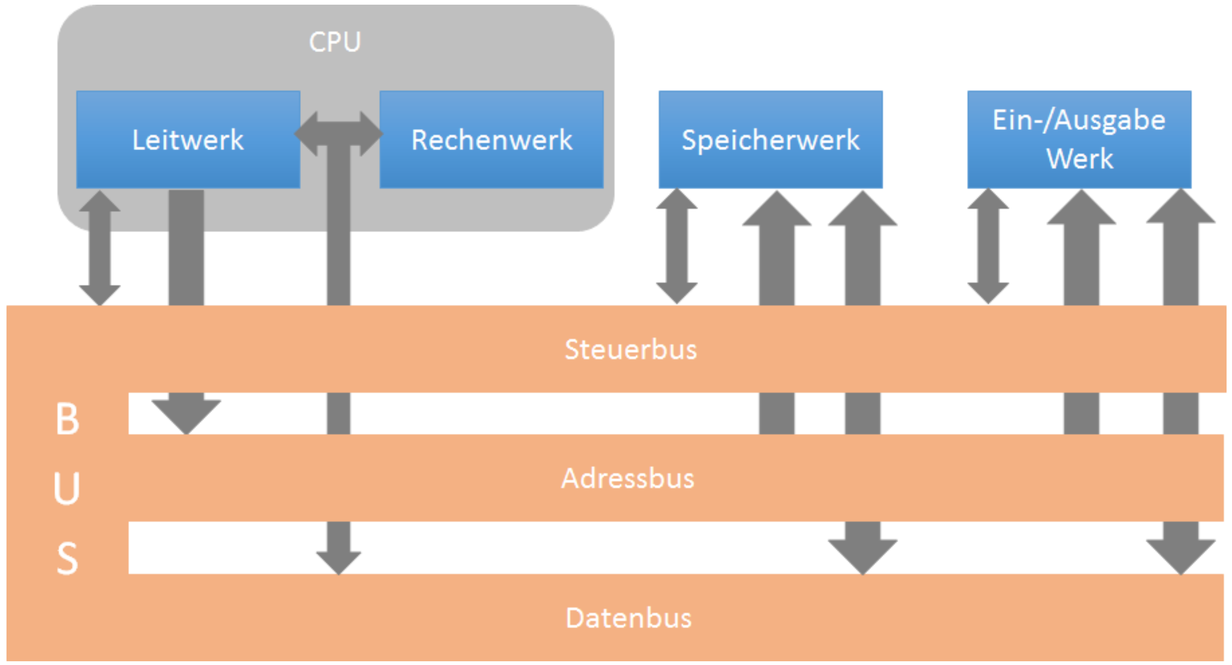
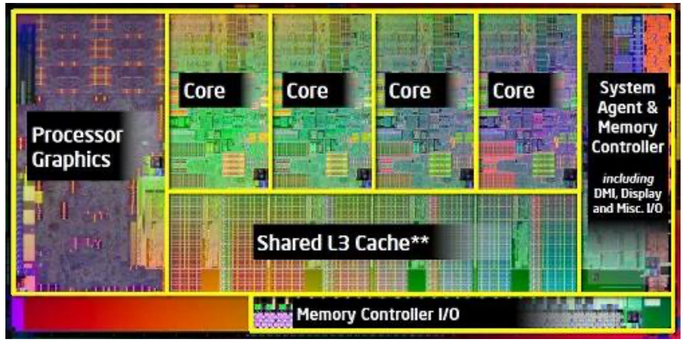
CPU

