

**تشخیص تصویر امنیتی سیستم مدیریت آموزشی سما**

نگارش:

**فراز حسین خانی**

**چکیده :**

هدف از ساخت و طراحی این پروژه آشکار ساختن ضعف امنیتی سامانه مدیریت آموزشی سما در مقابل روش های مختلف کرک شدن نام های کاربری و کلمه های عبور کارشناسان ، استادان و دانشجویان عضو این سامانه میباشد چرا که تنها دیوار دفاعی در مقابل چنین حملاتی یک تصویر امنیتی یا همان کپچا می باشد که در این پروژه نشان داده شده است که میتوان با برنامه ای در کسری از ثانیه آن را تشخیص داد. هدف از انتخاب زبان هسکل برای این منظور، معرفی این زبان بی همتا و پیچیده بود. زبان هسکل یک زبان تابعی خالص بوده که یک زبان همه منظوره بشمار می آید و می توان از آن برای توسعه هر نوع نرم افزاری استفاده نمود.

**کلمات** **کلیدی**: تصوير امنیتی، سامانه مديريت آموزشی، زبان برنامه نويسی هسکل

تقدیر و تشکر از سازندگان زبان برنامه نویسی هسکل

فهرست مطالب

[**مقدمه :** 7](#_Toc489834145)

[**فصل اول : زبان هسکل** 8](#_Toc489834146)

[**1 – 1 زبان برنامه نویسی هسکل :** 8](#_Toc489834147)

[**1 – 2 تاریخچه** 9](#_Toc489834148)

[**1 – 3 نسخه های مختلف هسکل** 9](#_Toc489834149)

[**هسکل ۱.۰ تا ۱.۴** 9](#_Toc489834150)

[**هسکل ۹۸** 9](#_Toc489834151)

[**هسکل 2010** 9](#_Toc489834152)

[**1 – 4 ویژگی ها** 10](#_Toc489834153)

[**1 – 5 مثال‌ها** 11](#_Toc489834154)

[**1 – 6 پیاده‌سازی‌ها** 13](#_Toc489834155)

[- ) Glasgow Haskell Compiler (GHC 13](#_Toc489834156)

[- Gofer 13](#_Toc489834157)

[- HBC 13](#_Toc489834158)

[- Helium 13](#_Toc489834159)

[- Hugs, the Haskell User's Gofer System, 13](#_Toc489834160)

[- Jhc 13](#_Toc489834161)

[- nhc98 14](#_Toc489834162)

[- Yhc, the York Haskell Compiler 14](#_Toc489834163)

[**1 – 7 کتابخانه‌ها** 14](#_Toc489834164)

[- Hackage 14](#_Toc489834165)

[**فصل دوم : تعاریف و توابع اولیه** 14](#_Toc489834166)

[**2 – 1 تحلیل اولیه** 14](#_Toc489834167)

[**2 – 2 تشخیص پیکسل ها** 16](#_Toc489834168)

[**2 – 3 تابع بازگشتی پر کردن لیست پیکسل ها :** 20](#_Toc489834169)

[**فصل سوم : توابع اصلی و ساخار کلی** 21](#_Toc489834170)

[**3 – 1 تابع main :** 21](#_Toc489834171)

[**3 – 2 تابع delzero :** 22](#_Toc489834172)

[**3 – 3 تابع match :** 24](#_Toc489834173)

[**3 – 4 توابع بررسی هر کاراکتر و شیوه عملکرد آن ها :** 25](#_Toc489834174)

[**فصل چهارم : ابزار ها و برنامه های استفاده شده** 27](#_Toc489834175)

[**4 – 1 برنامه ای برای تبدیل تصویر یک کاراکتر به لیست صفر و یک های معادل آن** 27](#_Toc489834176)

[**نتیجه گیری :** 28](#_Toc489834177)

[**منابع و مأخذ :** 29](#_Toc489834178)

[**پیوست :** 30](#_Toc489834179)

# **مقدمه :**

هدف از ساخت و طراحی این پروژه آشکار ساختن ضعف امنیتی سامانه مدیریت آموزشی سما در مقابل روش های مختلف کرک شدن نام های کاربری و کلمه های عبور کارشناسان ، استادان و دانشجویان عضو این سامانه میباشد چرا که تنها دیوار دفاعی در مقابل چنین حملاتی یک تصویر امنیتی یا همان کپچا[[1]](#footnote-1) می باشد که در این پروژه نشان داده شده است که میتوان با برنامه ای در کسری از ثانیه آن را تشخیص داد. سیستم سما دارای یک سرویس به نام تشخیص آدرس سخت افزاری میباشد که با نصب آن بر روی سیستم خود میتوانید یک مرحله امنیتی دیگر به حساب خود اضافه کنید به این صورت که حتی با در دست داشتن حساب کاربری و کلمه عبور برای وارد شدن به حساب کارت شبکه شما نیز باشد معتبر باشد. با این حال اندک کاربرانی از این سرویس استفاده می کنند.

بدست آوردن نام کاربری اعضا سامانه کاری دشوار نیست چرا که نام کاربری اعضا استاد اعدادی دو یا سه رقمی بوده و نام کاربری دانشجویان همان شماره دانشجویی میباشد، اما کلمه های عبور به صورت پیشفرض شماره ملی اعضا بوده هرچند ممکن است این کلمه در آینه توسط خود اعضا تغییر کند، اما با این وجود افراد معمولا از کلمه های عبوری استفداه میکنند که فقط از اعداد تشکیل شده و معمولا از 8 تا 12 رقم تشکیل شده اند. کرک چنین کلمه های عبوری کار دشواری نخواهد بود. ابزاری مانند سنتری تو[[2]](#footnote-2) به راحتی قابلیت کرک چنین حساب هایی را به استفاده کنندگان می دهند و با در دست داشتن یک پراکسی لیست[[3]](#footnote-3) معتبر و یک لیست واژه[[4]](#footnote-4) مناسب یا به اصتلاح دیکشنری که شامل تمام نام های کاربری ممکن با ساختار گفته شده و کلمه های عبور احتمالی که شامل ارقامی با ساختار شماره ملی و شماره های همراه باشد میتوان با صرف چند ساعت زمان با استفاده از برنامه مقابل برای تشخیص تصاویر امنیتی، درصد زیادی از نام های کاربری به همراه کلمه های عبور آنها را کرک کرد.

اما هدف از انتخاب زبان هسکل برای این منظور، معرفی این زبان بی همتا و پیچیده و به نقل قول از میران لیپوواکا "عجیب" بود. بی شک برنامه نویسی به این زبان تجربه ای متفاوت نسبت به تمام زبان های دیگر به ارمغان می آورد. در ادامه ابتدا به معرفی اجمالی زبان هسکل پرداخته و سپس مراحل طراحی پروژه شرح داده خواهد شد.

# **فصل اول : زبان هسکل**

## **1 – 1 زبان برنامه نویسی هسکل :**

پیش از پرداختن به موضوع اصلی پروژه لازم است ابتدا توضیحی راجع به زبان برنامه نویسی هسکل و دلیل انتخاب آن برای انجام این پروژه آورده شود. زبان هسکل یک زبان تابعی خالص[[5]](#footnote-5) بوده که یک زبان همه منظوره بشمار می آید و می توان از آن برای توسعه هر نوع نرم افزاری استفاده نمود. در زبان های برنامه نویسی دستوری تعدادی کد به کامپیوتر داده میشود که کامپیوتر آن ها را اجرا کرده و می تواند وضیعیت خود را تغییر دهد مثلا ممکن است متغییری توسط شما تعریف گردد و سپس در طی عملیاتی مقدار اولیه ای که به آن متغییر دادید تغییر کند. همچنین ساختار های کنترلی مانند انواع حلقه ها ( حلقه فور[[6]](#footnote-6) و حلقه وایل[[7]](#footnote-7) و... ) برای اجرای چند دستور به صورت مکرر وجود دارند (لیپوواکا, 2011).

اما زبان برنامه نویسی تابعی خالص متفاوت است. در این نوع زبان برنامه نویس به جای اعمال دستورات بر کامپیوتر، به کامپیوتر می گوید که کار ها و دستورات در حقیقت چه چیز هایی هستند. برای مثا در تعریف تابع فاکتوریل ما به سیستم می گوییم که فاکتوریل یک عدد حال ضرب تمامی اعداد از یک تا آن عدد در یکدیگر می باشد و همچنین در زبان برنامه نویسی تبعی شما نمی توانید یک متغیر را تعریف کرده و مقدار دهی کنید و سپس مقدار دهی اولیه آن را تغییر دهید، برای مثال اگر متغیری با نام ایکس تعریف کنید و مقدار آن را برابر 5 بگزارید بعداً نمیتوانید مقدار آن را تغییر دهید (لیپوواکا, 2011).

