `typedef ULONG(WINAPI *TypeRtlRandom)(PULONG Seed);`

`TypeRtlRandom RtlRandom;`

`LPWSTR GenerateRandomNumbers(int Length)`

{
 static wchar_t RetString[MAX_LINE];

 ULONG Seed = GetTickCount();

 HMODULE hlib = GetModuleHandleW(L"ntdll.dll");

```

RtlRandom = (TypeRtlRandom)GetProcAddress(hlib, "RtlRandom");

for(int i = NULL; i < Length; i++)
{
    _snwprintf(&RetString[i], sizeof(RetString) - 1, L"%d", RtlRandom(&Seed)%9);
}

return RetString;
}

```

```

LPWSTR GenerateRandomLetters(int Length)
{
    static wchar_t RetString[MAX_LINE];

    ULONG Seed = GetTickCount();

    HMODULE hlib = GetModuleHandleW(L"ntdll.dll");
    RtlRandom = (TypeRtlRandom)GetProcAddress(hlib, "RtlRandom");

    for(int i = NULL; i < Length; i++)
    {
        RetString[i] = RtlRandom(&Seed)%26 + 97;
    }

    return RetString;
}

```

(Jan 25, 2014 05:50 PM)TxTriton Wrote: [\[CITE\]](#) I was going to suggest this too. You'll have to dynamically link to this function, but I like that it generates a 64 bit integer instead of a 32 bit one. And to generate within a range, I don't see this as being a big deal, just use the % operator... You don't "have" to dynamically link ntdll lol

IRP_MJ_SCSI - January 27, 2014, 11:49 am

(Jan 27, 2014 12:41 PM)0x1B Wrote: [\[CITE\]](#) Code typedef ULONG(WINAPI
*TypeRtlRandom)(PULONG Seed);
TypeRtlRandom RtlRandom;

```

LPWSTR GenerateRandomNumbers(int Length)
{
    static wchar_t RetString[MAX_LINE];

    ULONG Seed = GetTickCount();

    HMODULE hlib = GetModuleHandleW(L"ntdll.dll");
    RtlRandom = (TypeRtlRandom)GetProcAddress(hlib, "RtlRandom");

    for(int i = NULL; i < Length; i++)
    {
        _snwprintf(&RetString[i], sizeof(RetString) - 1, L"%d", RtlRandom(&Seed)%9);
    }

    return RetString;
}

```

```

LPWSTR GenerateRandomLetters(int Length)
{
    static wchar_t RetString[MAX_LINE];

    ULONG Seed = GetTickCount();

    HMODULE hlib = GetModuleHandleW(L"ntdll.dll");
    RtlRandom = (TypeRtlRandom)GetProcAddress(hlib, "RtlRandom");

    for(int i = 0; i < Length; i++)
    {
        RetString[i] = RtlRandom(&Seed)%26 + 97;
    }

    return RetString;
}

```

You don't "have" to dynamically link ntdll lol

I'd probably manually encode the string though, because _snwprintf uses CRT also.

Masm4ever - January 27, 2014, 1:16 pm

(Jan 25, 2014 05:55 AM)IRP_MJ_SCSI Wrote: [\[CITE\]](#) RtlRandomEx()

It works... kind of. But its randomness is very bad. Test it with [ENT](#) or NIST, RtlRandomEx is really a lousy PRNG.

[Here](#) you have some choice.

0xDEAD10CC - January 27, 2014, 5:50 pm

0x1B Wrote:Code typedef ULONG(WINAPI *TypeRtlRandom)(PULONG Seed);
TypeRtlRandom RtlRandom;

```

LPWSTR GenerateRandomNumbers(int Length)
{
    static wchar_t RetString[MAX_LINE];

    ULONG Seed = GetTickCount();

    HMODULE hlib = GetModuleHandleW(L"ntdll.dll");
    RtlRandom = (TypeRtlRandom)GetProcAddress(hlib, "RtlRandom");

    for(int i = 0; i < Length; i++)
    {
        _snwprintf(&RetString[i], sizeof(RetString) - 1, L"%d", RtlRandom(&Seed)%9);
    }

    return RetString;
}

```

```

LPWSTR GenerateRandomLetters(int Length)
{
    static wchar_t RetString[MAX_LINE];

    ULONG Seed = GetTickCount();

```

```
HMODULE hlib = GetModuleHandleW(L"ntdll.dll");
RtlRandom = (TypeRtlRandom)GetProcAddress(hlib, "RtlRandom");
```

```
for(int i = NULL; i < Length; i++)
{
    RetString[i] = RtlRandom(&Seed)%26 + 97;
}
```

```
return RetString;
```

```
}
```

You don't "have" to dynamically link ntdll lol

You would suggest statically linking ntdll for a regular program? ? There's an underlying reason why I had those choice of words. And WHY are you using RtlRandom instead of RtlRandomEx? ? RtlRandomEx() is an improved version of the RtlRandom() function which is twice as fast and produces better random numbers also.