CPU

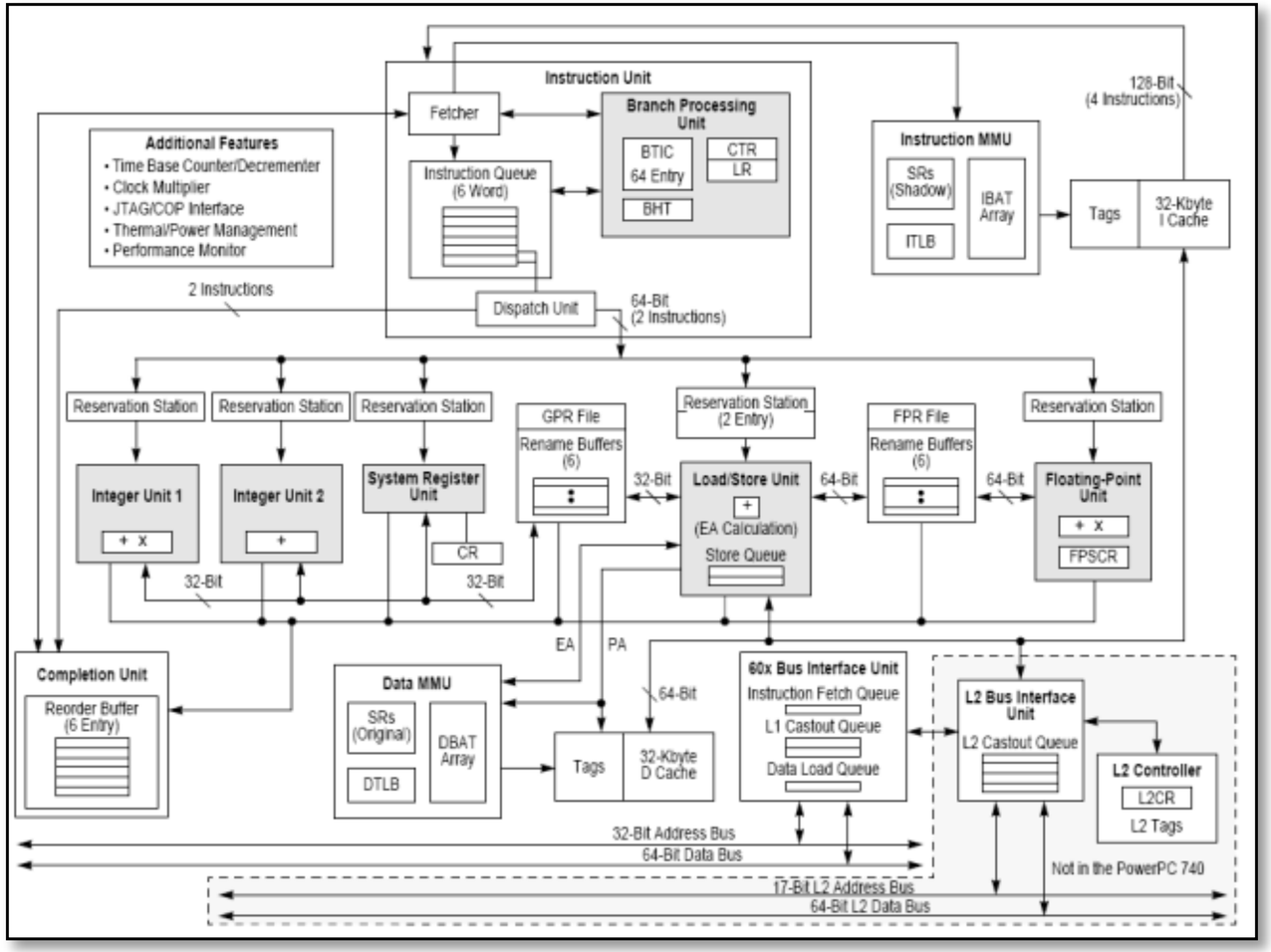
- Central Processing Unit (CPU)
- Versteht nur 0 und 1
- Führt Berechnungen und logische Operationen aus
- Holt sich die Daten und Befehle aus dem Speicher und speichert die Ergebnisse wieder darin ab