زبان هسکل تنبل است. این بدان معناست که تا زمانی که لازم نباشد تابعی را اجرا نمیکند. این توسط خاصیت شفافیت ارجاعی[[8]](#footnote-8) زبان هسکل امکان پذیر گردیده است. همچنین تنبل بودن هسکل این قابلیت را به این زبان می دهد که زمانی که برنامه نویس تابعی را صدا می زند، این زبان کمترین محاسبات را برای بازگرداندن نتیجه انجام دهد (لیپوواکا, 2011).

## **1 – 2 تاریخچه**

در پی انتشار زبان میراندا توسط Research Software Ltd در سال 1985، علاقه به زبان‌های تابعی تنبل افزایش یافت. تا سال 1987، زبان‌های تابعی خالص بسیاری به‌وجود آمده بودند. از بین اینها، میراندا بیشترین زبانی بود که استفاده می‌شد، ولی برنامه‌ها انحصاری بودند. در کنفرانسی دربارۀ زبان‌های برنامه‌نویسی تابعی و معماری کامپیوتر در پورتلند، اورگن، یک جلسه برگزار شد که در آن شرکت‌کنندگان بر تشکیل یک کمیته برای تعریف استاندارد‌های باز برای زبان‌های این‌چنینی توافق کردند.

## **1 – 3 نسخه های مختلف هسکل**

### **هسکل ۱.۰ تا ۱.۴**

اولین ورژن هسکل[[9]](#footnote-9) در سال ۱۹۹۰ تعریف شد. تلاش های کمیته به مجموعه ای از تعاریف زبان منجر شد (۱.۰، ۱.۱، ۱.۲، ۱.۳، ۱.۴).

### **هسکل ۹۸**

در اواخر سال ۱۹۹۷، این مجموعه تعاریف در هسکل ۹۸ به اوج خود رسید، که قرار شد یک نسخۀ پایدار، حداقلی[[10]](#footnote-10) و قابل حمل از زبان به همراه یک کتابخانه استاندارد برای آموزش و به عنوان پایه‌ای برای گسترش‌های آینده معین کنند و کمیته صریحاً از ایجاد نسخه‌های متفاوت از هسکل ۹۸ به همراه اضافه کردن ویژگی‌های تجربی در این نسخه‌ها استقبال کرد.

در فوریه ۱۹۹۹، استاندارد زبان هسکل ۹۸ به‌صورت عمومی با عنوان گزارش هسکل ۹۸ رسماً منتشر شد. در ژانویه ۲۰۰۳، یک ورژن بازبینی شده با عنوان زبان هسکل ۹۸ و کتابخانه‌هایش : گزارش بازبینی منتشر شد. زبان به رشد سریع خودش ادامه داد که با پیاده‌سازی Glasgow Haskell Compiler به اختصار GHC استاندارد فعلی هسکل ارائه شد.

### **هسکل 2010**

در اوایل سال ۲۰۰۶، فرایند تعریف یک جانشین برای استاندارد هسکل ۹۸، که به‌صورت غیر رسمی به آن هسکل پرایم می‌گفتند، آغاز شد. بنابراین گذاشته شد که هر ساله تعریف زبان بازبینی شود و یک نسخۀ جدید در هر سال تولید شود. اولین بازبینی، که هسکل 2010 نام داشت، در نوامبر 2009 معرفی شد و در جولای 2010 منتشر شد.

هسکل 2010 رابط توابع خارجی[[11]](#footnote-11) را به هسکل اضافه کرد، و اجازه داد تا به زبان‌های برنامه‌نویسی دیگر متصل[[12]](#footnote-12) شود. همچنین تعدادی مشکل در سینتکس را برطرف کرد، و الگوی n-plus-k-patterns را ممنوع کرد که برای همین استفاده از تعریفی به این فرم :

fact (n+1) = (n+1) \* fact n

دیگر مجاز نبود. این تغییرات منجر به ایجاد یک Language-Pragma-Syntax-Extension شد که نیازمند گسترش‌های مشخصی برای زبان هسکل بود. نام این گسترش‌ها در هسکل ۲۰۱۰ به شرح زیر است:

DoAndIfThenElse, HierarchicalModules, EmptyDataDeclarations, FixityResolution, ForeignFunctionInterface, LineCommentSyntax, PatternGuards, RelaxedDependencyAnalysis, LanguagePragma و NoNPlusKPatterns

## **1 – 4 ویژگی ها**

هسکل دارای ویژگی‌های ارزیابی تنبل، تطبیق الگویی، فهم لیست‌ها، تایپ کلس[[13]](#footnote-13)‌ها، و چند ریختی تایپی است. زبانی محضاً تابعی است، به این معنی که به طور کلی توابع در هسکل اثر فرعی ندارند. ساختاری متمایز برای بیان کردن آثار فرعی وجود دارد، که متعامد با تایپ توابع است. یک تابع محض میتواند اثر فرعی ای را برگرداند که متعاقبا اجرا می شود، و توابع ناخالص زبان های دیگر را پیاده سازی می کند.

هسکل یک زبان با تایپ های سخت و ایستا بر پایه استنتاج تایپ هیندلی-میلنر است. نو آوری اصلی هسکل در این مساله اضافه کردن تایپ کلس ها است، که در اصل به عنوان راهی اصولی برای اضافه کردن سربارگذاری به این زبان ساخته شدن اند ولی پس از آن اسفتاده های دیگری یافته هم یافته اند.

ساختاری که آثار فرعی را بیان می کند مثالی از یک مونَد[[14]](#footnote-14) است. مونَد ها چارچوبی کلی هستند که می توانند انواع مختلفی از محاسبات، مانند رفع خطا، غیر قطعی بودن، تجزیه کردن، و حافظه ی معاملاتی نرم افزاری را مدل سازی کنند. مونَد ها به عنوان دیتا تایپ های معمولی تعریف شده اند. اما هسکل برای استفاده از آن تعدادی زیبایی زبانی اراعه می دهد.

هسکل مشخصات باز و منتشر شده دارد که چند پیاده سازی از آن وجود دارد. پیاده سازی اصلی آن کامپایلر باشکوه هسکل گلاسگو[[15]](#footnote-15)(جی اچ سی)، هم تفسیر گر و هم کامپایلر بومی است که در اکثر پلتفورم[[16]](#footnote-16) ها اجرا می شود. جی اچ سی بخاطر تایپ سیستم غنی ای که نو آوری های اخیر همچون دیتا تایپ های جبری عمومی و خانواده های تایپ را در خود جای می دهد مورد ملاحضه است. بازی معیار زبان های کامپیوتری نیز کارایی بالای پیاده سازی همزمانی و همسانی این زبان را به برجستگی نشان میدهد.

اجتماعی فعال و در حال رشد در اطراف این زبان وجود دارد. و بیش از 5400 کتاب خانه و ابزار متن باز سوم شخص در مخزن بسته آنلاین هکیج[[17]](#footnote-17) موجود است.

## **1 – 5 مثال‌ها**

یک مثال ساده که معمولاً برای اثبات دستور زبان تابعی به کار می‌رود، تابع فاکتوریل برای اعداد صحیح نامنفی می‌باشد که درهسکل به صورت زیر نوشته می‌شود

factorial :: Integer -> Integer

factorial 0 = 1

factorial n | n> 0 = n \* factorial (n-1)

یا در یک خط:

factorial n = if n> 0 then n \* factorial (n-1) else 1

این مثال بالا فاکتوریل را به عنوان تابع بازگشتی شرح می‌دهد با یک مبنای پایان بخش (فرض استقرا). تعریف این تابع توسط کدهای هسکل شبیه به گزاری‌های موجود در کتب درسی است. خط اول تابع فاکتوریل نوع تابع را شرح می‌دهد زمانی که اختیاری است یک نوع خوب برای شامل شدن آن مطرح است.

as the function factorial (factorial) has type (::) from integer to integer (Integer -> Integer)

این یک عدد صحیح به عنوان ارگومان می‌گیرد ویک عدد صحیح دیگر بازمی‌گرداند. اگر برنامه نویس نوع تابع را مشخص نکرده باشد این نوع تعریف به طور اتوماتیک استنباط می‌شود. خط دوم اشاره به تطبیق الگو دارد که یک خصوصیت مهم هسکل است. یادداشتی که پارامترهای تابع ان در پرانتز نیستند به وسیله فاصله از دوم جدا شده است. وقتی که ارگومانهای تابع صفر می‌شوند عدد صحیح به صورت یک برگردانده می‌شود. برای همه موارد دیگر خط سوم آزموده شده. یک حایل خط سوم را از اعداد منفی که فاکتوریل آنها تعریف نشده حفظ می‌کند بدون حایل این تابع بدون رسیدن به مورد اصلی از طریق همه اعداد منفی تکرار می‌شود. همانطور که هست تطبیق الگو کامل نیست. اگر عدد صحیح منفی به عنوان ارگومان عبور کند برنامه با یک خطای زمان اجرا شکست می‌خورد. مورد آخر می‌تواند برای این خطای وضعیت چک شود و در عوض یک خطای مناسب چاپ می‌کند.