IRP_MJ_SCSI - January 27, 2014, 10:35 pm

(Jan 27, 2014 02:16 PM)Masm4ever Wrote: [\[CITE\]](#) It works... kind of. But its randomness is very bad. Test it with [ENT](#) or NIST, RtlRandomEx is really a lousy PRNG.

[Here](#) you have some choice.

Are you sure you're seeding it correctly?

_SLQ - January 28, 2014, 12:45 am

You can always just implement your own Markov Chain to create the illusion of random generation.

Just throwing that out there.

d4rkj4y - January 28, 2014, 1:29 am

(Jan 28, 2014 01:45 AM)_SLQ Wrote: [\[CITE\]](#) You can always just implement your own Markov Chain to create the illusion of random generation.

Just throwing that out there.

Hey dude, thanks for bringing that up, I'll look into it when I'm looking at more mathematics algebraic methods in programming

Masm4ever - January 28, 2014, 3:30 am

(Jan 27, 2014 11:35 PM)IRP_MJ_SCSI Wrote: [\[CITE\]](#) Are you sure you're seeding it correctly?

Pretty sure for my assembler version, but my C++ isn't perfect, so bear with me. I've put together a snippet here, compiles fine with VS 2010 Ex but returns only a 32-bit number, so something must be wrong:

```
Code #include <iostream>
#include "Windows.h"
```

```
typedef ULONG (__stdcall *pRex)(PULONG seed);
```

```
ULONGLONG get_random(ULONG seed)
```

```
{
    pRex RtlRandomEx = (pRex) GetProcAddress(GetModuleHandleA("ntdll.dll"),
    "RtlRandomEx");
    return RtlRandomEx(&seed)+(0xfffff+1)*seed; // trying to get a 64-bit number...
}
```

```
int main() {
    using namespace std;
    for (int i = 0; i < 10; i++)
        cout << get_random(i) << endl;
}
```

IRP_MJ_SCSI - January 28, 2014, 3:50 am

(Jan 28, 2014 04:30 AM)Masm4ever Wrote: [\[CITE\]](#) Pretty sure for my assembler version, but my C++ isn't perfect, so bear with me. I've put together a snippet here, compiles fine with VS 2010 Ex but returns only a 32-bit number, so something must be wrong:

```
Code #include <iostream>
#include "Windows.h"
```

```
typedef ULONG (__stdcall *pRex) (PULONG seed);
```

```
ULONGLONG get_random(ULONG seed)
{
    pRex RtlRandomEx = (pRex) GetProcAddress(GetModuleHandleA("ntdll.dll"),
    "RtlRandomEx");
    return RtlRandomEx(&seed)+(0xfffff+1)*seed; // trying to get a 64-bit number...
}
```

```
int main() {
    using namespace std;
    for (int i = 0; i < 10; i++)
        cout << get_random(i) << endl;
}
```

RtlRandomEx only returns a 32-bit number, to get a 64-bit one I'd recommend calling it twice then concatenating the 2 results.

Masm4ever - January 28, 2014, 7:08 am

(Jan 28, 2014 04:50 AM)IRP_MJ_SCSI Wrote: [\[CITE\]](#) RtlRandomEx only returns a 32-bit number, to get a 64-bit one I'd recommend calling it twice then concatenating the 2 results.

Hmm...

(Jan 25, 2014 05:50 PM)TxTriton Wrote: [\[CITE\]](#) I like that it generates a 64 bit integer instead of a 32 bit one

Of course, I could write now that I trust TxTriton more than MSDN ;-)

As a matter of fact, RtlRandomEx

- does return only 32 bits
- but modifies also the 32 bits of the seed

That is not documented but it works, so you can have your 64 bits. And their randomness is as lousy as the test for the 32 bits only...

IRP_MJ_SCSI - January 28, 2014, 7:27 am

(Jan 28, 2014 08:08 AM)Masm4ever Wrote: [\[CITE\]](#) Hmm...

Of course, I could write now that I trust TxTriton more than MSDN ;-)

As a matter of fact, RtlRandomEx

- does return only 32 bits
- but modifies also the 32 bits of the seed

That is not documented but it works, so you can have your 64 bits. And their randomness is as lousy as the test for the 32 bits only...