CPU

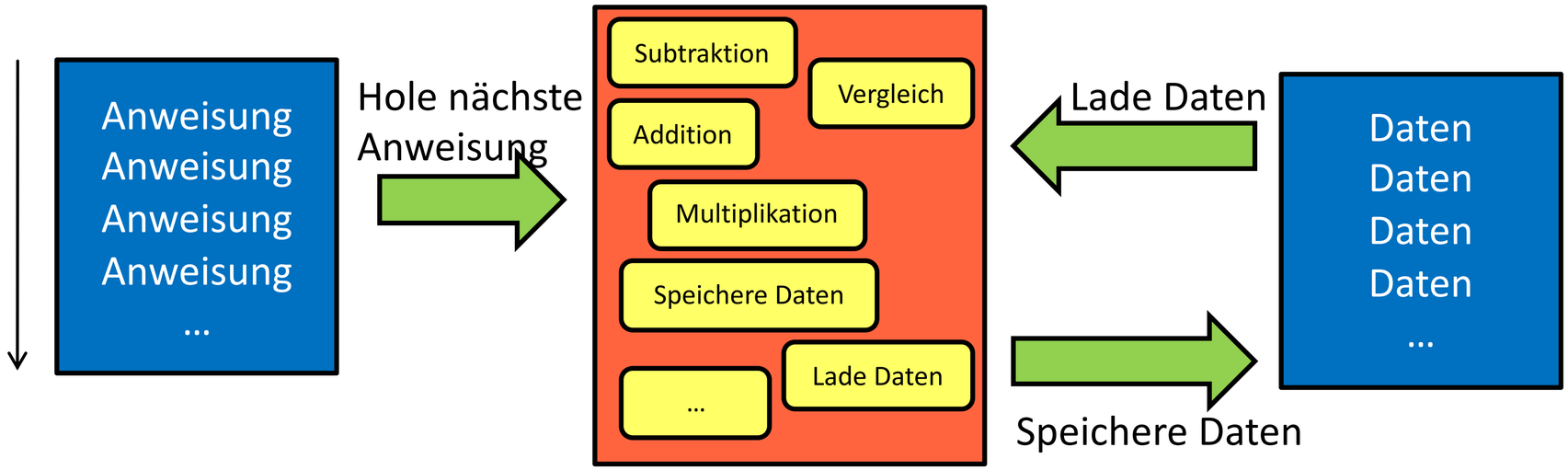


- Da die CPU universell aufgebaut ist, muss nicht für jedes Problem eine eigene Schaltung produziert werden
- Durch Programme lassen sich unterschiedliche Probleme lösen
- Die Grundfunktionen werden zur Verfügung gestellt
  - z.B. Addition, Multiplikation, usw.



Ablauf in der CPU

- Schrittweise werden Anweisungen aus dem Speicher geholt
  - Anweisungen sagen, welche Funktion anzuwenden ist
  - Anweisungen sagen, mit welchen Operanden und wohin mit dem Ergebnis



Typische Grundfunktionen

- Arithmetische Funktionen: Addition, Subtraktion, Multiplikation, Division, …
- Logische Funktionen: AND, OR, NOT, Schiebebefehle
- Speicherzugriffsfunktionen
- Bedingungen: Wenn c gilt mache weiter mit Anweisung X sonst mit Anweisung Y
- Sonstige Funktionen (z.B. Interaktion mit der Umgebung, etc.)
- Spezifische Organisation und Kodierungen von Speicherwörtern für Anweisungen (Hersteller-spezifisch)

Programmieren

- Aus den Grundfunktionen Programme bauen, welche ein gewünschtes Problem lösen
- Vorgabe: Grundfunktionen (je nach CPU unterschiedlich)
- Befehle, bzw. Programm liegt im Speicher und wird dann von der CPU geholt und ausgeführt.
- Das Programm kann durch Eingaben beeinflusst werden und dadurch erfolgt die gewünschte Ausgabe.

Programmiersprachen

Programm quadriere(int eingangsbox):

001110010101001010001010100100100100010001111110101010101010
101010100010101010101010111111010101010010101010101010101010
010101010101001010110101010101001010101000000100010100101 0
111111101010100101010101011101010101000010111101010101010
010100100101010001010010010101111111110101011101011111 0
100000101101010101000010101110100010001011100111001010101001
010001010100100100100010001111110101010101010101010100010101
010101010111111010101010010101010101010101001010101010101001
010110101010101001010101000000100010100101011111111 0101010
010101010101110101010100001011110101010101010010100100101010
001010010010010101111111110101011101011111010000010110101 0
101000010101110100010001011100111001010100101000101010 0100
100100010001111110101010101010101010100010101010101010111111
010101010010101010101010101010010101010101001010110101010101
001010101000000100010100101011111111010101010010101010101110
101010100001011110101010101001010010010101000101001001 0010
101111111110101011101011111010000010110101010101000010101110
1000100010111