مانند راه حل‌های گوناگون سری فیبوناچی که در بالا نشان داده شد از یک هم بازگشت برای ساخت یک لیست از اعداد به محض درخواست استفاده می‌کند. شروع از پایه یعنی ۱ و ساختد موارد جدید که نسبت به سایر اعضا حق تقدم دارد آغاز می‌شود. تکین ها-اتم ها- ورودی /خروجی‌ها هسکل به عنوان یک زبان خالص اثرات جانبی توابع را ندارد این یک رقابت برای برنامه‌های واقعی است که در میان بقیه موارد احتیاج پیدا می‌کند به تأثیرات محیط برنامه نویسی هسکل این مشکل را با نوع داده‌ای اتم حل کرده است. سیستم تایپ به ترتیب دستورات امری یک تناسب بر قرار می‌کند. یک مثال نوعی از این سیستم شامل حالات بی‌ثبات همزمانی حافظه تراکنشی مدیریت استثناء و انتشار خطاست ولی اتم‌ها برای رسیدن به بسیاری اهداف مفیدند. هسکل یک ترکیب ویژه برای عبارات اتمی ارائه کرده است پس برنامه‌هایی که اثر جانبی دارند می‌توانند به نوعی شبیه زبان‌های امری نوشته شوند برنامه زیر یک نام را از ورودی دریافت می‌کند و در خروجی یک پیغام خوشامد گویی چاپ می‌کند.

main = do putStrLn "What's your name?"

name <- getLine

putStr ("Hello, " ++ name ++ "!\n")

do عبارات با این اصطلاح کار با اتم‌ها را آسان می‌کنند این عبارت با عبارت قبلی یکسان می‌باشد ولی تقریباً برای نوشتن آسان‌تر و قابل فهم‌تر از De-sugared که از اپراتور اتم به طور مستقیم استفاده می‌کند.

main = putStrLn "What's your name?">>

getLine>>= \ name ->

putStr ("Hello, " ++ name ++ "!\n")

## **1 – 6 پیاده‌سازی‌ها**

### - ) Glasgow Haskell Compiler (GHC

به کد اصلی به تعداد معیارهای اصلی با استفاده از «سی» به عنوان زبان حیاتی جی اچ سی مشهورترین کامپایلر هسکل می‌باشد. هسکل به تعداد نسبتاً زیادی کتابخانه‌های مفید دارد که فقط با جی اچ سی کار می‌کند.

### - Gofer

یک لهجه یا گویش جدید آموزشی هسکل است با ویژگی به نام کلاس‌های سازنده به وسیلهٔ مارک جونز توسعه داده شد.

### - HBC

یکی دیگر از کامپایلر کد اصلی هسکل می‌باشد این کامپایلر به طور فعالانه کار نمی‌کند و توسعه داده نمی‌شود ولی در بعضی موارد قابل استفاده می‌باشد.

### - Helium

یک گویش جدید برای هسکل است. تمرکز روی این لهجه باعث می‌شود که آموخته‌های ما در مورد پیغام‌های خطا روشن‌تر شود. این در همان حالت یک تایپ را کم دارد و اجرای آن با بسیاری از برنامه‌های هسکل ناسازگار است.

### - Hugs, the Haskell User's Gofer System,

یک مترجم بایت کد می‌باشد که باعث کامپایل و اجرای سریع خواهد شد. همچنین با کتابخانه‌های گرافیکی ساده نیز همراه است. برای افرادی که در زبان هسکل مبتدی هستند این مترجم مفید است ولی به معنای یک اجرای ساده و بچگانه نیست بلکه بسیار قابل حمل‌تر در اجرا درون هسکل می‌باشد.

### - Jhc

یک کامپایلر هسکل است که این کامپایلرتاکید بر سرعت و کارایی بالای برنامه‌ها دارد چنانچه تغییراتی در برنامه اعمال شود.

### - nhc98

یک کامپایلر بایت کد هسکل است باید توجه داشت بایت کد اجرای بسیار سریعتری دارد توجه به استفادهٔ کمتر و بهینه تر از حافظه از مزایای این کامپایلر است و یک انتخاب مناسب برای کامپیوترهای قدیمی و کم مصرف است.

### - Yhc, the York Haskell Compiler

یک شاخه از ان اچ سی ۹۸ می‌باشد با اهداف ساده‌تر شدن قابل حمل شدن تأثیر گذار تر شدن. پشتیبانی تمام و کمال برای هت. همچنین ویژگی‌ها یی برای جاوا نیز دارد که اجازه می‌دهد به کار برانی که در محیط شبکه کار می‌کنند هسکل را نیز اجرا کند.

## **1 – 7 کتابخانه‌ها**

از ژانویه ۲۰۰۷ کتابخانه‌ها و موارد کاربردی در یک بسته به نام Hackage جمع شوند یک پایگاه داده ان لاین از برنامهٔ هسکل که از وسیله‌ای به نام بسته کننده cabal استفاده می‌کند. از دسامبر ۲۰۰۸ ۹۱۰ بسته قابل استفاده برای کاربران گذاشته شده است.

### - Hackage

یک نقطه میانی برای توزیع هسکل میبا شد به وسیله cabal فعالیت‌های توسعه آفرینی هسکل به صورت یک قطب در آمده نصب برنامه‌های جدید هسکل به وسیله Hackage و وسایل نصب cabal امکان‌پذیر است. این نصب‌ها بازگشتی و نیاز به وابستگی‌ها یی دارد که اینها با وجود ان در دسترس هستند. امروزه نصب کد هسکل بسیار اسانتر از قبل می‌باشد.

# **فصل دوم : تعاریف و توابع اولیه**

## **2 – 1 تحلیل اولیه**

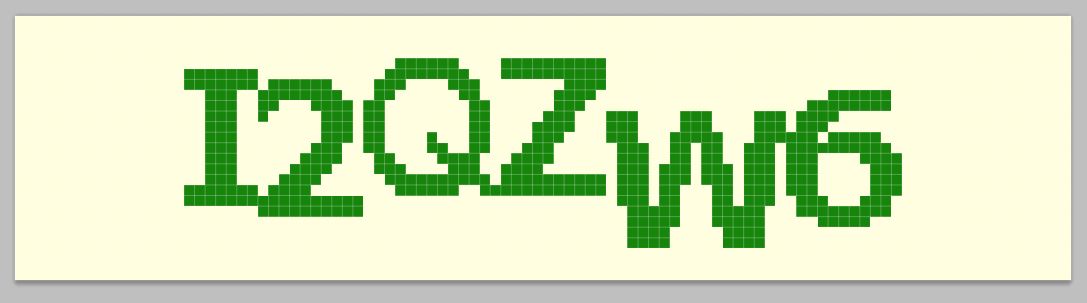
قبل از کد نویسی برنامه لازم بود تا درک کلی از آن چیزی که باید تحلیل شود به وجود آید. تصویر امنیتی سامانه هنگامی نمایش داده می شود که کاربر یک یا تعدادی از اطلاعات خواسته شده را وارد نکند و سامانه برای اینکه از فعالیت ربات های کرک کننده جلوگیری کند همزمان با تقاضای دوباره وارد کردن اطلاعات ورودی، تصویری را نشان داده و از کاربر می خواهد تا کاراکتر ها یا همان حروف و اعداد داخل تصویر را نیز وارد کند. این تصویر همان کپچا است. این تصویر همیشه دارای شش کاراکتر می باشد که این کاراکتر ها می توانند متشکل اعداد از یک تا نه و یا حروف بزرگ الفبای انگلیسی باشند. رنگ کاراکتر ها ممکن است سبز یا آبی تیره و یا آبی روشن باشد و رنگ پس زمینه تصویر ممکن است خاکستری روشن یا خاکستری تیره یا زرد روشن باشد. ابعاد تصویر همیشه 100 در 25 پیکسل[[18]](#footnote-18) است و فرمت آن بیتمپ[[19]](#footnote-19) با عمق رنگ 8 بیت[[20]](#footnote-20) می باشد. ارتفاع تمامی کاراکتر ها 13 پیکسل است اما طول کاراکتر ها متفاوت است. در این کپچا سه خاصیت باعث می شود تا خواندن آن توسط ماشین دشوار شود، اول اینکه رنگ کاراکتر ها و پس زمینه هر بار تغییر می کند، دوم مکان کاراکتر ها و ترتیب چیده شدن آن ها هر بار به صورت تصادفی تغییر می کند و در آخر هیچ فاصله ای بین کاراکتر ها وجود ندارد و هر شش کاراکتر به یکدیگر متصل هستند.