If you are looking to combine the seed and the return into a 64-bit integer, try this.

PHP Code

```
<?php
ULONGLONG get_random(ULONG seed)
{
    pRex RtlRandomEx = (pRex) GetProcAddress(GetModuleHandleA("ntdll.dll"), "RtlRandomEx");
    return (INT64)(RtlRandomEx(&seed)) + ((INT64)(seed) << 32);
}
```

0x1B - January 28, 2014, 9:36 am

(Jan 27, 2014 12:49 PM)IRP_MJ_SCSI Wrote: [\[CITE\]](#) I'd probably manually encode the string though, because _snwprintf uses CRT also.

Or you can stop being a pussy and make your own _snwprintf

IRP_MJ_SCSI - January 28, 2014, 9:49 am

(Jan 28, 2014 10:36 AM)0x1B Wrote: [\[CITE\]](#) Or you can stop being a pussy and make your own _snwprintf

I can think of better ways to spend a day than writing variadic string formatting functions.

Qmz - January 29, 2014, 1:37 pm

Was looking the same stuff today. I wish there was a site which has all CRT alternative's :P

CodeX - January 29, 2014, 2:09 pm

How random do you want? If you are only after something basic then you could look into [linear congruential generators](#), they aren't up to much but they can be better than nothing

0xDEAD10CC - January 29, 2014, 4:31 pm

The benefit of RtlRandomEx() is the range; most other random generators only produce values within the limits of the primitive int datatype.

Code int rand (void);

@CodeX - using C++11 the STL <random> header contains such methods, however, it's probably not desired if you're looking for a CRT alternative.

(Jan 28, 2014 08:27 AM)IRP_MJ_SCSI Wrote: [\[CITE\]](#) If you are looking to combine the seed and the return into a 64-bit integer, try this.

PHP Code

```
<?php
ULONGLONG get_random(ULONG seed)
{
    pRex RtlRandomEx = (pRex) GetProcAddress(GetModuleHandleA("ntdll.dll"), "RtlRandomEx");
    return (INT64)(RtlRandomEx(&seed)) + ((INT64)(seed) << 32);
}
```

@IRP_MJ_SCSI - You could use (ULONGLONG) for the end cast since that's the unsigned datatype you're returning too. Although you shouldn't even have to do that...

Code typedef ULONGLONG(__stdcall *_RtlRandomEx)(PULONG seed);

```
Code #include <iostream>
#include <Windows.h>
```

```
typedef ULONGLONG(__stdcall *_RtlRandomEx)(PULONG seed);
```

```
int main()
{
    _RtlRandomEx RtlRandomEx = (_RtlRandomEx)GetProcAddress(GetModuleHandleA("ntdll.dll"),
    "RtlRandomEx");
    ULONGLONG result = 31;
    for (int i = 0; i < 10; ++i)
    {
        ULONG seed = result ^ 7;
        result = RtlRandomEx(&seed);
        std::cout << result << std::endl;
    }
}
```

If you just change the return type, you'll get results returned as 64 bit.

```
Code 7069691053340399246
1662738671399549955
5895369796598402968
5776206001957582017
2707720144793354528
1163217151371547606
1449986147787694776
8835628898304554411
6198403967312275345
2815897181160215441
```

knox - January 29, 2014, 4:53 pm

(Jan 29, 2014 02:37 PM)Qmz_ Wrote: [\[CITE\]](#) Was looking the same stuff today. I wish there was a site which has all CRT alternative's :P

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q99456>

I keep this bookmarked :)

d4rkj4y - January 29, 2014, 5:20 pm

(Jan 29, 2014 05:53 PM)knox Wrote: [\[CITE\]](#)

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q99456>

I keep this bookmarked :)

Thank you that's very useful.

Also thanks to everyone else for contributing to this discussion, it has been interesting

Masm4ever - January 29, 2014, 7:59 pm

(Jan 29, 2014 05:31 PM)TxTriton Wrote: [\[CITE\]](#) The benefit of RtlRandomEx() is the range;It does indeed return 64 bits, in contrast to the MSDN documentation. A look under the hood reveals that both eax and edx carry information.

EDIT: Further investigation shows that it used to return 64 bits, under WinXP. Now, under Win7, edx is just a constant pointer to an internal memory location. So MSDN is correct, it returns only 32

bits. Unless you use the seed as the upper DWORD, but at that point you can also just call RtlRandomEx twice.

(Jan 29, 2014 05:31 PM)TxTriton Wrote: [\[CITE\]](#) If you just change the return type, you'll get results returned as 64 bit.