Programmiersprachen

- Sprache zur Programmierung sollte
  - Für Menschen mit vernünftigen Aufwand beherrschbar sein (Anlehnung an natürliche Sprache und Sprache der Mathematik)
  - Grundfunktionen einer CPU nutzen
  - Gleichzeitig aber unabhängig von speziellen CPU-Typen sein
  - Umsetzung von Algorithmen in Programme einfach ermöglichen
- Es gibt viele Sprache mit unterschiedlichen Möglichkeiten und Zielsetzungen

Programmiersprachen

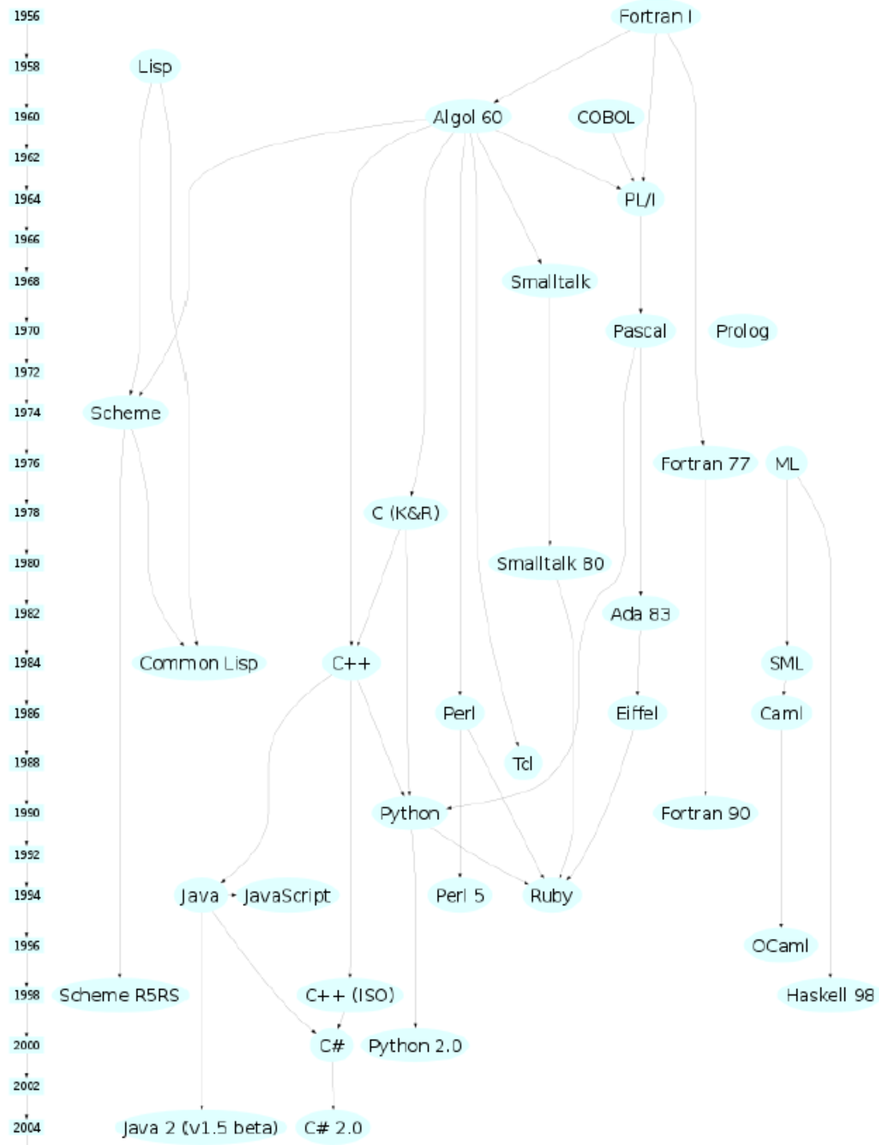
- **Maschinen-Sprachen**
  - CPU-spezifische Sequenzen aus „0“ und „1“
  - Beispiel: 0111011000001100 für „addiere 2 Werte“
  - Wird heutzutage nicht mehr vom Programmierer verwendet
- **Assembler-Sprachen**
  - CPU-spezifische Sprache über den Grundfunktionen, die die CPU anbietet
  - Beispiel: addx r1,#12,r2 für „addiere 2 Werte“
  - Wird für spezielle Fälle verwendet (z.B. für Gerätetreiber, Optimierungen)
- **Hochsprachen**
  - CPU-unabhängige Sprachen mit reichhaltigeren Sprachkonzepten
  - Beispiel: x = y + 12 für „addiere 2 Werte“
  - Standard in industrieller Anwendung