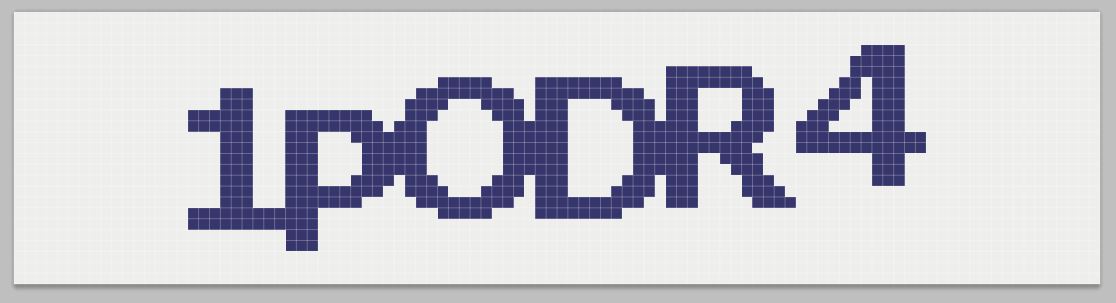
اما تمامی این مسائل به راحتی قابل حل هستند. با ملاک قرار دادن تفاوت رنگ پیکسل های کاراکتر ها با رنگ پیکسل های پس زمینه، مورد اول دیگر اهمیتی نخواهد داشت. موقیت مکانی هر کاراکتر به صورت عمودی فقط تغییر می کند پس با بررسی عمودی پیکسل ها می توان به نحوی با این مورد نیز کنار آمد که در ادامه به صورت دقیق حل ای مسئله به دقت توضیح خواهد داده شد. اما برای حل مسئله سوم باید در نظر داشت که عرض کاراکتر ها متفاوتند و تمامی شش کاراکتر به هم متصل و عرض کلی هر شش کاراکتر در هر بار تولید تصویر امنیتی توسط سامانه، متفاوت خواهد بود. اما با اعمال یک روش ساده می توان این مورد را نیز کنار گذاشت.

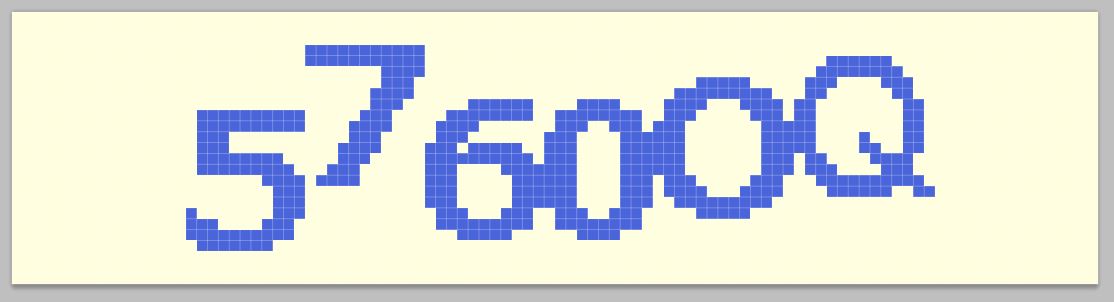
ساختار کلی این برنامه به این صورت است که ابتدا تصویر مورد نظر را از مسیر داده شده بار گزاری می کند و آن را درون یک فضا می ریزد. سپس بعد از تایید صحت تصویر تمام پیکسل ها را بررسی میکند و به جای تمامی پیکسل های پس زمینه یک "صفر" و به جای تمامی پیکسل های کاراکتر ها یک "یک" را درون یک آرایه قرار می دهد. سپس تک تک کاراکتر ها را تشخیص داده و به ترتیب در خروجی چاپ می کند.

در ادامه به توضیح دقیق موارد گفته شده به همراه کد های مربوط به آن ها، می پردازیم.

به چند نمونه از تصاویر کپچا مورد نظر دقت کنید :







**شکل 2 – 1 – 1 : انواع مختلف تصاویر امنیتی سامانه سما**

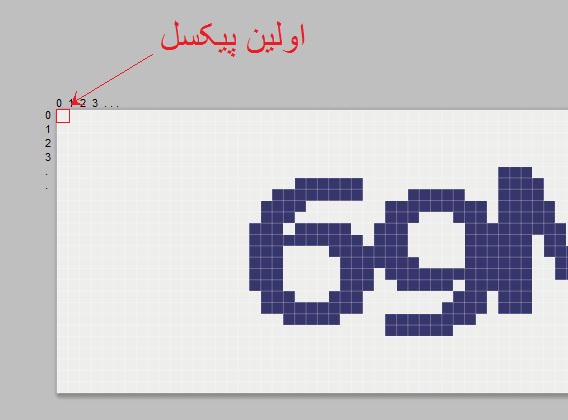
## **2 – 2 تشخیص پیکسل ها**

با توجه ساختار کلی که گفته شد به طور معمول ممکن است یک برنامه نویس که به زبان سی یا C++ یا جاوا برنامه نویسی می کند، برای انجام موارد گفته شده بدین صورت عمل کند که ابتدا یک آرایه دو بعدی با ابعاد 25 در 100 ایجاد کند سپس با دو حلقه فور تو در تو شروع به بررسی تمام پیکسل ها بپردازد و با توجه به رنگ هر پیکسل عدد صف یا یک را درون آرایه قرار دهد.

اما در زبان هسکل آرایه های دو بعدی یا چند بعدی به صورت پیشفرض وجود ندارند و اگر برنامه نویس نیاز به این چنین آرایه هایی داشته باشد مجبور است تا خودش آن ها را پیاده سازی کند. هسکل دارای معادلی برای آرایه می باشد که به آن لیست[[21]](#footnote-21) می گویند. لیست ها می توانند از نوع اینتیجر[[22]](#footnote-22) یا کاراکتر یا هر نوع داده ای دیگر باشند. لیست ها حتی می توانند شامل لیست های هم اندازه باشند و توابع مختلفی برای کار با لیست ها وجود دارد (لیپوواکا, 2011). همچنین در زبان هسکل بهتر است به جای استفاده از حلقه تا حد ممکن از توابغ بازگشتی استفاده شود. این کار باعث کاهش زمان محاسبه و بالا رفتن سرعت اجرایی برنامه می شود. هرچند در زبان هسکل توابعی هستند که مفهوم حلقه را به وسیله آن ها پیاده سازی می کنند مانند تابع مپ[[23]](#footnote-23) یا رپلیکیت[[24]](#footnote-24).

حال می خواهیم تابعی بنویسیم که پیکسل های تصویر را بررسی کند. باید توجه داشت که رنگ اولین پیکسل در تمام تصاویر همرنگ با تمامی پیکسل های پس زمینه می باشد. یعنی پیکسلی که همرنگ با اولین پیکسل باشد جزء پیکسل های کاراکتر ها نمی باشد.

شکل زیر درک بهتری نسبت به این موضوع ارائه می دهد. توجه داشته باشید که شماره بندی پیکسل ها از صفر شروع می شود و اولین پیکسل در گوشه سمت بالا تصویر قرار دارد. برای اشاره به هر پیکسل نیازمند شماره ارتفاع و شماره عرض آن می باشیم.



**شکل 2 – 2 – 1 : مکان و شماره اولین پیکسل**

با توجه به این موضوع می توان به ازای هر پیکسل صفر یا یک را درون آرایه یا همان لیست قرار داد. در ضمن نیازی نیست که حتما یک آرایه دو بعدی در اختیار داشته باشیم تا بتوانیم برنامه را پیاده سازی کنیم. اگر پیکسل ها به ترتیب داخل لیست تک بعدی چیده شود دیگر مشکلی نخواهیم داشت. لیست های دو بعدی و چند بعدی در حقیقت تک بعدی هستند و فقط برای دسترسی و درک بهتر برنامه نویس به آن صورت ساخته شده اند.