Thanks a lot, TxTriton! I have been fighting for a while with the solution proposed by IRP_MJ_SCSI, which returned only values in the 92... range, i.e. close to longlong overflow. Yours works, finally.

Legume - January 30, 2014, 12:51 am

CryptGenRandom can generate an array of random bytes of any size. Worth a look, documented properly and in advapi32.dll

IRP_MJ_SCSI - January 30, 2014, 3:10 am

(Jan 29, 2014 05:31 PM)TxTriton Wrote: [\[CITE\]](#) @IRP_MJ_SCSI - You could use (ULONGLONG) for the end cast since that's the unsigned datatype you're returning too. Although you shouldn't even have to do that...

```
Code typedef ULONGLONG(__stdcall *_RtlRandomEx)(PULONG seed);  
[code]#include <iostream>  
#include <Windows.h>
```

```
typedef ULONGLONG(__stdcall *_RtlRandomEx)(PULONG seed);
```

If you just change the return type, you'll get results returned as 64 bit. About the int64: I have a preference to int64 over ulonglong so i used that for the cast instead, i probably should have changed the return type to int64 too.

I see what you mean but when you change the return type to ULONGLONG, the return value is eax:edx, you can't really assume the edx register will always contain a random number (when i debugged it on windows 7 edx always contains the same number), it's not documented so it may work for you, but i think it's only chance that edx contains a suitable number.

(Jan 29, 2014 08:59 PM)Masm4ever Wrote: [\[CITE\]](#) Thanks a lot, TxTriton! I have been fighting for a while with the solution proposed by IRP_MJ_SCSI, which returned only values in the 92... range, i.e. close to longlong overflow. Yours works, finally.

It looks like i made an error with my solution in trying to do everything in the cast.

This should work:

PHP Code

```
<?php  
ULONGLONG get_random(ULONG seed)  
{  
    ULONGLONG Ret = 0;  
    pRex RtlRandomEx = (pRex) GetProcAddress(GetModuleHandleA("ntdll.dll"), "RtlRandomEx");  
  
    Ret |= RtlRandomEx(&seed);  
    Ret <=& 32;  
    Ret |= seed;  
    return Ret;  
}
```

0xDEAD10CC - January 30, 2014, 4:59 am

You could try something like this:

```
Code #include <iostream>
#include <Windows.h>
```

```
typedef ULONGLONG(__stdcall *_RtlRandomEx)(PULONG seed);
```

```
int main()
{
    _RtlRandomEx RtlRandomEx = (_RtlRandomEx) GetProcAddress(GetModuleHandleA("ntdll.dll"),
    "RtlRandomEx");
    ULONGLONG result = 31;
    srand(/* some seed value */);
    for (int i = 0; i < 100; ++i)
    {
        ULONG seed = (result ^ 7) & ULONG_MAX;
        int r = rand() | seed;
        result = RtlRandomEx(&seed);
        std::cout << result << std::endl;
    }
    std::cin.get();
}
```

The EDX register changes drastically every iteration.

```
Quote:01384587 8B F4    mov    esi,esp
01384589 8D 45 CC    lea    eax,[ebp-34h]
0138458C 50          push   eax
0138458D FF 55 F4    call   dword ptr [RtlRandomEx]
01384590 3B F4      cmp    esi,esp
01384592 E8 9D CD FF  call   __RTC_CheckEsp (01381334h)
01384597 89 45 E4    mov    dword ptr [result],eax
0138459A 89 55 E8    mov    dword ptr [ebp-18h],edx
```

Kind of a mix and match game however..

Masm4ever - January 30, 2014, 5:04 am

(Jan 30, 2014 04:10 AM)IRP_MJ_SCSI Wrote: [\[CITE\]](#) It looks like i made an error with my solution in trying to do everything in the cast.

Actually, your old solution works - on the right OS!

As mentioned above, it turns out that the XP version returns random numbers in edx::eax, while the Win7 RtlRandomEx returns a constant edx. And so I saw garbage when testing your code on Win7. Apologies for not correcting this above.

This is OK for XP but not Win7:

```
Code #include <iostream>
#include <Windows.h>
```

```
typedef ULONGLONG(__stdcall *_RtlRandomEx)(PULONG seed);
```

```
int main()
{
    _RtlRandomEx RtlRandomEx = (_RtlRandomEx)GetProcAddress(GetModuleHandleA("ntdll.dll"),
```

```
"RtlRandomEx");  
    ULONG seed = 123;  
    for (int i = 0; i < 10; ++i)  
        std::cout << RtlRandomEx(&seed) << std::endl;  
}
```