Hochsprachen

- **Logische Sprachen**
  - Prolog
  - Wird selten für bestimmte Problemstellungen benutzt, meist im akademischen Umfeld
- **Funktionale Sprachen**
  - Haskell, XSLT
  - Wird z.B. im Umfeld von XML verwendet. Oder um fehlerfreier zu programmieren
- **Imperative, Prozedurale Sprachen**
  - C, PASCAL
  - Wird sehr häufig im HW-näheren Bereich genutzt (Steuergerätesoftware)
- **Objektorientierte Sprachen**
  - C++, C#, Java
  - Wird sehr häufig für Anwendungsentwicklung benutzt (vorwiegend für Anwendungen auf Desktop-PCs)

Hochsprachen

- Hochsprache „C“ ist Basis für alle gebräuchlichen Sprachen im industriellen Kontext (prozedurale, imperative Sprache)
- PG1 vermittelt C
- Objektorientierung ist Weiterentwicklung und auch sehr gebräuchlich
- PG2 vermittelt C++, JAVA oder C#



The RedMonk Programming Language Rankings: January 2021

- Daten aus Github und Stack Overflow
- nicht statistisch valide

1 JavaScript	11 Swift
2 Python	12 R
3 Java	13 Objective-C
4 PHP	14 Shell
5 C#	14 Scala
5 C++	16 Go
5 CSS	17 PowerShell
8 TypeScript	18 Kotlin
9 Ruby	19 Rust
10 C	19 Perl

Eigenschaften von C

- Die Sprache ist klein, die Möglichkeiten sind groß
  - C hängt eng mit dem Betriebssystem UNIX zusammen (Unix bzw. Linux sind in C geschrieben)
  - Viele sehr gute Compiler verfügbar (Effiziente Maschinenprogramme!)
  - Portierungsaufwand (=Ablauffähigkeit auf verschiedenen Plattformen) lässt sich für Kenner relativ gering halten
  - C ist Hochsprache, erlaubt aber auch Maschinen-nahes Programmieren
  - C als Basis für C++, Java, JavaScript, C#
- Aber:
  - Fehleranfällige, manchmal schwer lesbare Programme
  - Programmierung verlangt Disziplin des Programmierers
  - „C –Das Rasiermesser ohne Handgriff“

C in der Vorlesung

- Grundlegende Konstrukte der Sprache werden vermittelt
  - Datenbehandlung
  - Kontrollstrukturen
  - Datenstrukturen und einfache Algorithmen
- Benutzen an manchen Stellen Sprachelemente von C++ (für Einsteiger leichter erlernbar)
  - Unterschiede werden im Laufe der VL diskutiert
  - Das ist erlaubt, weil wir grundsätzlich C++-Compiler verwenden und C++ im Prinzip eine Obermenge von C ist

Demo: das erste C Programm

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    // insert code here...
    printf("Hello, World!\n");
    return 0;
}

```

Demo: das erste C Programm

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    // insert code here...
    printf("Hello, World!\n");
    return 0;
}

```

#include <*name*>

Datei *name* mit übersetzen

name muss im Standardverzeichnis des Compilers für Bibliotheken zu finden sein

#include "*name*"

name muss im aktuellen Verzeichnis zu finden sein

Demo: das erste C Programm

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    // insert code here...
    printf("Hello, World!\n");
    return 0;
}

```

int main()

Das Betriebssystem ruft beim starten des Programms die main Funktion auf.

int ist der Rückgabewert der Funktion an das Betriebssystem

Demo: das erste C Programm

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    // insert code here...
    printf("Hello, World!\n");
    return 0;
}

```

int argc, const char * argv[ ]

argc ist die Anzahl der Übergabeparameter, welche dem Programm beim Starten übergeben wurden

argv[ ] zeigt auf die Parameter

Demo: das erste C Programm

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    // insert code here...
    printf("Hello, World!\n");
    return 0;
}

```

// alles hinter "//" ist ein Kommentar (in der gleichen Zeile)

/* Alles dazwischen ist ein Kommentar (kann über mehrere Zeilen gehen) */

/* */ ist aus C

// kommt aus der Erweiterung C++

Demo: das erste C Programm

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    // insert code here...
    printf("Hello, World!\n");
    return 0;
}