پیکسل ها در هر تصویری با هر فرمتی دارای سه پارامتر می باشند که هرکدام مقدار سه رنگ تشکیل دهنده هر پیکسل یعنی قرمز، سبز و آبی را نگه می دارند. برای مقایسه دو پیکسل باید این سه پارامتر آن ها را با هم مقایسه کنیم. دو پیکسل زمانی هم رنگ هستند که این سه پارامتر آن ها با هم برابر باشند. اگر حتی یکی از این سه پارامتر بین دو پیکسل با هم برابر نباشند، آن دو پیکسل هم رنگ نیستند.

برای تعیین اینکه آیا یک پیکسل جزء پیکسل های پس زمینه است یا جزء پیکسل های کاراکتر، باید آن پیکسل را با اولین پیکسل مقایسه کنیم. اگر آن پیکسل هم رنگ با اولین پیکسل بود یعنی هر سه پارامتر آن با هر سه پارامتر پیکسل اول برابر بود پس آن پیکسل جزء پیکسل های پس زمینه است در غیر این صورت جزء پیکسل های کاراکتر هاست.

قطعه کد زیر توابع مربوط انجام این بررسی ها را نشان می دهد:

pix :: DynamicImage -> Int -> Int -> Bool

pix (ImageRGB8 image@(Image w h \_)) i j = iseq (pixelAt image i j ) (pixelAt image 0 0 )

iseq :: PixelRGB8 -> PixelRGB8 -> Bool

iseq (PixelRGB8 r g b) (PixelRGB8 fr fg fb) = r == fr && g == fg && b == fb

برای اشاره به هر پیکسل باید دو پارامتر ارتفاع[[25]](#footnote-25) و عرض[[26]](#footnote-26) که هر دو اعدادی صحیح می باشند، را در دست داشته باشیم. در خط اول از تعریف تابع اول یعنی تابع pix پارامتر های ورودی و خروجی تابع مشخص می شود. در اینجا چنین گفته شده که تابع pix دارای سه پارامتر ورودی و یک خروجی است. همیشه آخرین پارامتر در تعریف نوع پارامتر های یک تابع در زبان هسکل، پارامتر خروجی است. اولین پارامتر ورودی تابع از نوع تصویر پویشی و دومین و سومین ورودی از نوع عدد صحیح می باشد و آخرین پارامتر که همان خروجی تابع است از نوع بولین[[27]](#footnote-27) بوده که نتیجه مقایسه را برمی گرداند. تصویر پویشی که همان تصویر کپچا مورد نظر است و عدد صحیح اول برای عرض پیکسل و عدد صحیح دوم برای ارتفاع پیکسل در تصویر است.

در خط دوم تابع pix گفته شده که یک تصویر با فرمت ImageRGB8 با طول و عرض مشخص به همراه دو عدد صحیح i و j به تابع داده می شود، سپس تابع iseq صدا زده می شود که دو پیکسل را با هم مقایسه می کند.

خط اول تعریف تابع iseq پارامتر های ورودی و خروجی تابع را مشخص می کند که به این معناست که تابع دارا دو ورودی از نوع پیکسل بوده و یک خروجی از نوع بولین.

در ادامه دستورات تابع این چنین است که دو پارامتر های رنگ دو پیکسل را با هم مقایسه می کند و در صورت برابر بودن هر سه پیکسل مقدار صحیح را بر می گرداند.

به ازای هر پیکسل در صورت صحیح بودن خروجی تابع pix مقدار صفر و در صورت غلط بودن مقدار یک درون لیست قرار می گیرد. قرار گرفتن این مقادیر به این گونه است که به آخر لیست اضافه می شوند.

## **2 – 3 تابع بازگشتی پر کردن لیست پیکسل ها :**

پیشتر گفته شد که بهتر است به جای استفاده از حلقه ها در زبان هسکل از توابع بازگشتی استفاده کنیم. البته استاده از توابع بازگشتی در هر زبان برنامه نویسی فرآیندی بهینه است. حال می خوایم ساز و کار این تابع را شرح دهیم. اگر بخواهیم این کار را با حلقه ها انجام دهیم نیازمند دو حلقه به صورت تو در تو هستیم، یکی برای شمارش ستون ها و یکی برای شمارش سطر ها. پس نیاز است تا دو تابع بازگشتی تو در تو را پیاده سازی کنیم. باید توجه داشت که در اینجا می خواهیم تصویر را به صورت عمودی یعنی ستون به ستون بررسی کنیم نه سطر به سطر. بنا براین تابع بازگشتی اول باید ستون ها را بشمارد و تابع دوم سطر های هر ستون را. در هر سطر از یک ستون فقط یک پیکسل وجود دارد، این پیکسل باید با پیکسل اول مقایسه شود و با توجه به نتیجه مقایسه عدد مناسب درون لیست قرار گیرد. ترتیب چیده شدن صفر و یک های مربوت به هر پیکسل باید به این صورت باشد که باید اولین پیکسل اولین ستون، اولین عضو لیست باشد، سپس دومین پیکسل اولین ستون و به همین ترتیب تمام پیکسل های ستون اول درون لیست قرار می گیرد و بعد از آن به ستون بعد می رویم و تمام مراحل برای این ستون نیز انجام می شود. بعد از تمام شدن ستون ها و سطر ها، لیست پایانی بازگردانده می شود.

به قطعه کد زیر توجه کنید :

toBit :: DynamicImage -> [Int] -> Int -> Int -> [Int]

toBit image' bitlist 0 j = bitlist ++ toBitrow image' bitlist 0 j

toBit image' bitlist i j = toBit image' bitlist (i - 1) j ++ toBitrow image' bitlist i j

toBitrow :: DynamicImage -> [Int] -> Int -> Int -> [Int]

toBitrow image' bitlist i 0 = case pix image' i 0 of

True -> bitlist ++ [0]

False -> bitlist ++ [1]

toBitrow image' bitlist i j = toBitrow image' bitlist i (j - 1) ++ case pix image' i j of

True -> bitlist ++ [0]

False -> bitlist ++ [1]

هر دو تابع دارای چهار پارامتر ورودی هستند. یک تصویر، یک لیست از نوع اعداد صحیح و دو عدد صحیح. در پایان هر تابع یک لیست را بر میگرداند. خط اول هر تابع شرط پایه یا شرط صفر آن می باشد. تابع اول برای پیمایش ستون ها و تابع دوم برای پیمایش سطر های هر ستون می باشد.

در تابع دوم دقت کنید که با توجه به نتیجه تابع pix یک 0 یا 1 به انتهای لیست اضافه می شود.

# **فصل سوم : توابع اصلی و ساخار کلی**

## **3 – 1 تابع main :**

کاری که این برنامه انجام می دهد به این صورت است که یک تصویر را به عنوان ورودی می گیرد و آن را به لیستی از صفر و یک ها (طبق توضیحات فصل قبل) تبدیل می کند و سپس هر بار قسمتی از لیست را با لیست کاراکتر های ممکن مقایسه می کند و در صورت تطابق آن کاراکتر را نمایش میدهد و این کار تا زمانی ادامه پیدا می کند که دیگر کاراکتری در لیست باقی نماند.

ابتدا تابع main را به صورت زیر می نویسیم :

main :: IO ()

main = do

[input] <- getArgs

bmp <- readBitmap input

let bitlist = []

case bmp of

Left error -> putStrLn "error"

Right image' -> putStr (match (delzero (toBit image' bitlist 99 24)))

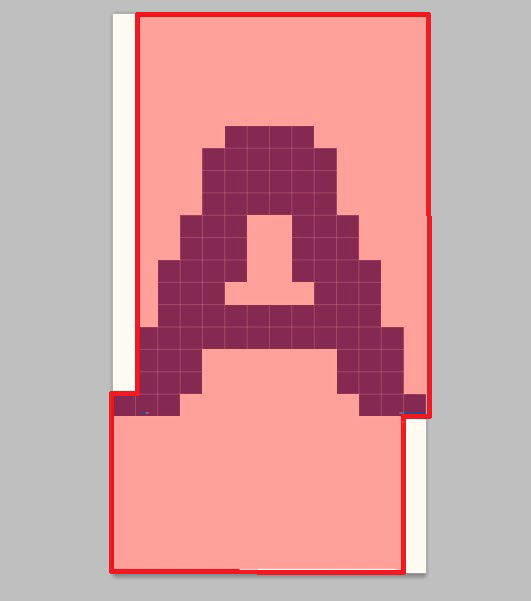
نوع پارامتر تابع IO تعریف شده است که به معنی خروجی و ورودی فیزیکی می باشد. یعنی تابع main چیزی را به خروجی سیستم یا همان کنسول[[28]](#footnote-28) می فرستد. در زبان هسکل اگر بخواهیم چند دستور پشت سر هم اجرا شوند از دستور do استفاده می کنیم. خط سوم برنامه پارامتری را که به همراه برنامه اجرا شده است را داخل متغیر input می دهد. خط بعد تصویر بیتمپ را از متغیر input می خواند و داخل bmp قرار می دهد. خط بعد یک لیت خالی با نام bitlist ایجاد می کند. خط بعد بررسی میکند که آیا bmp یک تصویر است یا نه. اگر مشکلی در خواندن تصویر وجود داشته باشد خط اول عبارت شرطی اجرا می شود که عبارت خطا را در خروجی چاپ می کند، در غیر این صورت خط دوم عبارت یا همان خط آخر اجرا می شود.