```

printf("Hello, World!\n");

Die Funktion printf(); gibt Daten aus

"Hello, World!\n" ist eine Zeichenkette

\n bedeutet eine neue Zeile

Demo: das erste C Programm

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    // insert code here...
    printf("Hello, World!\n");
    return 0;
}

```

return 0;

return beendet die Funktion und gibt 0 zurück (im Fall von main an das Betriebssystem)

Demo: das erste C Programm

```c
#include <stdio.h>

int main(int argc, const char * argv[]) {
    // insert code here...
    printf("Hello, World!\n");
    return 0;
}

```

{}

{ Blockbeginn und Ende }

Wie schreibt man „gute“ Programme?

- C ist eine sehr leistungsfähige Sprache
- Nachteil:
  - Manchmal Fehleranfällig (insb. bei Verwendung von Zeigern)
  - Man kann unglaublich unübersichtlichen Code schreiben
- Vorgehensweise:
  - Schreiben Sie den Quelltext übersichtlich (Formatierungsregeln)
  - Kommentieren Sie den Quelltext
  - Verwenden Sie möglichst einfache Konstrukte
  - Gliedern Sie den Algorithmus in lesbare Abschnitte

Wie schreibt man „gute“ Programme?

```
// Quadriert eine übergebene positive ganze Zahl (eingangsbox), liefert das Ergebnis
int quadrieren(int eingangsbox)
{
    int ausgangsbox ;
    int merkbox = eingangsbox; /* übergebene Zahnräder merken, Alg. Schritt 2 */
    if (eingangsbox==0) // Wenn 0 Zahnräder übergeben, 0 Zahnräder zurück, Alg. Schritt
1
      ausgangsbox = 0;
    else
    {
      ausgangsbox = 0;
      while (eingangsbox!=0) // Solange Eingangsbox nicht leer, Alg. Schritt 3
      {
        ausgangsbox = ausgangsbox + merkbox; // Füge übergebene Anzahl hinzu
        eingangsbox = eingangsbox – 1; // entferne Zahnrad
      }
    }
    return ausgangsbox; // Ende des Alg, Schritt 4
}
```

Wie schreibt man „gute“ Programme?

```
// Quadriert eine übergebene positive ganze Zahl (eingangsbox), liefert das Ergebnis
int quadrieren(int eingangsbox)
{
    int ausgangsbox ;
    int merkbox = eingangsbox; /* übergebene Zahnräder merken, Alg. Schritt 2 */
    if (eingangsbox==0) // Wenn 0 Zahnräder übergeben, 0 Zahnräder zurück, Alg. Schritt
1
      ausgangs
    else
    {
      ausgangsbox = 0;
      while (eingangsbox!=0) // Solange Eingangsbox nicht leer, Alg. Schritt 3
      {
        ausgangsbox = ausgangsbox + merkbox; // Füge übergebene Anzahl hinzu
        eingangsbox = eingangsbox – 1; // entferne Zah
      }
    }
    return ausgangsbox; // Ende des A
}
```

Aussagekräftige Namen verwenden

Formatierung des Quelltextes beachten

Kommentare verwenden

Übersichtliche Anweisungsblöcke

Einfache Operationen verwenden

"Schlechtes" Beispiel

- Schwer bis gar nicht zu lesen und auch für Experten nicht sofort ersichtlich
- https://www.ioccc.org/2020/endoh1/index.html

The International Obfuscated C Code Contest

About the 27th IOCCC | 27th IOCCC Winners | IOCCC home | List of All Winners | The Judges

A 27th IOCCC Winner

Most explosive

Yusuke Endoh
Twitter: @mametter

The code for this entry can be found in prog.c