خط آخر قسمت اصلی برنامه بوده و تمام توابع اصلی فراخوانی می شوند. معنی این خط اینگونه است، در صورت معتبر بودن فایل به عنوان یک عکس، مقدار بازگشتی توسط تابع match را در خروجی نمایش بده. ورودی تابع match ، خروجی تابع delzero می باشد و ورودی تابع delzero خروجی تابع toBit است.

تابع toBit را قبلا توضیح دادیم. ورودی این تابع همان تصویر اعتبار سنجی شده و لیست خالی ایجاد شده در خط های قبل و طول و عرض تصویر می باشد. در ادامه به توضیح دو تابع دیگر می پردازیم.

## **3 – 2 تابع delzero :**

برای درک دلیل لزوم این تابع ابتدا باید نحوه یافتن کاراکتر ها از درون لیست به طور دقیق توضیح داده شود. هر کاراکتر فارغ از مکانش درون تصویر، هنگام تبدیل شدن به یک لیست (آرایه تک بعدی) همیشه یک دنباله ثابت را ایجاد می کند. با داشتن طول لیستی که هر کاراکتر تولید میکند و مقایسه آن لیست با یک قسمتی از لیست اصلی با همان طول می توان مشخص کرد که آیا آن کاراکتر درون تصویر قرار دارد یا خیر. اگر لیست مربوط به کاراکتر با قسمت جدا شده از لیست اصلی برابر بود پس آن کاراکتر در تصویر وجود دارد و باید آن کاراکتر نمایش داده شده و آن قسمت از لیست حذف شود و ادامه لیست بررسی شود. اما اگر این چنین نبود وجود کاراکتر دیگری را بررسی می کنیم. در تصویر زیر پیکسل های مربوط به لیست هر کاراکتر نشان داده شده اند.



**شکل 3 – 2 – 1 پیکسل های درون لیست کاراکتر A**

دقت کنید که پیکسل های پس زمینه ای که تا قبل از اولین پیکسل کاراکتر اولین ستون قرار دارند داخل لیست قرار داده نشده اند، چرا که به این پیکسل ها برای مقایسه نیاز نخواهیم داشت و اضافه کردن این پیکسل ها باعث این میشود که دیگر نتوانیم حرف A را که در بالاتر یا پایین تر در تصویر قرار داشته باشند را شناسایی کنیم.

تا اولین کاراکتر نیز همیشه پیکسل های پس زمینه زیادی وجود دارند که تعداد آن ها در هر تصویر ممکن است متفاوت باشد و این پیکسل ها در تشخیص کاراکتر ها تاثیری ندارند.

پس با حذف کردن این پیکسل ها از لیست اصلی ( صفر هایی که به جای این پیکسل ها در لیست اصلی قرار دادیم) برای هر بار که می خواهیم یک کاراکتر را شناسایی کنیم، ضروری است و این کار وظیفه تابع delzero می باشد. به قطعه کد زیر توجه کنید :

delzero :: [Int] -> [Int]

delzero [] = []

delzero (x:xs)

| x == 1 = (x:xs)

| otherwise = delzero xs

تابع یک لیست را می گیرد و یک لیست را بر می گرداند. خط دوم شرط صفر تابع بوده که مشخص ی کند ک اگر لیست خالی بود، یک لیست خالی برگردانده شود. به عبارت (x:xs) توجه کنید. این عبارت به کل لیست عشاره می کند که x اولین عنصر این لیست و xs بقیه لیست می باشد. خطوط بعدی شرایط مختلف و نحوه عملکرد تابع تحت هر شرط را نشان می دهند، سه خط آخر یک گارد[[29]](#footnote-29) در زبان هسکل نام دارد (5). شرط اول بیان می کند که اگر اولین عضو لیست برابر عدد یک بود، کل لیست را برگردانده شود. خط دوم همانطور که از معنای آن پیداست بیان می کند که در غیر این صورت تمام لیست به جز عنصر اول به عنوان ورودی به همین تابع نسبت داده شود. همانطور که توضیح داده شد این یک تابع بازگشتی بوده وظیفه آن حذف کردن صفر های اولیه یک لیست است.

## **3 – 3 تابع match :**

یک تابع بازگشتی است برای تشخیص تمام کاراکتر های موجود در یک لیست. یک لیست از نوع اینتیجر می گیرد و یک رشته ( یک لیست از نوع کاراکتر) را بر می گرداند. ساز و کار این تابع چنین است که ابتدا خالی بودن لیست ورودی را بررسی می کند. اگر خالی بود (شرط پایه) یک رشته خالی را برمی گرداند. اما اگر خالی نبود 36 حالت مختلف ممکن را بررسی می کند. این 36 حالت شامل تمام حروف بزرگ زبان انگلیسی و اعداد صفر تا نه می باشد. به کد تابع توجه کنید ( این کد کامل نیست و فقط بخشی از آن برای تشریح بهتر ساختار تابع قرار داده شده است. کد کامل این تابع به همراه کد کامل برنامه در پیوست قرار داده شده است) :

match :: [Int] -> String

match bitlist = case null bitlist of

True -> ""

False | isA bitlist -> 'a' : match (delzero (drop 326 bitlist))

| isB bitlist -> 'b' : match (delzero (drop 261 bitlist))

| isC bitlist -> 'c' : match (delzero (drop 258 bitlist))

| isD bitlist -> 'd' : match (delzero (drop 284 bitlist))

| isE bitlist -> 'e' : match (delzero (drop 213 bitlist))

| isF bitlist -> 'f' : match (delzero (drop 207 bitlist))

در صروت صحت خالی بودن لیست ورودی یک رشته خالی برگردانده می شود. در غیر این صورت به ترتیب تمام حالت های ممکن بررسی می شودند. تابع بررسی و نحوه بررسی و ساز و کار آن در بخش بعدی به طور کامل شرح داده خواهد شد. اما در اینجا اگر تابع بررسی وجود حرف A مقدار صحیح را بازگرداند، کاراکتر a به ابتدای حاصل بررسی کاراکتر بعد اضافه می شود. به علامت : توجه کنید. این علام یک کاراکتر را به ابتدای یک لیست اضافه می کند.

به مقداری که به عنوان ورودی به تابع match فرستاده می شود دقت کنید. تابع drop یک عدد صحیح و یک لیست را دریافت می کند و به تعداد عدد صحیح دریافتی از ابتدای لیست پاک می کند و لیست را بر می گرداند. اما برای هر کاراکتر یک عدد متفاوت در نظر گرفته شده است. این بدان منظور است که هر کاراکتر دنباله منحصر به فردی از پیکسل هاست. و این عدد، تعداد پیکسل هایی است که بین اولین پیکسل کاراکتر (در بررسی عمودی) و آخرین پیکسل کاراتر قرار دادند. تعداد این دنباله برابر با 326 عدد صفر و یک می باشد. به شکل 3 – 2 – 1 توجه کنید. بنابراین در صورت وجود کاراکتر A ، این کاراکتر به رشته حاصل جستجو برای کاراکتر بعدی در لیستی که صفر و یک های مربوط به این کاراکتر حذف شده اند، افزوده می شود و این کار تا خالی شدن لیست ادامه پیدا می کند.