```
                                #include/**/<time.h>
                                #include <ncurses.h>
                                # include <stdlib.h>
          /**                   */#define O()for(y-=                  !!\
        y;y<H&&               /*...Semi-Automatic.*/y<               p/W+2;\
      y++)for(x=p%    W,x-=!!/*..MineSweeper...*/x;x<W&&    x<p%W+2;x++)
    #define _(x,y)COLOR_##x,COLOR_##y /*  click / (R)estart / (Q)uit  */
  #define Y(n)attrset(COLOR_PAIR(n)),mvprintw(/* IOCCC2019 or IOCCC2020 */
 typedef int I;I*M,W,H,S,C,E,X,T,c,p,q,i,j,k;char G[]=" x",U[256];I F(I p){ I
    r=0,x,y=p/W,q;O()q=y*W+x,r+=M[q]^=p-q?(M[q]&16)<<8:0;return r;}I K(I p
      ,I f,I g){ I x=(g+     f/256)%16-(f+g/256)%16,y=p/W,c=0,n=g/4096
       ,m=x==n?0:x==g           /16%16-f/16%16-n?256:-1; if(m+1)O()if
      ((4368&M[n=y*W             +x])==4112){ M[c=1,n]=(M[n]&~16)|m; }
     return c;}void              D(){I p,k,o=0,n=C,m=0,q=0;if(LINES-1<H
    ||COLS/2<W)clear             (),Y(4)LINES/2,COLS/2-16,"Make the ter\
minal bigger!");else{for         (p=0;p<S;o+=k==3,Y(k)p/W+1,p%W*2,G),p++)G[1]=""
"_*!..12345678"[k=E?256&M[p     ]?n--,2:E-2||M[p]%2<1?M[p]&16?q=p,m++,3:4+F(p)%16:
1:3];k=T+time(0);T=o||T>=0||E-1?T:k;k=T<0?k:T;Y(7)0,0,"%03d%*s%03d",n>999?999:n,W*
2-6,"",k>999?999:k);Y(9)0,W-1,E>1?"X-(":E-1||o?":-)":"8-)");M[q]|=256*(n==m&&n); }
refresh();}short B[]={_(RED,BLACK),_(WHITE,BLUE),_(GREEN,RED),_(MAGENTA,YELLOW),_(
CYAN,RED)};I main(I A,char**V){MEVENT e;FILE*f;srand(time(0));initscr();for(start\
_color();X<12;X++){init_pair(X+1,B[X&&X<10?X-1:2],B[X?X<3?2:1:0]);}noecho();cbreak
();timeout(9);curs_set(0);keypad(stdscr,TRUE);for(mousemask(BUTTON1_CLICKED|BUTTO\
N1_RELEASED,0);;){S=A<2?f=0,W=COLS/2,H=LINES-1,C=W*H/5,0:fscanf(f=fopen(V[A-1],"r"
),"%d %d %d",&W,&H,&C)>3; ;S+=W*H;M=realloc(M,S*sizeof(I)*2);for(i=0
;i<S;i++)!f?M[i]=i,i&&(k=M[j=rand()%i],M[j]=M[i],M[i]=k):fscanf(f,
"%d",M+i);if(f)fclose(f);T=E=X=0;for(clear();D(),c=getch(),c-'r'
&&(c-KEY_RESIZE||E);){ if(c=='q'){ return(endwin(),0); }if(c==
KEY_MOUSE&&getmouse(&e)==OK&&e.x/2<W&&e.y<=H){if(!e.y&&(W-2<e.x&&
e.x<W+2)){break;}p=e.x/2+e.y*W-W;if(p>=0){if(!E){for(i=0;i<S;i++)M[S+M
[i]]=i,M[i]=16+(M[i]<C);C-=M[p]&1;M[p]=16;E=1;T=-time(0);}if(E<2)M[p]&=(M[p]
&257)==1?T+=time(0),E=2,273:257;}}for(p=0;p<S&&E==1;M[p++]&=273){}for(i=
(X+S-1)%S;E==1&&i!=X;X=(X+1)%S){if(!(M[p=M[X+S]]&272)){if(K(p,c=F(p)
,0)){goto N;}   for(k=p/W-2,k=k<0?0:k;k<p/W+3&&k   <H;k++)for(j=
p%W-2,j          =j<0?0:j;j<W&&j<p%W+3;)if          (!(M[q=
k*W                +j++]&272)){ if(K(p,              c,F
                   (q))){ goto N; }F(q)
                   ; }F(p); }}N:; } } }
                   /*(c)Yusukse Endoh*/
```