## **3 – 4 توابع بررسی هر کاراکتر و شیوه عملکرد آن ها :**

برای بررسی هر کاراکتر می بایست ابتدا لیست صفر و یک های مربوط به آن کاراکتر را در اختیار داشته باشیم. نحوه بدست آوردن این لیست بسیار ساده است. این لیست در حقیقت شمال صفر و یک های مربوط به هر پیکسل بین پیکسل کاراکتر اول و آ خر است ( شکل 3 – 2 – 1 ). این لیست باید برای تمامی کاراکتر ها بدست آید. همچنین باید تعداد صفر و یک های هر لیست را نیز داشته باشیم. با در اختیار داشتن این دو مورد به سادگی می توانیم با جدا کردن تعداد صفر و یک های هر کاراکتر از لیست اصلی و مقایسه آن ها با لیست کاراکتر مربوط به آن تعداد، وجود آن کاراکتر در لیست را بررسی کنیم. در کد زیر تابع مربوط به بررسی کاراکتر A قرار داده شده است :

isA :: [Int] -> Bool

isA bitlist = take 326 bitlist == [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,1,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,

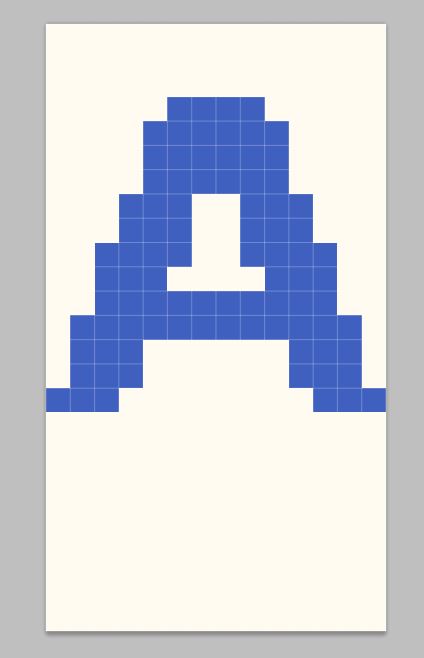
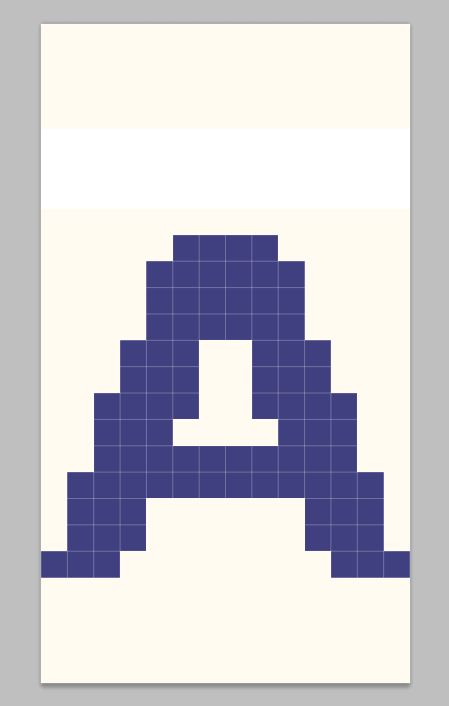
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,1]

تابع take به تعداد عدد صحیح داده شده از لیست ورودی بر میدارد و همان تعداد عناصر را بر می گرداند. سپس این لیست با لیست مربوط به کاراکتر A مقایسه می شود. برای مقایسه از == استفاده می شود که فقط در صورتی درست است که تمام عناصر دو لیست با یک دیگر برابر باشند. لیست بالا یک آرایه تک بعدی است و فقط برای نمایش بهتر به شکل فوق قرار داده شده است.

ممکن است ای سوال پیش آید که آیا این لیست می تواند حرف A را در هر مکان از تصویر پیدا کند و آیا این جست و جو برای یافتن یک حرف A بالا تر و یا پایین تر از حرف A متعلق به لیست بالا، موثر است ؟

در پاسخ این سوال باید گفت که این روش برای هر کاراکتر در هر نقطه از تصویر با هر رنگی موثر است. به دو تصویر بعد توجه کنید :

**شکل 3 – 4 – 1 : تصویر دو حرف A در مکان های مختلف**

دقت کنید که در حقیقت تعداد پیکسل های پس زمینه یا همان صفر ها در هر ستون در هر دو تصویر برابر است. تنها مسئله ای ک مشکل ساز است، صفر های قبل از اولین کاراکتر است که توسط تابع delzero حذف می شوند. برای درک بهتر موضوع این دو تصویر را به عنوان دو آرایه تک بعدی در نظر بگیرید که در آن عناصر، ستون به ستون پشت سر هم قرار گرفته اند. حال اگر صفر های ابتدای هر دو آرایه تا اولین یک را حذف کنیم و هم چنین صفر های پایانی آخرین ستون بعد از آخرین پیکسل کاراکتر یا همان آخرین یک را نیز در نظر نگیریم، این دو لیست برابرند.

# **فصل چهارم : ابزار ها و برنامه های استفاده شده**

برای تحلیل تصاویر و بدست آوردن جزئیات تصاویر و ویرایش آنها برای اهداف ای پروژه از برنامه فوتوشاپ[[30]](#footnote-30) استفاده شده است. این جزئیات شامل فرمت و ابعاد تصاویر است. اما برای یافتن لیست هر کاراکتر و تعداد عناصر هر لیست ابزاری در دست نبود و لازم بود که به صورت دستی و با شماردن تک تک پیکسل ها این لیست ها ایجاد شوند. بنابراین لازم بود برای انجام این کار برنامه ای ساخته شود.

## **4 – 1 برنامه ای برای تبدیل تصویر یک کاراکتر به لیست صفر و یک های معادل آن**

وظیفه این برنامه این است که تصویر یک کاراکتر را به عنوان ورودی گرفته و لیست معادل آن را به همراه تعداد اعضا در خروجی نمایش می دهد. کد این برنامه به صورت زیر است :

main :: IO ()

main = do

[input] <- getArgs

bmp <- readBitmap input

case bmp of

Left error -> putStrLn "error"

Right image -> forM\_ [0..9] $ \i ->

forM\_ [0..24] $ \j ->

case pix image i j of

True -> putStr "0,"

False -> putStr "1,"

where

pix :: DynamicImage -> Int -> Int -> Bool

pix (ImageRGB8 image@(Image w h \_)) a b = iseq (pixelAt image a b ) (pixelAt image 0 0 )

iseq :: PixelRGB8 -> PixelRGB8 -> Bool

iseq (PixelRGB8 r g b) (PixelRGB8 fr fg fb) = r == fr && g == fg && b == fb

همان طور که مشاهده می کنید ساختار این برنامه بسیار شبیه ساختار برنامه اصلی است. با این تفاوت که اینجا برای نشان دادن توانایی زبان هسکل از حلقه forM استفاده شده است.

# **نتیجه گیری :**

در ساخت این برنامه سعی شد تا حد امکان سادگی و بهینه بودن مورد توجه قرار گیرد. هرچند که با وجود این باز هم برنامه جای ساده تر شدن دارد. اما متاسفانه از دید برخی مخاطبان این پروژه کمتر بودن تعداد کد خط های یک برنامه و ساده بودن آن نشانه ضعیف بودن محتوا و پایین بودن سطح کار انجام شده است.

# **منابع و مأخذ :**

Lipovaˇca, M . (2011). ***Learn You A Haskell For Greater Good : a bigenner's guide****.* San Francisco: William Pollock.

# **پیوست :**

import Codec.Picture

import System.Environment

main :: IO ()

main = do

[input] <- getArgs

bmp <- readBitmap input

let bitlist = []

case bmp of

Left error -> putStrLn "error"

Right image' -> putStr (match (delzero (toBit image' bitlist 99 24)))

where

delzero :: [Int] -> [Int]

delzero [] = []

delzero (x:xs)

| x == 1 = (x:xs)

| otherwise = delzero xs

toBit :: DynamicImage -> [Int] -> Int -> Int -> [Int]

toBit image' bitlist 0 j = bitlist ++ toBitrow image' bitlist 0 j

toBit image' bitlist i j = toBit image' bitlist (i - 1) j ++ toBitrow image' bitlist i j

toBitrow :: DynamicImage -> [Int] -> Int -> Int -> [Int]

toBitrow image' bitlist i 0 = case pix image' i 0 of

True -> bitlist ++ [0]

False -> bitlist ++ [1]

toBitrow image' bitlist i j = toBitrow image' bitlist i (j - 1) ++ case pix image' i j of

True -> bitlist ++ [0]

False -> bitlist ++ [1]

pix :: DynamicImage -> Int -> Int -> Bool

pix (ImageRGB8 image@(Image w h \_)) i j = iseq (pixelAt image i j ) (pixelAt image 0 0 )

iseq :: PixelRGB8 -> PixelRGB8 -> Bool

iseq (PixelRGB8 r g b) (PixelRGB8 fr fg fb) = r == fr && g == fg && b == fb

match :: [Int] -> String

match bitlist = case null bitlist of

True -> ""

False | isA bitlist -> 'a' : match (delzero (drop 326 bitlist))

| isB bitlist -> 'b' : match (delzero (drop 261 bitlist))

| isC bitlist -> 'c' : match (delzero (drop 258 bitlist))

| isD bitlist -> 'd' : match (delzero (drop 284 bitlist))

| isE bitlist -> 'e' : match (delzero (drop 213 bitlist))

| isF bitlist -> 'f' : match (delzero (drop 207 bitlist))

| isG bitlist -> 'g' : match (delzero (drop 308 bitlist))

| isH bitlist -> 'h' : match (delzero (drop 263 bitlist))

| isI bitlist -> 'i' : match (delzero (drop 176 bitlist))

| isJ bitlist -> 'j' : match (delzero (drop 175 bitlist))

| isK bitlist -> 'k' : match (delzero (drop 288 bitlist))

| isL bitlist -> 'l' : match (delzero (drop 213 bitlist))

| isM bitlist -> 'm' : match (delzero (drop 312 bitlist))

| isN bitlist -> 'n' : match (delzero (drop 263 bitlist))

| isO bitlist -> 'o' : match (delzero (drop 305 bitlist))

| isP bitlist -> 'p' : match (delzero (drop 239 bitlist))

| isQ bitlist -> 'q' : match (delzero (drop 309 bitlist))

| isR bitlist -> 'r' : match (delzero (drop 288 bitlist))

| isS bitlist -> 's' : match (delzero (drop 278 bitlist))

| isT bitlist -> 't' : match (delzero (drop 252 bitlist))

| isU bitlist -> 'u' : match (delzero (drop 261 bitlist))

| isV bitlist -> 'v' : match (delzero (drop 326 bitlist))

| isW bitlist -> 'w' : match (delzero (drop 403 bitlist))

| isX bitlist -> 'x' : match (delzero (drop 338 bitlist))

| isY bitlist -> 'y' : match (delzero (drop 301 bitlist))

| isZ bitlist -> 'z' : match (delzero (drop 238 bitlist))

| is0 bitlist -> '0' : match (delzero (drop 232 bitlist))

| is1 bitlist -> '1' : match (delzero (drop 211 bitlist))

| is2 bitlist -> '2' : match (delzero (drop 237 bitlist))

| is3 bitlist -> '3' : match (delzero (drop 260 bitlist))

| is4 bitlist -> '4' : match (delzero (drop 278 bitlist))

| is5 bitlist -> '5' : match (delzero (drop 235 bitlist))

| is6 bitlist -> '6' : match (delzero (drop 256 bitlist))

| is7 bitlist -> '7' : match (delzero (drop 253 bitlist))

| is8 bitlist -> '8' : match (delzero (drop 259 bitlist))

| is9 bitlist -> '9' : match (delzero (drop 256 bitlist))

isA :: [Int] -> Bool

isA bitlist = take 326 bitlist == [

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,1,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,1]

isB :: [Int] -> Bool

isB bitlist = take 261 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,

1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1]

isC :: [Int] -> Bool

isC bitlist = take 258 bitlist == [

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,

0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,

0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,1]

isD :: [Int] -> Bool

isD bitlist = take 284 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,

0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,

0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1]

isE :: [Int] -> Bool

isE bitlist = take 213 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,

1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,

0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,

0,0,0,1,1]

isF :: [Int] -> Bool

isF bitlist = take 207 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1]

isG :: [Int] -> Bool

isG bitlist = take 308 bitlist == [

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,

0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,

0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,1,1,1,1]

isH :: [Int] -> Bool

isH bitlist = take 263 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,

1,1,1]

isI :: [Int] -> Bool

isI bitlist = take 176 bitlist == [

1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,

0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1]

isJ :: [Int] -> Bool

isJ bitlist = take 175 bitlist == [

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,0,0,

0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,

0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,

0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1]

isK :: [Int] -> Bool

isK bitlist = take 288 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,

0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,

0,1]

isL :: [Int] -> Bool

isL bitlist = take 213 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,1,1]

isM :: [Int] -> Bool

isM bitlist = take 312 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,

1]

isN :: [Int] -> Bool

isN bitlist = take 263 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,

1,1,1]

isO :: [Int] -> Bool

isO bitlist = take 305 bitlist == [

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,

0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,

0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1]

isP :: [Int] -> Bool

isP bitlist = take 232 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1]

isQ :: [Int] -> Bool

isQ bitlist = take 309 bitlist == [

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,

0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,

0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,1,1,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]

isR :: [Int] -> Bool

isR bitlist = take 288 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,1]

isS :: [Int] -> Bool

isS bitlist = take 278 bitlist == [

1,1,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,

1,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,

1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,

1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,1,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,1,1,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1]

isT :: [Int] -> Bool

isT bitlist = take 252 bitlist == [

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,

1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,

1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1]

isU :: [Int] -> Bool

isU bitlist = take 261 bitlist == [

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,

1]

isV :: [Int] -> Bool

isV bitlist = take 326 bitlist == [

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,1]

isW :: [Int] -> Bool

isW bitlist = take 403 bitlist == [

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,1,1,1]

isX :: [Int] -> Bool

isX bitlist = take 338 bitlist == [

1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1]

isY :: [Int] -> Bool

isY bitlist = take 301 bitlist == [

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]

isZ :: [Int] -> Bool

isZ bitlist = take 238 bitlist == [

1,1,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,1,1,1,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1,

1,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,

1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,

0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,

0,0,1,1]

is0 :: [Int] -> Bool

is0 bitlist = take 232 bitlist == [

1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,

0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,

0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1]

is1 :: [Int] -> Bool

is1 bitlist = take 211 bitlist == [

1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,

1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,

1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,1,1]

is2 :: [Int] -> Bool

is2 bitlist = take 237 bitlist == [

1,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,

1,1,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,1,1]

is3 :: [Int] -> Bool

is3 bitlist = take 260 bitlist == [

1,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,

1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,

0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,

0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,1]

is4 :: [Int] -> Bool

is4 bitlist = take 278 bitlist == [

1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,1,1,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,

0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1]

is5 :: [Int] -> Bool

is5 bitlist = take 235 bitlist == [

1,1,1,1,1,1,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,

1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,

1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,

1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,1,

1]

is6 :: [Int] -> Bool

is6 bitlist = take 256 bitlist == [

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,

1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,0,

0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,1,0,0,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,

0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,

0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,0,0,0,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,1,1,1,1,

0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1]

is7 :: [Int] -> Bool

is7 bitlist = take 253 bitlist == [

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,

1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,

0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,

0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,

0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1]

is8 :: [Int] -> Bool

is8 bitlist = take 259 bitlist == [

1,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,

1,1,1,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,

1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,

1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,

1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,

0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,1,1,1,0,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,1]

is9 :: [Int] -> Bool

is9 bitlist = take 256 bitlist == [

1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,

1,1,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,

1,1,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,

1,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,

1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,

1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,

0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,1,0,1,

1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1]

1. CAPTCHA [↑](#footnote-ref-1)
2. Sentry two [↑](#footnote-ref-2)
3. Proxy list [↑](#footnote-ref-3)
4. Dictionary [↑](#footnote-ref-4)
5. Purely Functional [↑](#footnote-ref-5)
6. For [↑](#footnote-ref-6)
7. While [↑](#footnote-ref-7)
8. Referential Transparency [↑](#footnote-ref-8)
9. "Haskell 1.0" [↑](#footnote-ref-9)
10. Minimal [↑](#footnote-ref-10)
11. Foreign Function Interface [↑](#footnote-ref-11)
12. bind [↑](#footnote-ref-12)
13. Type-class [↑](#footnote-ref-13)
14. Monad [↑](#footnote-ref-14)
15. Glasgow [↑](#footnote-ref-15)
16. Platform [↑](#footnote-ref-16)
17. Hackage [↑](#footnote-ref-17)
18. Pixel [↑](#footnote-ref-18)
19. Bitmap [↑](#footnote-ref-19)
20. Bit [↑](#footnote-ref-20)
21. List [↑](#footnote-ref-21)
22. Integer [↑](#footnote-ref-22)
23. Map [↑](#footnote-ref-23)
24. Replicate [↑](#footnote-ref-24)
25. Height [↑](#footnote-ref-25)
26. Weight [↑](#footnote-ref-26)
27. Boolean [↑](#footnote-ref-27)
28. Console [↑](#footnote-ref-28)
29. Guard [↑](#footnote-ref-29)
30. Photoshop [↑](#footnote-ref-30)